Note!

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

First Edition (June 1994)

This edition applies to the licensed program IBM* ILE* RPG/400* (Program 5763-RG1), Version 3 Release 0 Modification 5, 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 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, you can 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 non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation

Contents

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service 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.

Changes or additions to the text are indicated by a vertical line (|) to the left of the change or addition.

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

This RPG/400 Reference is intended to help you create RPG programs. It contains reference information for the RPG/400 compiler. This RPG/400 Reference primarily documents *general-use programming interfaces and associated guidance information* provided by the RPG/400 compiler.

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

However, this information unit also documents *product-sensitive programming interfaces and associated guidance information*.

Product-sensitive programming interfaces are provided to allow the customer installation to perform tasks such as tailoring, monitoring, modification, tuning, or diagnosis of this IBM product. Use of such interfaces create dependencies on the detailed design or implementation of the IBM product. Product-sensitive interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive programming interfaces and associated guidance information is explicitly identified where it occurs, either by an introductory statement to a chapter or section that is entirely *product-sensitive programming interface and associated guidance information,* or by the following marking:

Product-Sensitive Programming Interface

Description of *product-sensitive programming interface and associated information...*

End of Product-Sensitive Programming Interface

This manual contains small programs which are furnished by IBM as simple examples to provide an illustration. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. All programs contained herein are provided to you 'AS IS'. THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED.

Trademarks and Service Marks

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

Application System/400
400
IBM
Operating System/2
RPG/400
SAA
SQL/400
AS/400
ILE
OS/2
OS/400
Operating System/400
Systems Application Architecture

About This Manual

This manual is a reference for the RPG/400* compiler, the Systems Application Architecture* (SAA*) implementation of the RPG/400 programming language on the Application System/400* (AS/400*) system.

The topics this manual covers include:

- RPG/400 specifications
- RPG/400 indicators
- RPG/400 cycle
- Operation codes
- RPG/400 words with special functions
- Arrays and tables
- Editing numeric fields
- General file considerations
- Double Byte Character Set (DBCS) support.

You may need to refer to other IBM* manuals for more specific information about a particular topic. The *Publications Guide, GC41-9678*, provides information on all the manuals in the AS/400* library.

For a list of related publications, see the “Bibliography” on page 445.

Who Should Use This Manual

This manual is intended for readers who, having read the *RPG/400 User's Guide*, SC09-1816 are interested in more specific and more detailed explanations of RPG/400 code. Some references are made to the System/38 system and its various products and features. In some instances, special reference is made to specific publications, but if no such reference is given, the reader should use the *RPG/400 User's Guide*.

Before you use this manual, you should be familiar with certain information:

- You should know how to use data management support, which allows an application to work with files. This information is contained in *Data Management Guide, SC41-9658*

  The manual includes information on:

  – Fundamental structure and concepts of data management support on the system

  – Data management support for display stations, printers, tapes, and diskettes, as well as spooling support

  – Overrides and file redirection (temporarily making changes to files when an application is run)

  – Copying files by using system commands to copy data from one place to another

  – Tailoring a system using double-byte data.

- You should be familiar with your display station (also known as a workstation), and its controls. There are also some elements of its display and certain keys on the keyboard that are standard regardless of which software system is currently running at the display station, or which hardware system the display station is hooked up to. Some of these keys are:
  - Cursor movement keys
  - Command keys
  - Field exit keys
  - Insert and delete keys
  - The Error Reset key.

  This information is contained in *New User's Guide, SC41-8211*.

  - You should know how to operate your display station when it is hooked up to the IBM AS/400 system and running AS/400 software. This means knowing about the operating system and the Control Language (CL) to do such things as:
    - Sign on and sign off the display station
    - Interact with displays
    - Use Help
    - Enter control commands and procedure commands
    - Call utilities
    - Respond to messages.

  To find out more about control language, refer to:

  - *Programming: Control Language Reference, SC41-0030*
  - *Programming: Control Language Programmer's Guide, SC41-8077*
- You should know how to call and use certain utilities available on the AS/400 System:
  - The Screen Design Aid (SDA) utility used to design and code displays. This information is contained in *Application Development Tools: Screen Design Aid User's Guide and Reference, SC09-1340*.
  - The Source Entry Utility (SEU), which is a full-screen editor you can use to enter and update your source and procedure members. This information is contained in *Application Development Tools: Source Entry Utility User's Guide and Reference, SC09-1338*.
- You should know how to interpret displayed and printed messages. This information is contained in *RPG/400 User's Guide*.
- You should be familiar with the RPG/400 program cycle and how indicators affect the program cycle. See chapters 2 and 3 for more information.

Chapter 1. Introduction to RPG/400

This document describes the RPG/400 programming language.

The valid character set for the RPG/400 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
- The numbers 0 1 2 3 4 5 6 7 8 9
- The characters + – * , . ' & / $ # : @
- The blank character

RPG/400 code is written on a variety of specifications. Each specification has a specific set of functions. See “Common Entries” on page 5 for details on specification types.

The following illustration describes the specifications.

Note

The RPG/400 source program must be entered into the system in the order shown. Extension and Line Counter specifications are the only exceptions to this rule. The order can be E L or L E, but the specifications must be placed between the File Description and Input specifications. Any of the specification types can be absent, but at least one must be present.

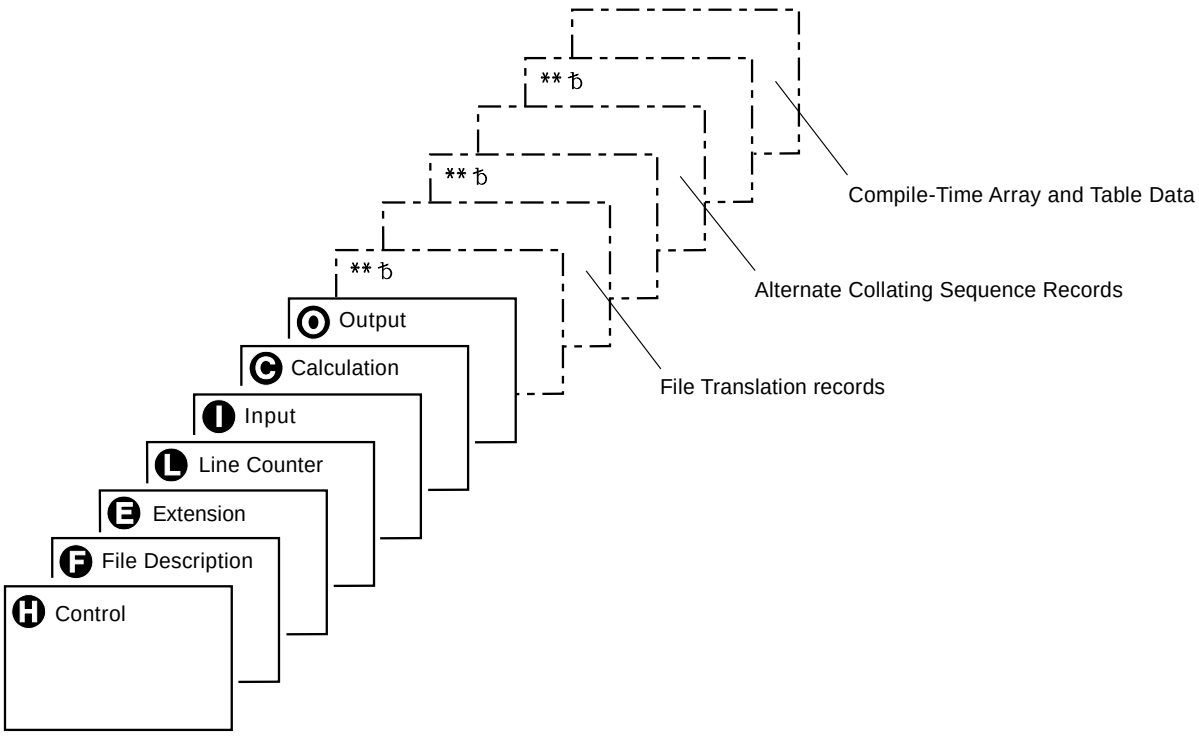


Figure 1. Order of the Types of Specifications in an RPG/400 Source Program

H Control (header) specifications provide information about program generation and running of the compiled program. Refer to Chapter 4, “Control Specifications” for a description of the entries on this specification.

- **F** File description specifications define all files in the program. Refer to Chapter 5, “File Description Specifications” for a description of the entries on this specification.
- **E** Extension specifications describe all arrays and tables and indicate how they are initialized. Refer to Chapter 6, “Extension Specifications” for a description of the entries on this specification.
- **L** Line counter specifications indicate the length of the overflow lines and the forms length to be printed on each page for each printer file in the program. Refer to Chapter 7, “Line Counter Specifications” for a description of the entries on this specification.
- **I** Input specifications describe data structures, named constants, records, and fields in the input files and indicate how the records and fields are used by the program. Refer to Chapter 8, “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 9, “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 10, “Output Specifications” for a description of the entries on this specification.

The RPG/400 language is a position-dependent language. Each entry must start in a specific column. To represent this, each illustration of RPG/400 code will be in listing format with a scale drawn across the top. You may find it helpful to use your RPG/400 Debugging Template (GX21-9129).

This reference contains a detailed description of the individual RPG/400 specifications. Each field and its possible entries are described. Chapter 11, “Operation Codes” describes the operation codes that are coded on the Calculation specification, which is described in Chapter 9, “Calculation Specifications.”

In addition, there is information on indicators, the logic cycle, arrays and tables, edit codes and edit words, file translation, multifile processing, and match fields.

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 arrays, tables, 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
14-74	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/400 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)

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 74 of the /EJECT statement must be blank. /EJECT is not printed on the compiler listing.

/SPACE (Positions 7-12)

Use the compiler directive /SPACE to control line spacing within 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.

If the number specified in positions 14 through 16 is greater than the number of lines remaining on the current page, subsequent specifications begin on a new 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. If position 6 is blank, it is considered to be equal to the preceding specification and the two lines are not skipped.

/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
13-44	Identifies the location of the member to be copied (merged). The format is: libraryname/filename,membername (RPG/400 AS/400 environment) filename.libraryname,membername (RPG III System/38 environment) • A member name must be specified. • If a file name is not specified, QRPGSRC 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.

45-49 Blank
50-80 Comments

Figure 2 shows some examples of the /COPY directive statement.

Note: Programs compiled under the System/38 environment can use the extended naming convention. If extended names are used, each part of the qualified name must be enclosed in quotation marks. Programs compiled under System/38 Environment must use the naming convention *filename.libraryname*.

How the Compiler Recognizes a Compiler /COPY

The treatment of a /COPY directive as a compiler copy or auto report copy depends on which create CL command is used, either CRTRPGPGM or CRTRPTPGM. Use the rules for RPG/400 symbolic names, to specify the library files and member.

1. The CRTRPGPGM command will treat any /COPY directive encountered in the source code as a compiler copy and does not accept the sorting or string replacement functions.
2. The use of CRTRPTPGM will treat any /COPY directive encountered as an auto report copy and will process any sorting or string replacement.

The SEU syntax checker does not distinguish between the two types of copy.

```
C/COPY MBR1 1

I/COPY SRCFIL,MBR2 2

O/COPY SRCLIB/SRCFIL,MBR3 3

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

Figure 2. Examples of the /COPY Compiler Directive Statement

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

Conditions on the Members That Are Copied

If the member being copied from is not a source type file, meaning that the file was not created with the filetype *SRC, then a message is issued. This condition does not stop compilation and the copy function is still processed.

The members being copied can contain any valid RPG/400 specifications except another /COPY directive.

The member being copied can contain a control specification (form type H), which is then processed as if it were part of the source code. Therefore, the first statement in the program can be a /COPY directive pointing to the member containing the control specification. The H specification must still follow the rules governing its

use (refer to Chapter 4, “Control Specifications”) and precede any other RPG/400 specification.

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. Any overrides defined that apply to the specified file and member are ignored.

Records that are copied into the program in this way contain a “+” adjacent to the sequence number field on the left side of the listing, between the sequence number and the form type field.

Sequence Numbering of the Listing after a Compile

The low-order 6 digits of the 8-character sequence number in the listing reflect the original source sequence number of the /COPY member. In this way you can correlate the compiler listing sequence numbers (the last 6 digits) to the source member sequence numbers (in SEU).

The high order 2 digits of the sequence number are made up of the characters A through Z and 0 through 9 in the following order: A, B, C, ..., Z, 1, 2, ..., 9, A0, AA, AB, ..., AZ, A1, A2, ..., A9, B0, BA, ..., ZZ, ..., Z9, 10, ..., 99.

This structure allows for up to 1295 different increments of the high order sequence number.

Each /COPY directive causes the high order 2 characters of the sequence number for all the code lines brought in by this /COPY to be incremented in value.

If the copied code contains specifications for externally defined files, each externally defined file may cause the high-order characters of the sequence number to be incremented in value two more times: once for input specifications, and once for output specifications.

If more than 1295 increments result from /COPY directives and/or externally defined files, a message is issued. A message is also issued if more than 50 externally defined files are specified. Remember that one externally defined file can take up two increments: for input and output specifications.

The low order 6 digits of the sequence number is incremented by one for each I specification generated for an externally defined file.

Note: The /COPY directive is treated as a comment line. Because the compiler may have to read ahead to gather sufficient information about specifications, comments, or /COPY directives, certain lines may appear on the listing out of sequence.

Common Entries

The following entries are common to all RPG specifications:

- Page (positions 1-2) and Line (positions 3-5). These are the equivalent of source line number.
- Specification type (position 6). The following letter codes can be used:

Entry	Specification Type
H	Control
F	File description

E Extension
L Line counter
I Input
C Calculation
O Output

- Program Identification (positions 75-80). Must be blank or program name on the Control specification.
- Comment Statements (* in position 7). On a comment statement or a compiler directive, position 6 may be blank.

Symbolic Names

A symbolic name is a name that uniquely identifies a specific entity in a program. Its purpose is to allow you to access that entity. In the RPG/400 language, symbolic names are used for the following entities:

- Arrays (on page 6)
- Constants (on page 8)
- Data structures (on page 7)
- EXCPTs (exception output records) (on page 7)
- Fields (on page 7)
- Files (on page 7)
- KLISTs (key field lists) (on page 7)
- Labels (on page 7)
- Named constants (on page 9)
- PLISTs (parameter lists) (on page 7)
- Record names (on page 7)
- Subfields (on page 8)
- Subroutines (on page 8)
- Tables (on page 8).

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 and part of the RPG/400 character set.
- The name must be left-adjusted in the entry on the specification form.
- Blanks can fill out the entry but must not be embedded in the name.
- A symbolic name cannot be an RPG/400 reserved word.
- A symbolic name can be 1 to 6 characters.
- A symbolic name must be unique.

Array Names

An array name is a symbolic name assigned to an array. The following additional rule applies to array names:

- An array name cannot begin with the letters TAB.

Data Structure Names

A data structure name is a symbolic name assigned to a data structure. 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 field can be defined as a data structure only once.

EXCPT Names

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

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

Field Names

A field name is a symbolic name assigned to a field in a program. The following additional rules apply to field names:

- A field name can appear on more than one field definition statement if each definition using that name has the same data type (character or numeric), 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).
- A field can be defined in a data structure only once.

File Names

A file name is a symbolic name assigned to a file. The following additional rules apply to file names:

- A file name can contain from 1 to 8 characters.

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 destination point of a GOTO or CABxx 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/400 program:

- A record name can contain from 1 to 8 characters.
- A record name can exist in only one file in the program.

Subfield Names

A subfield name is a symbolic name assigned to a data structure subfield. The following additional rules apply to subfield names:

- A subfield name cannot be specified as the result field on an *ENTRY PLIST parameter.
- A field or an array can appear only once as a subfield name.

Subroutine Names

A subroutine name is a symbolic name assigned to a subroutine. The name is defined in factor 1 of the BEGSR (begin subroutine) operation.

Table Names

A table name is a symbolic name assigned to a table. The following additional rules apply to table names:

- A table name can contain from 3 to 6 characters.
- A table name must begin with the letters TAB.

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 to specify initialization values for data structure subfields on the input specification. A literal is a self-defining constant that can be referred to in a program. A literal can be character, hexadecimal or numeric.

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. 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 cannot be used for arithmetic operations.

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, BITOF, 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'.

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 must not be 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 character (comma or period) used for the decimal notation is determined by the inverted print option specified in position 21 of the control specification.

Named Constants

A named constant is a symbolic name assigned to a character or numeric constant. Named constants are defined on Input specifications. The value of a named constant follows the rules specified for literals. See “Named Constant Specifications” on page 155 for detailed information.

Chapter 2. RPG/400 Program Cycle and Error Handling

The RPG/400 compiler supplies part of the logic for an RPG/400 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/400 specifications in your source program need not explicitly specify when records should be read or written. The RPG/400 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 16 of the file description specifications) and secondary (identified by an S in position 16 of the file description specifications) files indicate input is controlled by the program cycle. A full procedural file (identified by an F in position 16 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 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/400 Program Cycle

Figure 3 on page 12 shows the specific steps in the general flow of the RPG/400 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/400 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, 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 UDATE. 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. Tables and data area structures are written out, and the program ends.

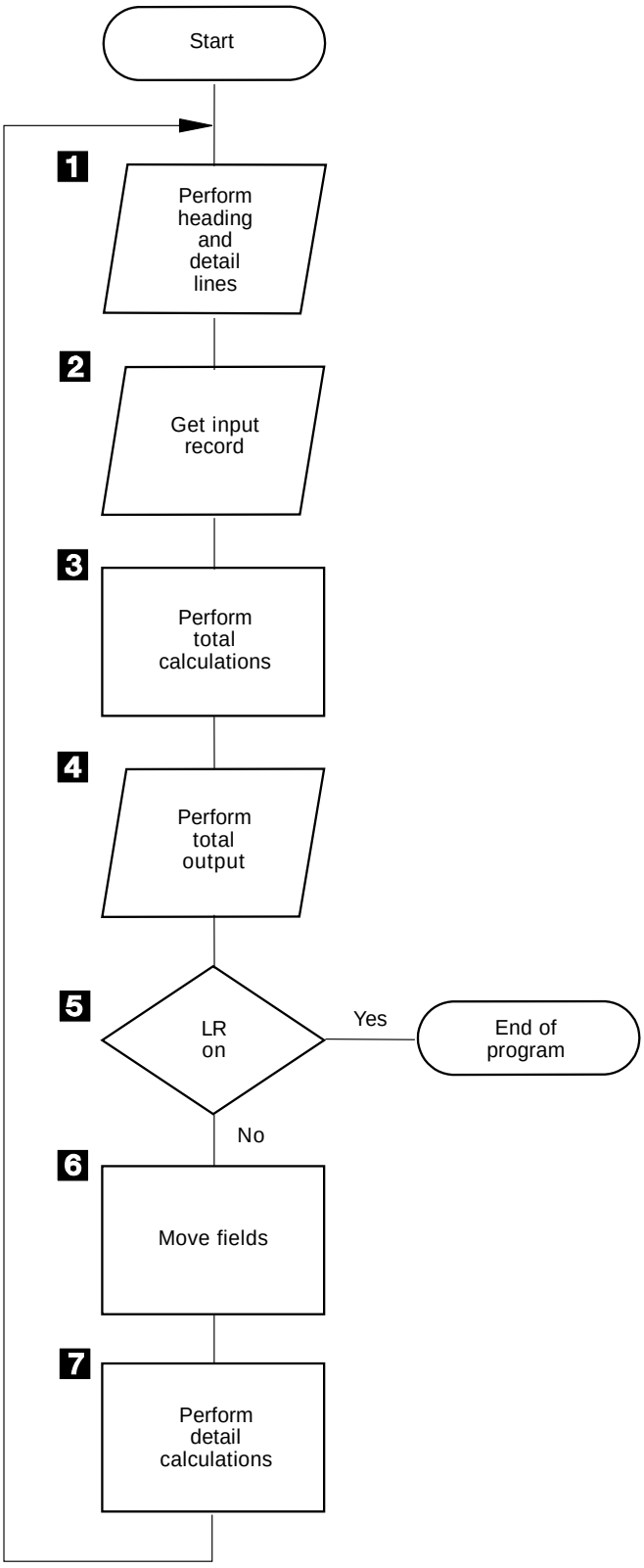


Figure 3. RPG/400 Program Logic Cycle

1. All heading and detail lines (H or D in position 15 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 15 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/400 Program Cycle

In “General RPG/400 Program Cycle” on page 11, the basic RPG/400 Logic Cycle was introduced. The following figures provide a detailed explanation of the RPG/400 Logic Cycle.

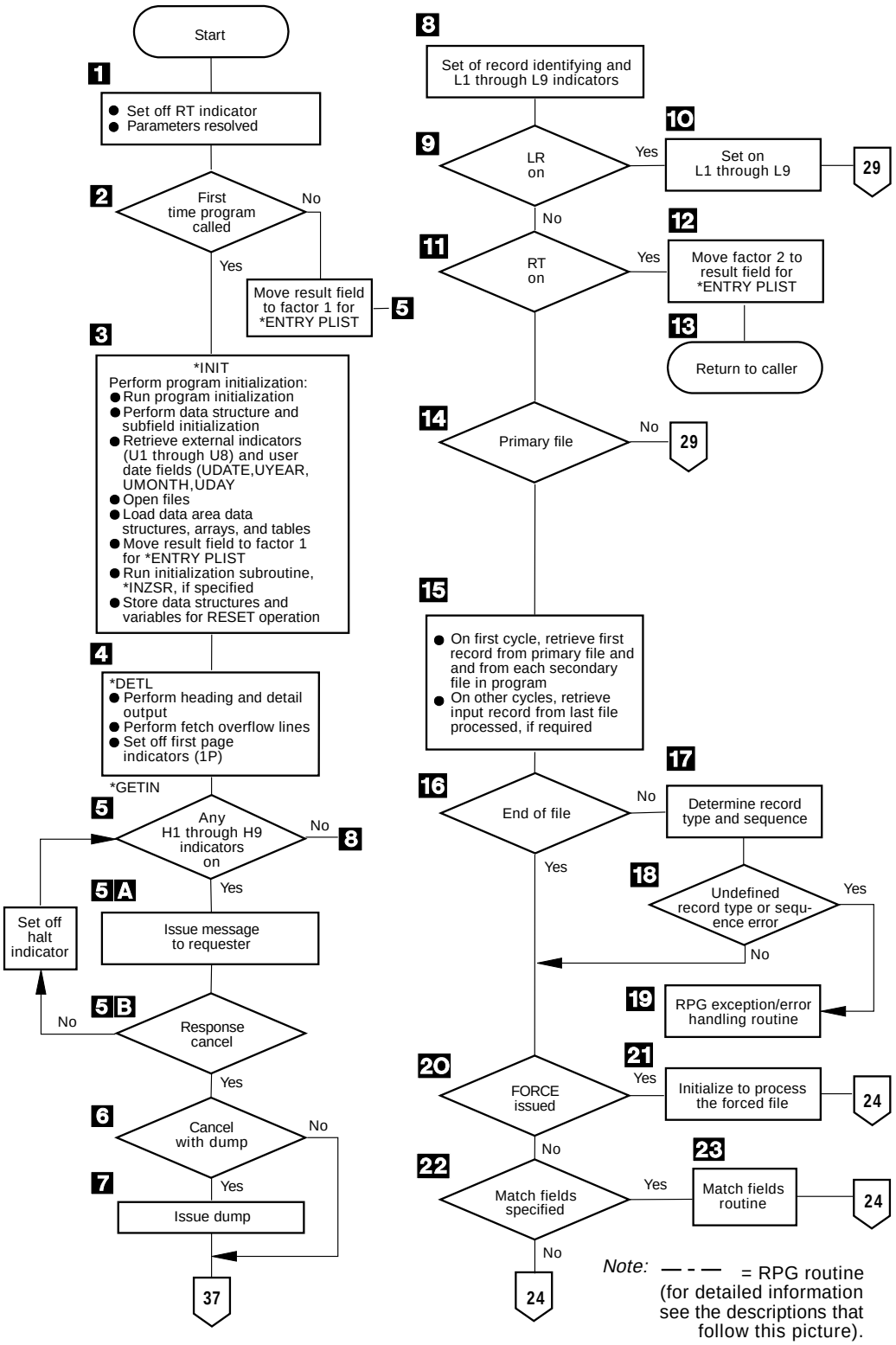


Figure 4 (Part 1 of 2). Detailed RPG/400 Object Program Cycle

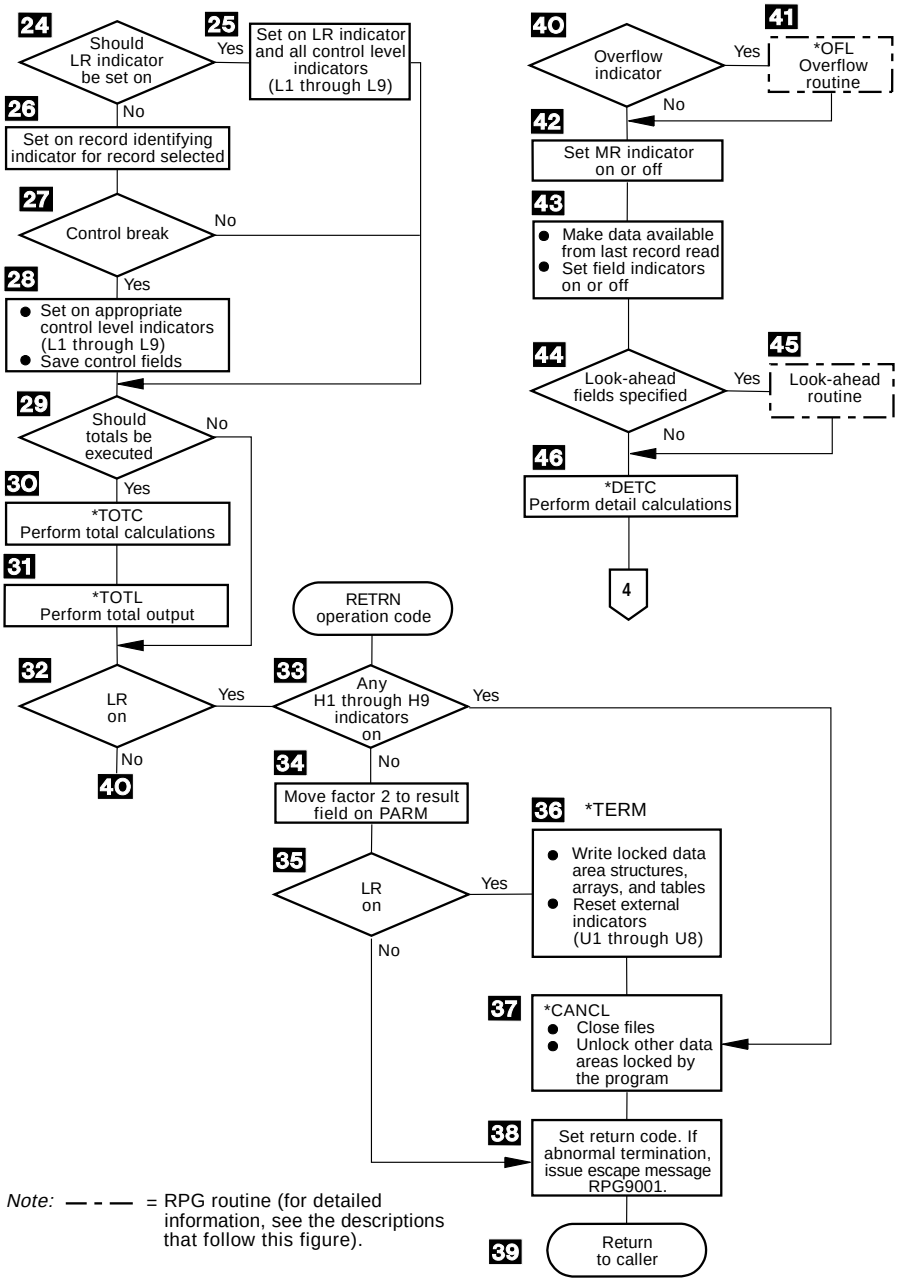


Figure 4 (Part 2 of 2). Detailed RPG/400 Object Program Cycle

Detailed RPG/400 Object Program Cycle

Figure 4 on page 14 shows the specific steps in the detailed flow of the RPG/400 program cycle. The item numbers in the following description refer to the numbers in the figure. Routines are flowcharted in Figure 7 on page 24 and in Figure 5 on page 20.

1 The RT indicator is set off. The RPG/400 language determines whether *ENTRY PLIST is specified. Parameters are resolved.

2 The first invocation of the is checked 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. The RPG/400 language performs data structure and subfield initialization; sets up the external indicators (U1 through U8) and user date fields (UDATE, UYEAR, UMONTH, UDAY); opens the files; loads all data area data structures, arrays, and tables; moves the result field to factor 1 in the PARMS statement in *ENTRY PLIST; runs the initialization subroutine *INZSR; and then stores the structures and variables for the RESET operation.

For more information see "Initialization" in chapter 9 of the *RPG/400 User's Guide*.

4 Heading and detail lines (identified by an H or D in position 15 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 a field name or a literal with 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 a field name or a literal with the value *GETIN.

a. If any halt indicators are on, a 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.

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 37.

7 The program issues a dump and branches to step 37 (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 It is determined whether the LR (last record) indicator is on. If it is on, the program continues with step 10; otherwise, the program branches to step 11.

10 The RPG/400 language sets the appropriate control level (L1 through L9) indicators on and branches to step 29.

11 It is determined whether the RT indicator is on. If it 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 *ENTRY PLIST.

13 If the RT indicator is on, the called program returns to the caller.

14 It is determined if the program contains a primary file. 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 RPG/400 language reads the first record from the primary file and from each secondary file in the program. In other program cycles, the RPG/400 language reads a record from the last file processed. If this file is processed by a record address (RA) file, the data in the record address file defines the record to be retrieved. If look-ahead fields are specified in the last record processed, the record may already be in storage. No read may be done at this time.

16 It is determined if end of file has occurred on the file just read. If it has not occurred, the program continues with step 17; otherwise, the program branches to step 20.

17 If a record has been read from the file, the record type and record sequence (positions 15 through 18 of the input specifications) are determined.

18 It is determined if the record type is defined in the program, or 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/400 exception/error handling routine receives control. (For detailed information on the RPG/400 exception/error handling routine, see “RPG/400 Exception/Error Handling Routine” on page 24.)

20 It is determined if 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 removing any match fields from the file just read. If the file forced is at end of file, normal primary/secondary multi-file logic selects the next record for processing and the program branches to step 24.

22 It is determined if match fields are specified. If they are, 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 21.)

24 The RPG/400 language sets on the LR (last record) indicator when all records are processed from the files that have an E specified in position 17 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 RPG/400 language sets on the LR (last record) indicator and all control level (L1 through L9) indicators, and processing continues with step 29.

26 The RPG/400 language sets on the record identifying indicator for the record selected for processing.

27 It is determined if 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. This causes all lower level control indicators to be set on. The program saves the contents of the control field.

29 It is determined if the total-time calculations and total-time output should be done. Totals are bypassed on the first cycle if control levels are not specified in the input specifications. After the first cycle, totals are processed on every cycle. If control levels are specified in the input specifications, totals are bypassed until after the first record containing control fields has been processed. Totals are always processed when the LR indicator is on.

30 The RPG/400 language processes all total calculations conditioned by a control level entry (positions 7 and 8 of the calculation specifications). This step is the return point in the program if factor 2 of an ENDSR operation contains a field name or a literal with 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 a field name or a literal with the value *TOTL.

32 If LR is on, the program continues with step 33; otherwise, the program branches to step 40.

33 The halt indicators (H1 through H9) are tested. If any halt indicators are on, the program branches to step 37 (abnormal ending). If the halt indicators are off, the program continues with step 34. If the RETRN operation code is used in calculations, the program branches to step 33 after processing of that operation.

34 The factor 2 fields are moved to the result fields on the PARMs.

35 If LR is on, the program continues with step 36; otherwise, the program branches to step 38.

36 The RPG/400 language writes all arrays or tables for which a file name is specified in the extension specifications (positions 19 through 26) and writes all locked data area data structures. External indicators (U1 through U8) are also reset. Output arrays and tables are translated, if necessary.

37 All open files are closed. If factor 2 of an ENDSR operation contains a field name or a literal with the value *CANCL, the return point in the program includes steps 37 through 39. The RPG/400 language also unlocks all data areas that have been locked (*NAMVAR DEFN statement) but not unlocked by the program.

38 The internal return code is set. Escape message RPG9001 is issued if ending is abnormal.

39 Control is returned to the caller.

Note: Steps 32 through 39 constitute the normal ending routine. For an abnormal ending, steps 34 through 36 are bypassed.

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

41 The overflow routine receives control. (For detailed information on the overflow routine, see "Overflow Routine" later in this chapter.) This step is the return point in the program if factor 2 of an ENDSR operation contains a field name or a literal with the value of *OFL.

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

43 Data from the last record read is made available for processing. All field indicators are set on, if specified.

44 It is determined whether look-ahead fields are specified. If so, the program continues with step 45; otherwise, the program branches to step 46.

45 The look-ahead routine receives control. (For detailed information on the look-ahead routine, see "Lookahead Routine" on page 22.)

46 Detail calculations are processed. This step is the return point in the program if factor 2 of an ENDSR operation contains a field name or a literal with the value *DETC. The program branches to step 4.

Initialization Subroutine

Refer to Figure 4 on page 14 to see a detailed explanation of the RPG/400 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 and if the program is still active (that is, not deactivated with a FREE operation), 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.

| If the initialization subroutine (*INZSR) does not complete before an exit from the
| program is made with LR off, the *INZSR will be rerun at the next invocation 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 may not be 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.

For more information see "Initialization" in Chapter 9 of the *RPG/400 User's Guide*.

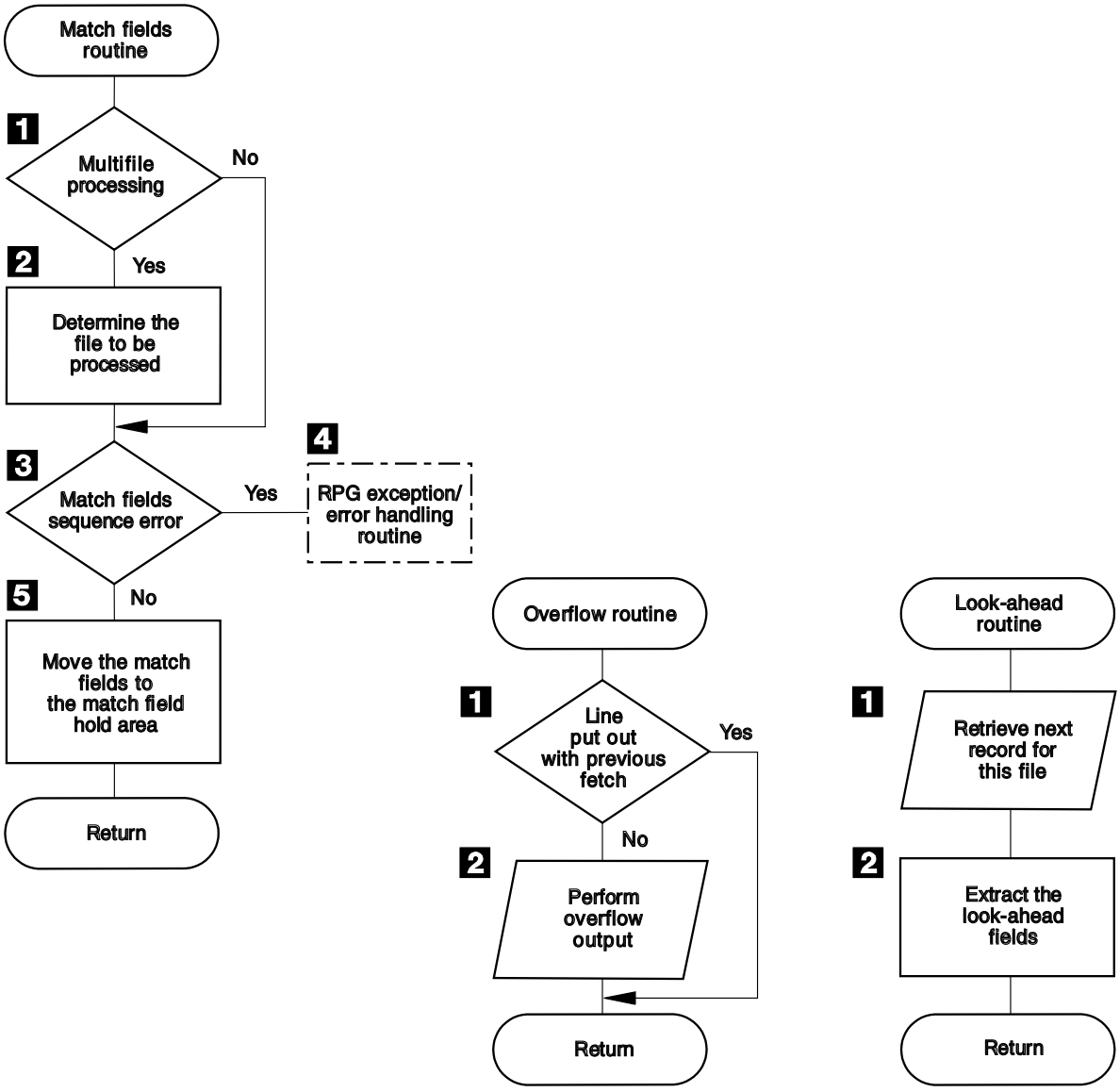


Figure 5. Detail Flow of RPG/400 Match Fields, Overflow, and Lookahead Routines

Match Fields Routine

Figure 5 on page 20 shows the specific steps in the RPG/400 match fields routine. The item numbers in the following descriptions refer to the numbers in the figure.

1. It is determined whether multifile processing is being used. 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/400 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/400 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 5 on page 20 shows the specific steps in the RPG/400 overflow routine. The item numbers in the following descriptions refer to the numbers in the figure.

1. The RPG/400 program determines whether the overflow lines were written previously using the fetch overflow logic (step 30 in Figure 4 on page 14). 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/400 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/400 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/400 program checks overflow on each line for which fetch overflow is specified.

Specify fetch overflow with an F in position 16 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 16, the RPG/400 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 16 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 23 through 31 of the same specification line. If this occurs, the overflow routine is not fetched.

Using the fetch overflow routine when printing a particular line causes overflow and 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 5 shows the specific steps in the RPG/400 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 15 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 RETRN operation code
- By allowing an exception/error to end the program if:
  - No exception/error subroutine is specified
  - No return point is specified on an exception/error subroutine
  - The user's response is to cancel the program.

The LR, RT, H1 through H9 indicators, the RETRN operation code, and the exception/error routine can be used in conjunction with each other.

Program Control of File Processing

Specify a full procedural file (F in position 16 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 16 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.

The file operation codes can be used for program control of input. These file operation codes are discussed in Chapter 11, “Operation Codes” on page 185.

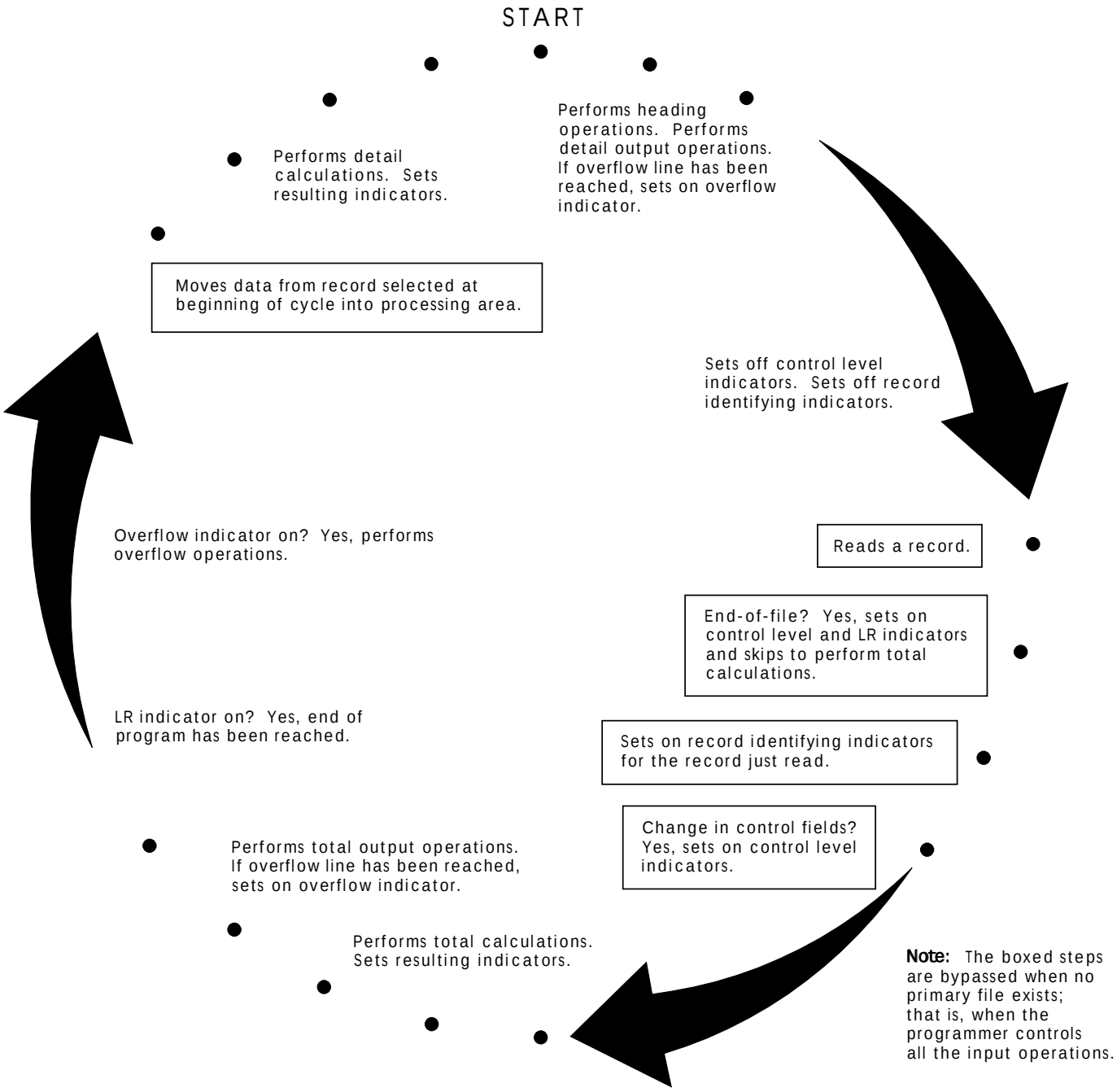


Figure 6. Programmer Control of Input Operation within the Program-Cycle

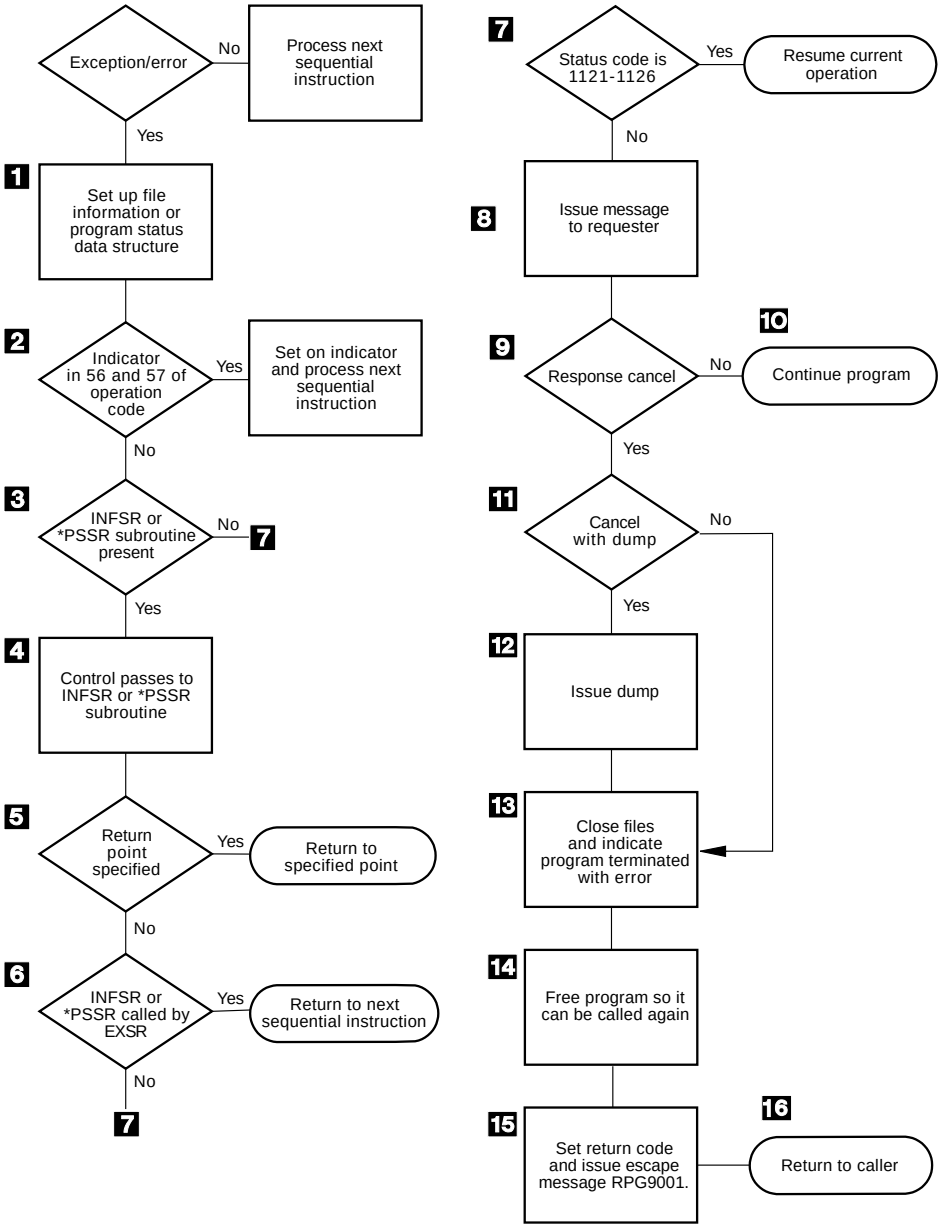


Figure 7. Detail Flow of RPG/400 Exception/Error Handling Routine

RPG/400 Exception/Error Handling Routine

Figure 7 shows the specific steps in the RPG/400 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 program 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 56 and 57, the indicator is set on, and control returns to the next sequential instruction in the calculations.
3. It is determined whether the appropriate exception/error subroutine (INFSR or *PSSR) is present in the program. If it is, the program goes to step 4; otherwise, the program branches to step 7.
4. Control passes to the exception/error subroutine (INFSR or *PSSR).

5. It is determined if a return point is specified in factor 2 of the ENDSR operation for the exception/error subroutine. If a return point is specified, the program goes to the specified return point. If a return point is not specified, the program goes to step 6. If a field name is specified in factor 2 of the ENDSR operation and the content is not one of the RPG/400-defined return points (such as *GETIN or *DETC), the program goes to step 6. No error is indicated, and the original error is handled as though the factor 2 entry were blank.
6. If the exception/error subroutine was called explicitly by the EXSR operation, the program returns to the next sequential instruction. If not, the program continues with step 7.
7. If the Status code is 1121-1126 (see “Status Codes” on page 39), control returns to the current instruction in the calculations.
8. A 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.
9. It is determined if the user's response is to cancel the program. If the response is to cancel, the program branches to step 11. If not, the program continues with step 10.
10. The program continues processing at *GETIN.
11. It is determined if the user's response is to cancel with a dump. If the response is to cancel with a dump, the program continues with step 12. If not, the program branches to step 13.
12. A dump is issued.
13. All files are closed and the return code is set to indicate that the program ended with an error.
14. The program is freed so that it can be called again.
15. Set return code and issue escape message RPG9001.
16. Return to caller.

File Exception/Errors

File Information Data Structure

A file information data structure (INFDS) can be defined for each file to make file exception/error information available to the program. The file information data structure must be unique for each file. A file information data structure contains predefined subfields 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/400 routine in which the exception/error occurred.

Note: The file information data structure is always provided and updated even if INFDS is not specified in the program. If INFDS is not specified, the information in the data structure is available for program debugging through the OS/400* debug facilities or the DUMP operation.

Overwriting the INFDS data structure may cause unexpected results in subsequent error handling and is not recommended.

You can use the file information data structure to access the information in the data management feedback area, which is available following each I/O operation. See "POST (Post)" on page 322 for information on how POST affects the updating of the feedback area.

To specify a file information data structure, make the following entries on a continuation line (or on the main file specification line) for the file description specifications:

Position	Entry
6	F
7-52	Blank (if the information is specified on a separate continuation line)
53	K (indicates a continuation statement)
54-59	INFDS (identifies this data structure as a file information data structure)
60-65	Name of the file information data structure.

Also make the following entries on an input specification line:

Position	Entry
6	I
7-12	Name of the file information data structure
13-18	Blank
19-20	DS
21-74	Blank.

For each subfield of the information data structure, make the following entries on an input specification line:

Position	Entry
6	I
7-43	Blank
44-51	A special keyword (listed below) or a From and To position in the file information data structure.
52	Blank
53-58	Name of the subfield of the information data structure
59-74	Blank.

The location of the subfields in the file information data structure is defined by special keywords.

Specify the special keywords left-adjusted, in positions 44 through 51. The keywords are not labels and cannot be used to access the subfields. Short entries are padded on the right with blanks. The keywords and their descriptions are as follows:

Keyword	Description
*FILE	Eight-position character field that identifies the name of the file (as specified in positions 7 through 14 of the file description specifications).
*INP	A two-digit numeric field containing 0. The national language input capability of the device is for single characters.
*MODE	A two-digit numeric field containing 0. The preferred national language mode of the device is for single characters.
*OUT	A two-digit numeric field containing 0. The national language output capability of the device is for single characters.
*OPCODE	Six-position character field that contains the name of the last operation processed on the file. 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. 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.
*SIZE	A four-digit numeric field containing the product of the number of rows and the number of columns on the device screen.
*STATUS	Five-digit numeric field, with zero decimal positions, that contains the status code. For a description of these codes, see “Status Codes” on page 39.
*RECORD	*Program described file:* Eight-position character field in which the record identifying indicator is placed left-adjusted; the remaining six positions are filled with blanks. *Externally described file:* Eight-position character field which contains the name of the record being processed when the exception/error occurred.
*ROUTINE	Eight-position character field that contains the name of the routine in which the exception/error occurred. This subfield is updated at the beginning of an RPG/400 routine or after a program call only when the location *STATUS is updated with a value of nonzero. 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 pgmname Name of program called (first 8 characters). subrname Name of subroutine EXCPT name Name of EXCPT

Note: The fields defined using keywords *SIZE, *INP, *OUT, and *MODE are only valid after a POST operation to a specific device.

A file information data structure (INFDS) can be defined for each file in an RPG/400 program:

Even if a file information data structure is not specified, the information in it is available for program debugging through the OS/400 debug facilities or the RPG/400 DUMP operation.

Table 1 to Table 5 provide the layout of the subfields of the feedback information available in the file information data structure. You can use the predefined From and To positions to access the feedback information.

The input/output feedback information section (positions 241 through 366) and the device-dependent feedback information section (positions 367 on, see *Data Management Guide*). of the file information data structure are not updated for each operation to files in which the records are blocked and unblocked. The feedback information is updated only when a block of records is transferred between RPG/400 system and the OS/400 system. For a potential run-time performance improvement of input and output operations, the RPG/400 language unblocks input records and blocks output records in SEQ or DISK files if:

- The file is an output-only file (O is specified in position 15 of the file description specifications) and contains only one record format if the file is externally described.
- The file is a combined table file. (C is specified in position 15, and T in position 16 of the file description specifications.)
- The file is an input-only file. (I is specified in position 15 of the file description specifications). It contains only one record format if the file is externally described, and uses only the OPEN, CLOSE, FEOD, and READ operation codes.

Even if all of the above conditions are met, certain OS/400 system restrictions may prevent blocking and unblocking. In these cases, performance is not improved and the input/output feedback area is updated for each input/output operation.

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/400 language does not block or unblock the records in the file.

Product-Sensitive Programming Interface

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

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
1	8	Character	8	File name (same as subfield location *FILE).
9	9	Character	1	Open indication (1 = open).
10	10	Character	1	End of file (1 = end of file)
11	15	Zoned decimal	5 (zero decimal positions)	Status code (same as subfield location *STATUS).
16	21	Character	6	Operation code (same as subfield location *OPCODE).
22	29	Character	8	Name of the RPG/400 routine in which the exception/error occurred (same as subfield location *ROUTINE).
30	37	Character	8	RPG/400 source statement sequence number.
38	42	Zoned decimal	5 (zero decimal positions)	User-specified reason for error on SPECIAL file.
38	45	Character	8	For a program described file the record identifying indicator is placed left-adjusted in the field (same as the subfield location *RECORD). For an externally described file, the name of the record being processed when the exception/error occurred.
46	52	Character	7	Machine or system message number.
53	56	Character	4	MI/ODT (machine instruction/object definition template) number.
57	66	Character	10	Unused.

Table 2 (Page 1 of 2). Contents of the File Feedback Information Available in the File-Information Data Structure (INFDS) Valid after a POST Operation to a Specific De

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
67	70	Zoned decimal	4 (zero decimal positions)	Screen size (same as subfield location *SIZE).

Table 2 (Page 2 of 2). Contents of the File Feedback Information Available in the File-Information Data Structure (INFDS) Valid after a POST Operation to a Specific De

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
71	72	Zoned decimal	2 (zero decimal positions)	The national language input capability of the device is for single characters. The value is 0 (same as subfield location *INP).
73	74	Zoned decimal	2 (zero decimal positions)	The national language output capability of the device is for single characters. The value is 0 (same as subfield location *OUT).
75	76	Zoned decimal	2 (zero decimal positions)	The preferred national language mode of the device is for single characters. The value is 0 (same as subfield location *MODE).

Note: The remaining feedback information (starting at position 81) is copied from the open feedback and I/O feedback areas, which are described in the *Data Management Guide*. The description in the information column indicates system usage of the fields and may not apply to the RPG/400 user. For example, information in positions 241 and 242 is used by the system to determine the start of the file-dependent feedback area and is not applicable to the RPG/400 user.

The length of the INFDS 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.

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

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
81	82	Character	2	Open data path (ODP) type: DS Device file DB Database member SP Spooled file
83	92	Character	10	Name of the file. For a nonspooled file, this is the name of the file actually being opened. For a spooled file, this is the name of the device file or the inline data file being opened.
93	102	Character	10	Name of the library containing the file. For a spooled input file, this is *N.

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

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
103	112	Character	10	Name of the spooled file. This entry is set only if the ODP type is SP. It is the name of a database file containing the spooled input or output records.
113	122	Character	10	Name of the library where the spooled file is located.
123	124	Binary	2	Spooled file number (supplied only for spooled output).
125	126	Binary	2	Record length (number of bytes transferred at a time).
127	128	Binary	2	Reserved.
129	138	Character	10	Member name: • If ODP type is DB, this entry is the member name in the file named in positions 83 through 92. • If ODP type is SP, this entry is the member name in the file named in positions 103 through 112.
139	142	Binary	4	Not used.
143	146	Binary	4	Not used.
147	148	Binary	2	File type (supplied only if the ODP type is DS or SP). See the *Data Management Guide* for file type codes.
149	151	Character	3	Reserved.
152	153	Binary	2	Number of rows on a display screen or number of lines on a printed page (supplied only for a display device or a printer).
154	155	Binary	2	Number of columns on a display screen or number of characters per printed line (supplied only for display device or printer).
156	159	Binary	4	Number of records in the member at open time. This entry is supplied only if the ODP type is DB or SP and the file is being opened for input.
160	161	Character	2	Access type (supplied only if ODP type is DB): KU Keyed, unique KF Keyed, first-in-first-out (FIFO) with duplicate keys KL Keyed, last-in-first-out (LIFO) with duplicate keys AR Arrival sequence
162	162	Character	1	Duplicate key indication. This entry is set only if the access path is KU, KF, or KL:. D Duplicate keys allowed if the access path is KF or KL. U Duplicate keys are not allowed; all keys are unique and the access path is KU.
163	163	Character	1	Source file indication. This entry contains Y if this file is being opened as a source file.
164	173	Character	10	User file control block (UFCB) parameters. This entry indicates which UFCB parameters are in effect.

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

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
174	183	Character	10	User file control block (UFCB) overrides. This entry indicates which override parameters are in effect.
184	185	Binary	2	Offset to volume label fields of open feedback area. This entry is supplied only for tape or diskette; otherwise, the field contains zero.
186	187	Binary	2	Maximum number of records that can be sent or received in a block when using blocked record I/O.
188	189	Binary	2	Overflow line number (supplied only for a printer file).
190	191	Binary	2	Blocked record I/O record increment. This is the number of bytes to add to the address of a record to get the address of the next record in a block.
192	196		5	Unused.
197	206	Character	10	Name of the requester program device.
207	208	Binary	2	File open count. If the file is opened nonshareable, this field contains a 1. If the file is opened shareable, this field contains the number of UFCBs currently attached to this file.
209	210	Binary	2	Reserved.
211	212	Binary	2	Number of based-on physical members opened. For logical members, this is the number of physical members over which the logical member was opened. For physical members, this field is always set to 1.
213	213	Character	1	Miscellaneous flags. See the *Data Management Guide* for details.
214	215	Character	2	Open Identifier. Value is unique for a full open of a file (SHARE(*NO) or the first open of a file with SHARE(*YES)). Allows matching between this file and an entry on the associated Data Queue.
216	217	Binary	2	Maximum Record Length. This value includes the data, plus source sequence numbers, option indicators, response indicators, and P-data lengths if applicable. If this field is 0, then use the field from location 125 to 126.

See Table 5 on page 34 for the values after a POST operation with a device specified in factor 1.

Table 4. Contents of the Input/Output Feedback Information Available in the File-Information Data Structure (INFDS)

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
241	242	Binary	2	Offset to file-dependent feedback information. See the *Data Management Guide* for the layout of feedback information for specific files.
243	246	Binary	4	Write operation count. This entry is updated only when a Write operation is completed successfully. For information specific to ICF, see the corresponding entry in the *ICF Programmer's Guide*.
247	250	Binary	4	Read operation count. This entry is updated only when a Read operation is completed successfully. For database files, this entry is not updated for a position-only request. For information specific to ICF, see the corresponding entry in the *ICF Programmer's Guide*.
251	254	Binary	4	Write/Read operation count. This entry is updated only when a Write/Read operation is completed successfully.
255	258	Binary	4	Other I/O operation count. Number of successful operations other than Write, Read, or Write/Read. Updated only when operation successfully completes. Count includes update, delete, force-end-of-data, force-end-of-volume, ACQ, REL, and release record lock requests.
259	259		1	Unused.
260	260	Character	1	Current operation. This entry represents the last operation requested. See *Data Management Guide*, Common I/O Feedback Area table for operation codes.
261	270	Character	10	Name of the record format just processed, which is either: • Specified on the I/O request • Determined by system processing. For a display device, the default name is either the name of the only record format in the file or the previous record format name for the record put to the display that contains input data. For ICF files, see the description of the FMTSLT parameter on the CL commands ADDICFDEVE and OVRICFDEVE in the *ICF Programmer's Guide*.
271	272	Character	2	Device class. In the *Data Management Guide*, see the Common I/O Feedback Area table for device class codes.
273	282	Character	10	Program device name. This entry is the name of the program device for which the operation just completed.
283	286	Binary	4	Length of the record processed by the last I/O operation (supplied for display device file, database file, tape file, and ICF file). On ICF write operations, this entry is the record length of the data. On ICF read operations, this entry is the length of the record for the last input operation.

File Dependent Feedback Information

Complete file-dependent feedback information is available in the *Data Management Guide*. Refer to the section "I/O Feedback Area" and the tables on the file-dependent area in that section.

To calculate the From and To positions (positions 44 through 47 and 48 through 51 of the input specifications) that specify the subfields of the file-information data structure (INFDS) for the file-dependent area, use the Offset, Data Type, and Length given in the *Data Management Guide* and do the following calculations:

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

For example, for the relative record number of a subfile record, the *Data Management Guide* gives:

```
Offset = 9
Data Type is Binary
Length = 2

Therefore,

From = 367 + 9 = 376,
To = 376 - 1 + 2 = 377.
```

Contents of File-Information Data Structure after POST

After a POST operation with a program device specified in factor 1, the following information overlays positions 241 on. For more information on these positions see the section on getting attributes for display devices in the *Data Management Guide*.

Note

For information on the System/38 environment POST operation, see the *System/38 RPG III Reference Manual and Programmer's Guide*, SC21-7725.

Table 5 (Page 1 of 4). Contents of the Input/Output Feedback Information Available in the File-Information Data Structure (INFDS) after a POST Operation

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
241	250	Character	10	Program device name.
251	260	Character	10	Device description name.
261	270	Character	10	User ID
271	271	Character	1	Device class: • 'D' is display • 'I' is ICF • 'U' is unknown

Table 5 (Page 2 of 4). Contents of the Input/Output Feedback Information Available in the File-Information Data Structure (INFDS) after a POST Operation

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
272	277	Character	6	Device type: • '3179' 3179 display • '317902' 3179 model 2 display • '3180' 3180 display • '3196A' 3196 model A1/A2 display • '3196B' 3196 model B1/B2 display • '3197C1' 3197 model C1 display • '3197C2' 3197 model C2 display • '3197D1' 3197 model D1 display • '3197D2' 3197 model D2 display • '3197W1' 3197 model W1 display • '3197W2' 3197 model W2 display • '3270' 3270 display • '3476EA' 3476 model EA display • '3477FA' 3477 model FA display • '3477FC' 3477 model FC display • '3477FD' 3477 model FD display • '3477FG' 3477 model FG display • '3477FH' 3477 model FH display • '3477FW' 3477 model FW display • '525111' 5251 display • '5291' 5291 display • '5292' 5292 display • '529202' 5292 model 2 display • '5555B1' 5555 model B01 display • '5555E1' 5555 model E01 display • 'APPC' APPC • 'ASYNC' Asynchronous • 'BSC' Bisynchronous communications • 'BSCEL' BSCEL • 'DHCF77' 3277 DHCF display • 'DHCF78' 3278 DHCF display • 'DHCF79' 3279 DHCF display • 'FINANC' Finance • 'INTRA' Intrasystem communications • 'LU1' LU1 communications • 'NVT' Network virtual terminal • 'RETAIL' Retail • 'SNUF' SNUF
278	278	Character	1	*REQUESTER device. This flag indicates if this entry is defining a *REQUESTER device. • 'Y' is requesting program device. • 'N' is not a requesting program device.
279	279	Character	1	Acquire status. Set even if a device is implicitly acquired at open time. • 'Y' device is acquired. • 'N' device is not acquired.

Table 5 (Page 3 of 4). Contents of the Input/Output Feedback Information Available in the File-Information Data Structure (INFDS) after a POST Operation

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
280	280	Character	1	Invite status. • 'Y' device is invited. • 'N' device is not invited.
281	281	Character	1	Data available • 'Y' invited data is available. • 'N' no data is available.
282	283	Binary	2	Number of rows on display.
284	285	Binary	2	Number of columns on display.
286	286	Character	1	Display allow blink capability. • 'Y' blink capable. • 'N' not blink capable.
287	287	Character	1	Online/Offline status. • 'O' display is online. • 'F' display is offline.
288	288	Character	1	Display location. • 'L' local display. • 'R' remote display.
289	289	Character	1	Display type. • 'A' alphanumeric or Katakana. • 'I' ideographic.
290	290	Character	1	Keyboard type of display. • 'A' alphanumeric or Katakana keyboard. • 'I' ideographic keyboard.
291	291	Character	2	Session status (All communication types). • 'N' transaction is not started. • 'Y' transaction is started.
292	292	Character	1	Synchronization level (APPC communications only). • '0' synchronization level 0 (SYNCLVL(*NONE)). • '1' synchronization level 1 (SYNCLVL(*CONFIRM)).
293	293	Character	1	Conversation being used (APPC communications only). • 'M' mapped conversation. • 'B' basic conversation.
294	301	Character	8	Remote Location (All communications types).
302	309	Character	8	Local LU name (APPC communications only).
310	317	Character	8	Local network ID (APPC communications only).
318	325	Character	8	Remote LU name (APPC communications only).
326	333	Character	8	Remote network ID (APPC communications only).
334	341	Character	8	Mode (APPC communications type only).

Table 5 (Page 4 of 4). Contents of the Input/Output Feedback Information Available in the File-Information Data Structure (INFDS) after a POST Operation

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
386	*	Character	*	More fields More information about feedback area layouts is available in the *Data Management Guide*.

End of Product-Sensitive Programming Interface

File Exception/Error Subroutine (INFSR)

To identify the user-written RPG/400 subroutine that may receive control following file exception/errors, make the following entries on a continuation line (or on the main file line) for the file description specifications:

Position	Entry
6	F
7-52	Blank (if the information is specified on a separate continuation line)
53	K (indicates a continuation statement)
54-59	INFSR
60-65	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 56 and 57. The file exception/error subroutine can also be run by the EXSR operation code. Any of the RPG/400 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 in positions 60 through 65 of the file description specifications continuation line).
The ENDSR operation must be the last specification for the file exception/error subroutine and should be specified as follows:

Position	Entry
6	C
7-17	Blank
18-27	Can contain a label that is used in a GOTO specification within the subroutine.
28-32	ENDSR
33-42	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/400 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.

43-59 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/400 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/400 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/400 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 RETRN operation as follows:

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C* If INFSR is already handling the error, exit.
C           ERRRTN    BEGSR
C           SW        IFEQ '1'
C                     SETON                     H1
C                     RETRN
C* Otherwise, flag the error handler.
C                     ELSE
C                     MOVE '1'       SW
C                     :
C                     :               Error handling routine.
C                     :
C                     END
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 56 and 57, 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 6. 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 7 (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).
01031	W,D,SQ	n/a	Match field out of sequence.
01041	n/a	n/a	Array/table load sequence error.
01051	n/a	n/a	Excess entries in array/table file.
01052	n/a	n/a	Clearing of table prior to dump of data failed.
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.
01231	SP	n/a	Error on SPECIAL file.
01235	P	n/a	Error in PRTCTL space or skip entries.
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.
01281	W	n/a	Operation to unacquired device.
01282	W	0309	Job ending with controlled option.
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.

Table 7 (Page 2 of 2). Exception/Error Codes

Code	Device[1]	RC[2]	Condition
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 7 on page 24 for special handling.

Note

For System/38 Environment *STATUS, see the *System/38 RPG III Reference Manual and Programmer's Guide*, SC21-7725.

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 Guide* 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, or FREE operation code, 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 56 and 57 of the calculation specifications for an operation code. This indicator is set on if an exception/error occurs during the processing of the specified file 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.

- 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 18 of the data structure statement on the input specifications.
- If the indicator or the program exception/error subroutine is not present, program exception/errors are handled by the RPG/400 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/400 program.

A data structure is defined as a program status data structure by an S in position 18 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 in positions 53 through 58.

The keywords must be specified, left-adjusted in positions 44 through 51. The keywords are not labels and cannot be used to access the subfields. Short entries are padded on the right with blanks. The keywords and their descriptions are as follows:

Keyword	Description
*STATUS	Five-digit numeric field with zero decimal positions, that contains the status code.
*ROUTINE	Eight-position character field that contains the name of the RPG/400 routine in which the exception/error occurred. This subfield is updated at the beginning of an RPG/400 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 pgmname Name of program called (first 8 characters). **Note:** *ROUTINE is not valid unless you use the normal RPG/400 cycle. Logic that takes the program out of the normal RPG/400 cycle may cause *ROUTINE to reflect an incorrect value.
*PARMS	Three-digit numeric field that contains the number of parameters passed to this program from the calling program.
*PROGRAM	Ten-position character field that contains the name of the program in which this program status data structure is specified.

Information from the program status data structure is also provided in a formatted dump.

Table 8 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.

Product-Sensitive Programming Interface

Table 8 (Page 1 of 2). Contents of the Program Status Data Structure

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
1	10	Character	10	Program name (same as subfield location *PROGRAM).
11	15	Zoned decimal	5 (zero decimal positions)	Status code (same as subfield location *STATUS).
16	20	Zoned decimal	5 (zero decimal positions)	Previous status code.
21	28	Character	8	RPG/400 source statement sequence number.
29	36	Character	8	Name of the RPG/400 routine in which the exception or error occurred (same as subfield location *ROUTINE).
37	39	Zoned decimal	3 (zero decimal positions)	Number of parameters passed to this program (same as subfield location *PARMS).
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	MI/ODT (machine instruction / object definition template) number.
51	80	Character	30	Work area for messages. This area is only meant for internal use by the RPG/400 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.
171	174	Character	4	Identification of the exception that caused RPG9001 exception to be signaled (the called program failed).
175	198		24	Unused.
199	200	Zoned decimal	2	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).

Table 8 (Page 2 of 2). Contents of the Program Status Data Structure

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
209	243	Character	35	Status information on the last file used. This information includes the status code, the RPG/400 routine name, the statement number, and record name. It is updated only when an error occurs.
244	253	Character	10	Job name.
254	263	Character	10	User name from the user profile.
264	269	Zoned decimal	6 (zero decimal positions)	Job number.
270	275	Zoned decimal	6 (zero decimal positions)	Date (in UDATE format) the program started running in the system (UDATE is derived from this date). See on page 382 for a description of UDATE.
276	281	Zoned decimal	6 (zero decimal positions)	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.
314	323	Character	10	Source library name.
324	333	Character	10	Source file member name.
334	429		96	Unused.

End of Product-Sensitive Programming Interface

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 56 and 57, 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
00121	Array index not valid
00122	OCUR outside of range
00123	Reset attempted during initialization step of program
00202	Called program failed; halt indicator (H1 through H9) not on
00211	Program specified on CALL or FREE not found
00221	Called program tried to use a parameter that was not passed to it
00231	Called program returned with halt indicator on
00232	Halt indicator on in this program
00233	Halt indicator on when RETRN operation run
00299	RPG/400 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 UNLCK operation
00431	Data area previously locked by another program
00432	Data area locked by program in the same process
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/400 run-time subroutines.
09998	Internal failure in RPG/400 compiler or in run-time subroutines
09999	Program exception in system routine.

Program Exception/Error Subroutine

To identify the user-written RPG/400 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 56 and 57 for the operation code, 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 in positions 60 through 65 of the file description specifications continuation line for the file exception/error subroutine (INFSR in positions 54 through 59), and receives control if a file exception/error occurs.

Any of the RPG/400 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 37.

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/400 language resets the field or array element specified in factor 2 to blanks. Thus, if you do not place a value in this field during the running of the subroutine, the RPG/400 default error handler receives control following the running of the subroutine, unless the *PSSR subroutine was called by the EXSR operation. This allows you to specify the return point within the subroutine that is best suited for the running/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.
- 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.

Chapter 3. RPG/400 Indicators

An indicator is a two-character entry on a specification that either is set on (1) or off (0) as the result of an operation or is used to condition (or control) the processing of an operation.

Indicators are defined either by an entry on the specification or by the RPG/400 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/400 program sets and resets certain indicators at specific times during the program cycle. In addition, the state of most indicators can be changed by the SETON and SETOF operation codes. 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 SETOF operation code.

This chapter is divided into the following topics:

- Indicators defined on the RPG/400 specifications
- Indicators not defined on the RPG/400 specifications
- Using indicators
- Indicators referred to as data.

Indicators Defined on RPG/400 Specifications

You can define an indicator on the RPG/400 specifications if it is specified as one of the following:

- Overflow indicator (positions 33 and 34 of the file description specifications).
- Record identifying indicator (positions 19 and 20 of the input specifications).
- Control level indicator (positions 59 and 60 of the input specifications).
- Field indicator (positions 65 through 70 of the input specifications).
- Resulting indicator (positions 54 through 59 of the calculation specifications).
- *IN array, *IN,xx array element or *INxx field (See “Indicators Referred to As Data” on page 75 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 an entry in positions 33 and 34 of the file description specifications. It is set on when the last line on a page has been printed or passed. Valid indicators are OA through OG, OV, and 01 through 99. 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 the PRINTER file section of the *RPG/400 User’s Guide*.

Record Identifying Indicators

A record identifying indicator is defined by an entry in positions 19 and 20 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 to cause certain total steps to be processed. 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/400 cycle. (See Figure 6 on page 23.) 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/400 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 19 and 20, 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 21 through 41) 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 14.

For an example of record identifying indicators, see Figure 8.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+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 CX2 CY3 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
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+L1M1FrPlMnZr...*
I*
I* For an externally described file, record identifying indicator 10
I* is 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 8. Examples of Record Identifying Indicators

Control Level Indicators (L1-L9)

A control level indicator is defined by an entry in positions 59 and 60 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.

You can specify a control break 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 9 on page 51).

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 53 through 58 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 9 on page 51).
- 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 10 on page 52). 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 10 on page 52, 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 11 on page 53 shows an example of how to avoid unwanted control breaks.

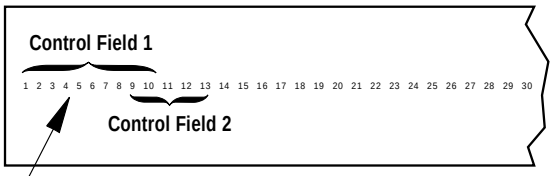
Figure 11 on page 53 shows an example of how to avoid unwanted control breaks.

```
*...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 9 (Part 1 of 2). Control Level Indicators (Two Record Types)

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+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                                                  EMPLNOL1
I                                                  DIVSONL3
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                                                  EMPLNOL1
I                                                  DEPT  L2
I                                                  DIVSONL3
```

Figure 9 (Part 2 of 2). Control Level Indicators (Two Record Types)



A total of 15 positions has been assigned to these control levels.

Figure 10. Overlapping Control Fields

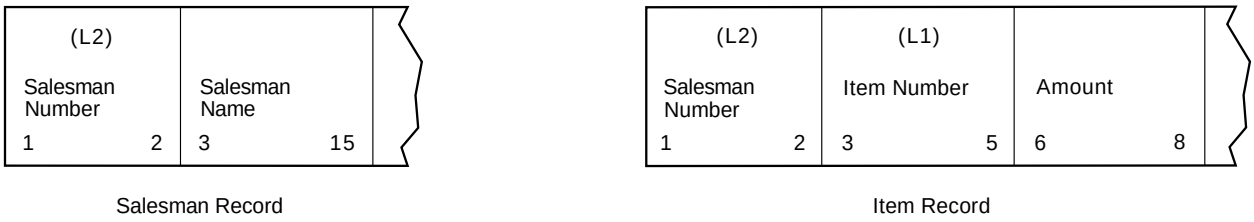


Figure 11 (Part 1 of 3). How to Avoid Unwanted Control Breaks

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+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
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++++*
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               SETOF                     11
C   02      AMT      ADD  L1TOT     L1TOT   50
CL1         L1TOT    ADD  L2TOT     L2TOT   50
CL2         L2TOT    ADD  LRTOT     LRTOT   50
```

Figure 11 (Part 2 of 3). How to Avoid Unwanted Control Breaks

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
OName++++DFBASbSaN01N02N03Field+YBEnd+PConstant/editword+++++++++...*
OPRINTER D 11     01
O                         L2FLD      5
O                         NAME      25
O        D  1     02
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*
O        T  1     L1N11
O                         L1TOT ZB  25
O                                   27 '*'
O        T  1     L2
O                         L2TOT ZB  25
O                                   28 '**'
O        T  1     LR
O                         LRTOT ZB  25
```

```
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 11 (Part 3 of 3). 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/400 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.

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 12). 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...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+L1M1FrPlMnZr...*
IMASTER      01
I                                          28  31 CUSNO L4
I                                          15  20 ACCTNOL4
I                                          50  52 REGNO L4
```

Figure 12. 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.

Figure 13 shows examples of the preceding rules.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+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 13. 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 65 and 66, 67 and 68, or 69 and 70 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 65 through 68 are valid for numeric fields only; positions 69 and 70 are valid for numeric or character fields. An indicator specified in positions 65 and 66 is set on when the numeric input field is greater than zero; an indicator specified in positions 67 and 68 is set on when the numeric input field is less than zero; and an indicator specified in positions 69 and 70 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 54 through 59 of the calculation specifications. The purpose of the resulting indicators depends on the operation code specified in positions 28 through 32. (See the individual operation code in Chapter 11, “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 SETOF)
RT
```

You can specify resulting indicators in three places (positions 54-55, 56-57, and 58-59) 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 “LOKUP (Look Up)” on page 286, “SETOF (Set Off)” on page 351, and “SETON (Set On)” on page 352. 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 14 on page 59).

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 2, “RPG/400 Program Cycle and Error Handling” on page 11).
- 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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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  30 1020
C*
C  N20      PAYRAT    MULT 40        PAY     62H
C   10      OVERTM    MULT OVRRAT    OVRPAY  62H
C   10      OVRPAY    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   20      PAYRAT    MULT HRSWKD    PAY
C*
```

Figure 14. Resulting Indicators Used to Condition Operations

Indicators Not Defined on the RPG/400 Specifications

Not all indicators that can be used as conditioning indicators in an RPG/400 program are defined on the specification forms. External indicators (U1 through U8) are defined by a CL command or by a previous RPG/400 program. Internal indicators (1P, LR, MR, and RT) are defined by the RPG/400 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/400 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/400 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/400 program has no effect on the copy of the external indicators used by the RPG/400 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/400 program. The current status of the external indicators can then be used by other programs.

Note: When using RETRN 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/400 program when the program starts running and is set off by the RPG/400 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 by 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 2, "RPG/400 Program Cycle and Error Handling" on page 11.) 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 15, "General File Considerations."

Return Indicator (RT)

You can use the return indicator (RT) to indicate to the internal RPG/400 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 *RPG/400 User'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/400 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 in positions 71 and 72 of the file description specifications. Only the external indicators U1 through U8 are valid for file conditioning. (An entry of UC can also be made in positions 71 and 72; however, UC is not an indicator.)

If the external indicator specified in positions 71 and 72 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, REDPE, 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 63 and 64 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 63 and 64 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 63 and 64 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 63 and 64 (see Figure 15 on page 65).

An indicator (01 through 99) that is not a record identifying indicator can also be used in positions 63 and 64 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 59 and 60 of the input specifications, and match fields, which are specified by a match value (M1 through M9) in positions 61 and 62 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 63 and 64.

You can use positions 63 and 64 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 63 and 64. Data from the field named in positions 53 through 58 is accepted only when the field record relation indicator is on.

External indicators are primarily used when file conditioning is specified in positions 71 and 72 of the file description specifications. However, they can be used even though file conditioning is not specified.

A halt indicator (H1 through H9) in positions 63 and 64 relates a field to a record that is in an OR relationship and also has a halt indicator specified in positions 19 and 20.

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

- Control level (positions 59 and 60) and matching fields (positions 61 and 62) with the same field record relation indicator must be grouped together.
- Fields used for control level (positions 59 and 60) and matching field entries (positions 61 and 62) without a field record relation indicator must appear before those used with a field record relation indicator.
- Control level (positions 59 and 60) and matching fields (positions 61 and 62) with a field record relation indicator (positions 63 and 64) take precedence, when the indicator is on, over control level and matching fields of the same level without an indicator.
- Field record relations (positions 63 and 64) for matching and control level fields (positions 59 through 62) 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 59 and 60) and matching field (positions 61 and 62) specifications can be interspersed with groups of field record relation entries (positions 63 and 64).
- 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 61 and 62) are specified with field record relation indicators (positions 63 and 64), 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...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+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 15. 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 *RPG/400 User'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 SETOF 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/400 cycle (see Chapter 2, "RPG/400 Program Cycle and Error Handling" on page 11). 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 RETRN 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/400 cycle in Chapter 2, "RPG/400 Program Cycle and Error Handling" on page 11.

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 17.

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 9-17

You can use positions 9 through 17 of the calculation specifications to specify indicators that control the conditions under which an operation is processed. The valid entries for positions 9 through 17 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 17 must be previously defined as one of the following types of indicators:

- Overflow indicators (file description specifications, positions 33 and 34)
- Record identifying indicators (input specifications, positions 19 and 20)
- Control level indicators (input specifications, positions 59 and 60)
- Field indicators (input specifications, positions 65 through 70)
- Resulting indicators (calculation specifications, positions 54 through 59)
- 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 75 for a description of how an indicator is defined when used with one of these reserved words).

You can specify one, two, or three indicators (positions 10 and 11, 13 and 14, and 16 and 17) on each line. If the indicator must be off to condition the operation, place an N before the appropriate indicator (positions 9, 12, and 15). When more than one indicator is specified in positions 9 through 17, the indicators are in an AND relationship. The indicators on one line or 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 16.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C   25 L1             SUB  TOTAL     TOTAL              A
CL2 10NL3   TOTAL     MULT 05        SLSTAX             B
C*
```

Figure 16. 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 17. 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.

Lines of conditioning indicators in positions 9 through 17 can be placed in AND/OR relationships. AND/OR is specified in positions 7 and 8. A maximum of seven AND/OR lines can be specified in one group.

Some special considerations you should know when using conditioning indicators in positions 9 through 17 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 17, 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 17 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 17, 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 17 are blank, the calculation specified on the line is done at detail calculation time.

Figure 17 through Figure 19 show examples of conditioning indicators.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  AN and OR entries group line of indicators.  When indicators 01, 02,
C*  03, and 04 are on, or when indicators 01, 02, 03, and 05 are on, the
C*  calculation is processed.
C*
C   01 02 03
CAN 04
COR 01 02 03
CAN 05     FIELDA    SUB  FIELDB    QTY     40   23
C*
C*  The L4 calculations are performed under one of three conditions:
C*  01 and 02 are on, but not 03; or 01 and 03 are on, but not 02;
C*  or 02 and 03 are on, but not 01.  Each OR entry in positions
C*  7 and 8 identifies the start of a new group of conditions.
C*
CL4 01 02N03
COR 01N02 03
CORN01 02 03SUM       ADD  SUMTOT    SUMTOT  82H
C*
C*  Seven conditioning indicators must be satisfied before the operation
C*  is processed: L4, 01, 02, 04, 05, and 06 must be on, and 03 must be
C*  off.
C*
CL4 01 02N03
CAN 04 05 06SUM       ADD  SUMTOT    SUMTOT  82H
C*
```

Figure 17 (Part 1 of 2). Calculation Conditioning Indicators in AND and OR Lines

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  This calculation is done under one of three conditions: (1) when
C*  indicator 15 is on, (2) when indicators 20 and 25 are on, or (3)
C*  when indicator 30 is on.  Each OR entry in positions 7 and 8 iden-
C*  tifies the start of a new group of conditions.
C   15
COR 20
CAN 25
COR 30      SUM       ADD  SUMTOT    SUMTOT
C*  In this example, indicators 20 and 25 could have been
C*  coded in an AND relationship on the same line as follows:
C   15
COR 20 25
COR 30      SUM       ADD  SUMTOT    SUMTOT
C
```

Figure 17 (Part 2 of 2). Calculation Conditioning Indicators in AND and OR Lines

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+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 calculated only
I*  if indicator 10 is on.
I*
ITIME    AB  01
I                                       1   7 EMPLNO
I                                       8  100OVERTM      10
I                                      15  202RATE
I                                      21  252RATEOT
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  Field indicator 10 was assigned on the input specifications.
C*  It is used here to condition calculation operations.
C*
C           RATE      MULT 40        WAGE    62H
C   10      OVERTM    MULT RATEOT    OVERPY  62H
C   10      WAGE      ADD  OVERPY    TOTAL   62
```

Figure 18. Conditioning Operations (Field Indicators)

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+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  152SAVE
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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*
C   01               ADD  SAVE      SUM     82
C   02               SUB  SAVE      SUM     82
```

Figure 19. Conditioning Operations (Record Identifying Indicators)

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 23 through 31. 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, positions 33 and 34)
- Record identifying indicators (input specifications, positions 19 and 20)
- Control level indicators (input specifications, positions 59 and 60)
- Field indicators (input specifications, positions 65 through 70)
- Resulting indicators (calculation specifications, positions 54 through 59)
- Indicators set by the RPG/400 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 75 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 20 on page 73). 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 20 on page 73).

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 24 and 25, 27 and 28, and 30 and 31). If these indicators are on, the output operation is done. An N in the position preceding each indicator (positions 23, 26, or 29) 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 14 through 16. 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 SETON operation in calculations. For example, if indicators 10, 12, 14, 16, and 18 are needed to condition an output field, in calculations use the SETON operation to set indicator 20 on if indicators 10, 12, and 14 are on. Then condition the output field with indicators 20, 16, and 18.

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 45 through 70. If 1P is used in an OR relationship with an overflow indicator, the information is printed on every page (see Figure 21 on page 73). 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 22 on page 74).
- To condition certain output records on external conditions, use external indicators to condition those records.

See the Printer File section in the *RPG/400 User'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...
OName++++DFBASbSaN01N02N03Field+YBEnd+PConstant/editword+++++++++...*
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  1     44
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 1      44
O                         INVOIC    10
O                         AMOUNT    18
O                         CUSTR     65
O                    L1   SALSMN    85
```

Figure 20. Output Indicators

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
OName++++DFBASbSaN01N02N03Field+YBEnd+PConstant/editword+++++++++...*
O*
O*  The 1P indicator is used when headings are to be printed
O*  on the first page only.
O*
OPRINT   H  3     1P
O                                    8 'ACCOUNT'
O*
O*  The 1P indicator and an overflow indicator can be used to print
O*  headings on every page.
O*
OPRINT   H  301   1P
O       OR        OF
O                                    8 'ACCOUNT'
```

Figure 21. 1P Indicator

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+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 FIELDAL1
I                                       4   80FIELDB          H1
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++*
C*
C*  When H1 is on, all calculations are bypassed.
C*
C         H1          GOTO END
C                     :
C                     :               Calculations
C                     :
C           END       TAG
ONname++++DFBASbSaN01N02N03Field+YBEnd+PConstant/editword+++++++++...*
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 0201   L1
O                                    50 'HEADING'
O        D 10     01NH1
O                         FIELDA      5
O                         FIELDBZ    15
```

Figure 22. Preventing Fields from Printing

Indicators Referred to As Data

An alternative method of referring to and manipulating RPG/400 indicators is provided by the RPG/400 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, as a result field of a PARM operation, or in a SORTA operation.

*INxx

The field *INxx is a predefined one-position character field where xx represents any one of the RPG/400 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/400 indicator tests may yield undesirable results.

See Figure 23 on page 76 for some examples of indicators referred to as data.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++*
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           MOVE TO IND 50
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        SAVE 61-68
C                   EXSR SUB1
C                   MOVEASAV8      *IN,61           RESTORE 61-68
```

Figure 23 (Part 1 of 2). Examples of Indicators Referred to as Data

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++*
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           SET OFF 71-75
C           70      ADD  CODE      X       30       X=IND TO SETON
C                   MOVE *ON       *IN,X            SETON ONE 71/75
C   71              EXSR CODE1
C   72              EXSR CODE2
C   73              EXSR CODE3
C   74              EXSR CODE4
C   75              EXSR CODE5
```

Figure 23 (Part 2 of 2). Examples of Indicators Referred to as Data

Summary of Indicators

Table 9 and Table 10 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 10 on page 78 indicates the primary condition that causes each type of indicator to be set on and set off by the RPG/400 program. "Function Key Indicators" on page 65 lists the function key indicators and the corresponding function keys.

Table 9. 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, positions 33-34	X						X			
	Record identifying indicator input specifications, positions 19-20	X		X	X	X			X		X
	Control level, input specifications, positions 59-60				X						
	Field level, input specifications, positions 65-70	X		X					X		X
	Resulting indicator, calculation specifications, positions 54-59	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, positions 71-72								X		
	File record relation, input specifications 63-64[3]	X		X	X		X		X		X
	Control level, calculation specifications, positions 7-8				X	X					
	Conditioning indicators, calculation specifications, positions 9-17	X		X	X	X	X	X	X	X	X
	Output indicators, output specifications, positions 23-31	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 SETOF 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 10. When Indicators Are Set On and Off by the RPG/400 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.
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 SETOF 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 4. Control Specifications

The control specification statement, identified by an H in column 6, provides information about generating and running programs. Only one control specification is allowed per program. In the OS/400 system, you can create a data area named RPGHSPEC to contain the information to be used for all RPG/400 programs that do not contain a control specification.

The data area must be a character string 80 positions long. Use the CL command CRTDTAARA (Create Data Area) to create the data area. Specify as the initial value of the data area the entries for the control specification that are to be used. For example, if the DEBUG operation is to be used for all RPG/400 programs, place an initial value of 1 in position 15 of the data area. (See the *CL Reference* for a description of the Create Data Area command.) The library in which the data area is placed must be in the library list when the program is compiled.

The RPG/400 language uses the control specification that is present in the program. In the OS/400 system, if a control specification is not present, the RPG/400 compiler checks for the data area RPGHSPEC in *LIBL. If the data area is not found, the RPG/400 compiler checks for the data area DFTHSPEC in QRPG. If it is not found, a default specification with blanks in positions 7 through 74 is used, (no data area is created). Since QRPG is the product library for the CRTRPGPGM command, if a data area called RPGHSPEC exists in QRPG it will always be found. Use the data area DFTHSPEC in QRPG to create a common control specification for your installation, and use RPGHSPEC in your library to override this specification.

See the description of the individual entries for the meaning of blank entries and for an explanation of the program name. If the default blank specification is used, asterisks are printed on the compiler listing under the Page/Line heading.

Note: For System/38 Environment programs the product library used is QRPG38.

Control Specification Summary Chart

Table 11 (Page 1 of 3). Control Specification Summary Chart

Positions	Name	Entry	Explanation
1-2	Page	Page number	Entry assigns a page number to each specification.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	H	Identification for a control (or header) specification.
7-14		Blank	
15	Debug	Blank	DEBUG and DUMP operations are not used. Compiler-generated symbols are not placed in the symbol table.
		1	DEBUG and DUMP operations are used. Compiler-generated symbols are placed in the symbol table.
16-17		Blank	

Table 11 (Page 2 of 3). Control Specification Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
18	Currency symbol	Blank	Dollar sign ($) is used as currency symbol for editing.
		Currency symbol	Any character except zero (0), asterisk (*), comma (,), period (.), ampersand (&), minus sign (-), letter C, or letter R can be specified as the currency symbol.
19	Date format (user dates)	Blank	Month/day/year format (mmddyy) if position 21 is blank. If position 21 contains a D, I, or J, the day/month/year (ddmmyy) format is used.
		M	Month/day/year (mmddyy). The separator character used depends on the entry in position 20 or position 21.
		D	Day/month/year (ddmmyy). The separator character used depends on the entry in position 20 or position 21.
		Y	Year/month/day (yymmdd). The separator character used depends on the entry in position 20 or position 21.
20	Date edit (Y edit code)	Any character	Separator character used between the fields of the date. If this position is blank, the separator is specified by the entry in position 21.
		&	Blank is used as separator character.
21	Decimal Notation	Blank	Numeric fields and edit codes use a period as decimal notation and a comma for separators. If position 19 is blank, uses mmddyy format. If position 20 is blank, uses a slash (/) as separator for date.
		I	Numeric fields use comma as decimal notation and a period as a separator. If position 19 is blank, uses ddmmyy format. If position 20 is blank, uses a period (.) as separator for date.
		J	Numeric fields use comma as decimal notation and a period as a separator. If position 19 is blank, uses ddmmyy format. If position 20 is blank, uses a period (.) as separator for date.
		D	Numeric fields use a period as decimal notation and a comma as a separator. If position 19 is blank, uses ddmmyy format. If position 20 is blank, uses a slash (/) as separator for date.
22-25		Blank	
26	Alternate collating sequence	Blank	Normal collating sequence is used.
		S	Alternate collating sequence is used.
27-39		Blank	
40	Sign Handling	Blank	The sign is always forced on input and output of zoned numeric fields.
41	Forms Alignment	Blank	First line is printed only once.
		1	First line can be printed repeatedly.
42		Blank	
43	File translation	Blank	No file translation is requested.
		F	Files are to be translated.
44-56		Blank	

Table 11 (Page 3 of 3). Control Specification Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
57	Transparency check	Blank 1	No check for DBCS in literals Check for DBCS in literals
58-74		Blank	
75-80	Program identification		Entry used to assign a unique name to the program. This name can be overridden by the CRTRPGPGM command. If a name is not specified in positions 75 through 80 or on the CRTRPGPGM command, but the source file is a database file, the member name is used as the program name. If the source is not from a database file, the program name defaults to RPGOBJ.

Control Specification Statement

Position 6 (Form Type)

An H must appear in position 6 to identify this line as the control specification (or header) statement.

Positions 7-14 (Reserved)

Positions 7 through 14 must be blank.

Position 15 (Debug)

Entry	Explanation
Blank	DEBUG and DUMP operations are not done. Compiler-generated symbols are not placed in the symbol table.
1	DEBUG and DUMP operations are done. Compiler-generated symbols are placed in the symbol table.

Use position 15 to indicate whether the DEBUG and DUMP operations will be done. A 1 in position 15 when the source program is compiled causes the compiler to generate the object code for these operations. When the program is run, DEBUG and DUMP are performed.

The DEBUG entry also controls the contents of the symbol table that is produced with the program. If position 15 contains a 1, the compiler-generated symbols, starting with a period (.), are placed in the symbol table. If position 15 is blank, the compiler-generated symbols are not placed in the symbol table. The symbol table is printed with a program dump. You can then use the compiler-generated symbols in debugging a program.

See the DEBUG and DUMP operations in Chapter 11, “Operation Codes” for more information.

Positions 16-17 (Reserved)

Positions 16 and 17 must be blank.

Position 18 (Currency Symbol)

Entry	Explanation
Blank	A dollar sign (hexadecimal 5B in EBCDIC hexadecimal 24 in ASCII) is used as the currency symbol in editing (edit words and edit codes).
Currency Symbol	Any character except zero (0), asterisk (*), comma (,), period (.), ampersand (&), minus sign (-), the letter C, or the letter R may be specified as the currency symbol.

The specification of a currency symbol does not affect the user-defined edit codes 5 through 9.

Position 19 (Date Format)

The entry in this position specifies the format of the RPG/400 user dates.

Entry	Explanation
Blank	Defaults to month/day/year if position 21 is blank. Defaults to day/month/year if position 21 contains a D, I, or J.
M	Month/day/year.
D	Day/month/year.
Y	Year/month/day.

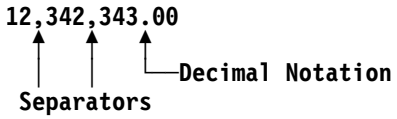
Position 20 (Date Edit)

Entry	Explanation
Ampersand (&)	Blank is used as the separator character
Any nonblank character	The character entered is used as a separator character
Blank	The separator character follows the specification in position 21.

The entry in this position specifies the separator character to be used with the Y edit code.

Position 21 (Decimal Notation)

The entry in this position specifies the notation for the user date. It also specifies the decimal notation and the separator used for numeric literals and edit codes. The term *decimal notation* refers to the character that separates whole numbers from decimal fractions. The word *separator* refers to the character that separates the hundreds position from the thousands position, the hundred thousands position from the millions position, and so on. Below is an example of a number using a period for the decimal notation character and commas for the separator characters.



An entry in this position does not affect the edit words.

Entry	Explanation
Blank	Uses a period for the decimal notation, and a comma for the separator. If position 19 is blank, uses the month/day/year format for user date. If position 20 is blank, uses a slash as the separator for the Y edit code.
I	Uses a comma for the decimal notation, and a period for the separator. If position 19 is blank, uses the day/month/year format for user date. If position 20 is blank, uses a period as the separator for the Y edit code.
J	Uses a comma for the decimal notation, and a period for the separator. If position 19 is blank, uses the day/month/year format for user date. If position 20 is blank, uses a period as the separator for the Y edit code. When you use edit codes that cause zero balances to be printed, zero is written to the left of the decimal notation (comma): 0,00. If the number of decimal positions in the field is equal to the length of the field, the decimal notation (comma) is the leftmost character printed.
D	Uses a period for the decimal notation, and a comma for the separator. If position 19 is blank, uses the day/month/year format for user date. If position 20 is blank, uses a slash as the separator for the Y edit code.

Positions 22-25 (Reserved)

Positions 22 through 25 must be blank.

Position 26 (Alternate Collating Sequence)

Entry	Explanation
Blank	Normal collating sequence is used.
S	Alternate collating sequence is used.

Use position 26 to indicate whether an alternate collating sequence is to be used for character compare operations or match fields. For more information see “Alternate Collating Sequence” on page 427.

Positions 27-39 (Reserved)

Positions 27 through 39 must be blank.

Position 40 (Sign Handling)

Entry	Explanation
Blank	The sign is always forced on input and output of zoned numeric fields.

Position 40 must be blank to ensure a consistent + or – sign when data is extracted or moved from or to numeric input or output fields. When data is moved to unpacked numeric fields in output records, the valid external signs are forced.

Position 41 (Forms Alignment)

Entry	Explanation
Blank	First line is printed only once.
1	First line can be printed repeatedly allowing the operator to adjust the printer forms.

If the program contains more than one printer file, the entry in position 41 applies to each printer file that has 1P (first-page) output. This function may also be specified by the CL command OVRPRTF (Override to Print File) or in the printer device file and can be affected by the ALIGN option on the STRPRTWTR command.

Use column 41 only when the first output line is written to a printer file.

When forms are first put in the printer, they may not be in the correct position. Sometimes several lines must be printed to determine the correct position. If you specify the 1P forms position, the system prints the first line of output and issues a message. The operator can then line up the forms and select the option from the message to print the line again or to continue printing. The 1P forms specification is also valid if the output is spooled. The page counter is not increased until the forms are positioned correctly.

Position 42 (Reserved)

Position 42 must be blank.

Position 43 (File Translation)

Entry	Explanation
Blank	No file translation is requested.
F	Files are to be translated.

An entry of F indicates that a file translation table is to be used to translate all data in specified files. For more information on file translation, see “File Translation” on page 429.

Positions 44-56 (Reserved)

Positions 44 through 56 must be blank.

Position 57 (Transparency Check)

Entry	Explanation
Blank	No check for transparency.
1	Check for transparency.

If you specify 1 in position 57 of the control specification, the RPG/400 compiler scans literals and constants for DBCS characters. It does not check hexadecimal literals. For more information on transparency and DBCS data, see “Where You Can Use DBCS Data in RPG/400 Programs” on page 435, “Transparent Literals and Constants” on page 435, and “Additional Considerations for Using DBCS Data” on page 436.

Positions 58-74 (Reserved)

Positions 58 through 74 must be blank.

Positions 75-80 (Program Identification)

The symbolic name entered in these positions is the name of the program that can be run. You can override this name with the PGM parameter in the CL command CRTRPGPGM (Create RPG Program).

If you do not specify a name in positions 75 through 80 of the control specification or on the CRTRPGPGM command, but the source is from a database file, the member name is used as the program name. If the source is not from a database file, the program name defaults to RPGOBJ.

If you specify the program name on the control specification, its maximum length is 6 characters. If you specify the program name in the CRTRPGPGM command, its maximum length is 10 characters.

Note: Names entered here must follow the RPG/400 symbolic name rules. Extended names are not allowed. However, the name specified on the CRTRPGPGM command can follow the extended naming rules.

Chapter 5. File Description Specifications

File description specifications identify each file used by a program. One file-description specification statement is required for each file in the program.

A maximum of 50 files can be described per program. Only one primary file can be specified; however, the presence of a primary file is not required.

Only one record-address file is allowed per program. You can specify a maximum of eight PRINTER files. The maximum number of other file types is limited only by the maximum number of files allowed for the program.

Program-described files require more entries on the file-description specifications than externally described files. Many of the entries required for a program-described file are part of the external description for an externally described file.

Enter the file-description specifications on the RPG/400 Control and File Description Specifications.

You can specify the following file types:

- Input
- Output
- Update
- Combined (input/output).

You enter file-description specifications on the main file description line. Additional details can be entered on the continuation lines.

Main File Description Line Summary Chart

Table 12 (Page 1 of 4). Main File Description Line Summary Chart

Positions	Name	Entry	Explanation
6	Form type	F	Identification for a file-description specification.
7-14	File name	A valid file name	Every file must have a unique file name that is defined to the OS/400 system. The file name can be from 1 to 8 characters long, and must begin with an alphabetic character.
15	File type	I	Input file.
		O	Output file.
		U	Update file.
		C	Combined (input/output) file.
16	File designation	Blank	Output file.
		P	Primary file.
		S	Secondary file.
		R	Record address file.
		T	Array or table file (prerun-time arrays or tables).
		F	Full procedural file.
17	End of file	E	All records from the file must be processed before the program can end. Not valid for files processed by a record-address file.
		Blank	If position 17 is blank for all files, all records from all files must be processed before end-of-program (LR) can occur. If position 17 is not blank for all files, all records from this file may or may not be processed before end-of-program occurs in multifile processing.
18	Sequence	A or blank	Match fields are in ascending sequence.
		D	Match fields are in descending sequence.
19	File format	F	Program described file.
		E	Externally described file.
20-23		Blank	
24-27	Record length	1-9999	Specifies the length of logical records contained in a program-described file. The device record size constraints may override an excessive record length.

Table 12 (Page 2 of 4). Main File Description Line Summary Chart

Positions	Name	Entry	Explanation
28	Limits processing	L	Sequential-within-limits processing by a record-address file.
		Blank	Random or sequential processing. Random and sequential processing are implied by a combination of positions 16 and 31 of the file-description specifications, and the calculation operation specified.
29-30	Length of key field or record address field	1-99	Length of the key field or the length of each entry in a record-address file. Valid for program-described files only. If the file being defined uses keys for record identification, enter the number of positions to be occupied by each record key. Record key length must be specified for indexed files.
		Blank	These positions must be blank for externally described files. For program described files, a blank entry indicates that keys are not used.
31	Record address type	Blank	• Relative record numbers are used to process the file. • Records are read consecutively. • Record address file contains relative-record numbers. • Keys in record-address-limits file are in same format as keys in file being processed.
		A	Character keys (valid only for program described file specified as indexed file or as record-address-limits file).
		P	Packed keys (valid only for program described files specified as indexed file or record-address-limits file).
		K	Key values are used to process this file. Valid only for externally described files.
32	File organization	Blank	Program described file is processed without keys, or 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.
33-34	Overflow indicator	Blank	No overflow indicator is used.
		0A-0G, 0V	Specified overflow indicator conditions the lines to be printed when overflow occurs. Valid only for program described PRINTER files.
		01-99	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. Valid for both program described and externally described PRINTER files.
35-38	Key field starting location	Blank	Key fields are not used for this program-described file, or the file is externally described.
		1-9999	Record position in a program described file in which the key field begins.

Table 12 (Page 3 of 4). Main File Description Line Summary Chart

Positions	Name	Entry	Explanation
39	Extension code	Blank	No extension or line counter specifications are used.
		E	Extension specifications further describe the file.
		L	Line counter specifications further describe the file.
40-46	Device	PRINTER	The file is a printer file: the printer is used as output device.
		DISK	The file is a disk file: this device supports sequential and random read/write functions.
		WORKSTN	The file is a work station file: input/output is through a display or ICF file.
		SPECIAL	The file is a special file: input or output is associated with a device that is accessed by a user-supplied routine. The name of the routine must be specified in positions 54 through 59. The file must be a fixed unblocked format.
		SEQ	The file is a sequentially organized file: the actual device is specified outside the RPG/400 program.
47-52		Blank	
53	Continuation lines	Blank	This specification is not a continuation line. The following position explanations apply when position 53 is blank.
		K	Indicates a continuation line. For an explanation of positions 54-80 when position 53 contains K, see “Continuation Line” on page 110.
54-59	Routine	Name of user supplied routine	When SPECIAL is the device entry, the routine named in positions 54 through 59 handles the support for the special I/O device.
60-65		Blank	
66	File addition	Blank	No additions of records can be made to an input or update file. For output files, a blank is equivalent to A.
		A	Add records to a DISK file. Positions 16 through 18 of the output specifications must contain ADD, or a WRITE operation code must be used in the calculation specifications.
67-70		Blank	
71-72	File condition	Blank	The file can be used by the program. If it is an input file, it is opened.
		U1-U8	The file can be used by the program when the indicator is on; it is not used when the indicator is off.
		UC	Programmer control of first open. If a file is to be opened by an OPEN operation in the calculation specification, a UC entry causes the file not to be opened at program initialization. Not valid for primary, secondary, table, or record address input files, or for output files conditioned by 1P indicator.
73-74		Blank	

Table 12 (Page 4 of 4). Main File Description Line Summary Chart			
Posi-tions	**Name**	**Entry**	**Explanation**
75-80		Optional	This space is available for comments.

File Description Specification Statement

Position 6 (Form Type)

An F must be entered in this position for file-description specifications.

Positions 7-14 (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 8 characters long, and must begin in position 7.

Each file used in the program is identified by a unique symbolic name in positions 7 through 14.

The file name specified in positions 7 through 14 must be an existing file name that has been defined to the OS/400 system, or one of the OS/400 system override commands must be used to associate the RPG/400 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. The file name specified in these positions, rather than the device name specified in positions 40 through 46, is used to point to the file. 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/400 device name defines the functions that can be processed on the associated file.

You can specify file names in positions 7 through 14 that correspond to predefined device-file definitions supplied by IBM.

Program Described File

For program-described files, the file name entered in positions 7 through 14 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 the file is an input file and records are also being added to the file
- Extension specifications if the file is a table, array, or record address file, or a file processed by a record-address file
- Calculation specifications if the file name is required for the operation code specified
- Line counter specifications if the device is a printer and default values are to be overridden.

Externally Described File

For externally described files, the file name entered in positions 7 through 14 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/400 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.
- A logical file with two record formats of the same name cannot have the duplicate format names renamed and cannot be externally described. However, such a file can be accessed if it is program described.

Position 15 (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 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 16 contains T (an array or table replacement file).

Position 16 (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 multifile processing, processing of the primary file takes precedence. Only one primary file is allowed per program.

Secondary File

When more than one file is used during cycle-controlled programming, secondary files are input files. The processing of secondary files is determined by the order in which they are specified in the file-description specifications and on the rules of multifile logic.

Record Address File

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 and extension specifications and not on the input specifications. The file processed by the record-address file must also be specified on the extension specifications and must be a primary, secondary, or full procedural file.

You cannot specify a record-address file for the device SPECIAL. You cannot specify an externally described file as a record-address file; however, you can use a record-address file to process a program described file or an externally described file.

A record-address file that contains relative-record numbers must also have a T specified in position 32 and an F in position 19.

Array or Table File

Array and table files specified by a T in position 16 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 16, you can specify C in position 15 for a DISK or SEQ file. This C allows an array or table file to be read from or written to the same file (an array or table replacement file). The To and From file names on the extension specifications must specify this file name.

Full Procedural File

This entry is used when the input is controlled by calculation operations. File operation codes such as CHAIN or READ are used to do input functions.

Position 17 (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.
Blank	If position 17 is blank for all files, all records from all files must be processed before end of program (LR) can occur. If position 17 is not blank for all files, all records from this file may or may not be processed before end of program occurs in multifile processing.

Use position 17 to indicate whether the program can end before all records from the file are processed. An E in position 17 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 17 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 17 have been processed. If position 17 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 17 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 17.
- 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 nonmatching 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 17.

Position 18 (Sequence)

Entry	Explanation
A or blank	Match fields are in ascending sequence.
D	Match fields are in descending sequence.

Position 18 specifies the sequence of input fields used with the match fields specification (positions 61 and 62 of the input specifications). Position 18 applies only to input, update, or combined files used as primary or secondary files. Use positions 61 and 62 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 18 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 18 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 61 and 62 of the input specifications, and by entering an A or D in position 18, 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/400 exception/error handling routine is given control.

Position 19 (File Format)

Entry	Explanation
F	Program described file
E	Externally described file

An F in position 19 indicates that the records for the file are described within the RPG/400 program on input/output specifications.

An E in position 19 indicates that the record descriptions for the file are external to the RPG/400 source program. The compiler obtains these descriptions at compilation time and includes them in the source program.

An entry is required in position 19.

Positions 20-23 (Reserved)

Positions 20 through 23 must be blank. (Block length, if allowed, is specified outside the RPG/400 program.)

Positions 24-27 (Record Length)

Use positions 24 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 9999; 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/400 programs to use System/36 environment SORT files as record-address files.

Table 13. Valid Combinations for a RAF Table File		
Record Length Positions 24-27	**RAF Length Positions 29-30**	**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 16 and 31 of the file-description specifications, and by the calculation operation specified.

The operation codes SETLL and SETGT can be used to position a file; however, the use of these operation codes does not require an L in position 28.

For more information on limits processing, refer to the *RPG/400 User's Guide.*

Positions 29-30 (Length of Key or Record Address)

Entry	Explanation
1-99	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.
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. Posi-

tions 29-30 can also be blank for a record-address file with a b in positions 24-27 (record length).

Position 31 (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. Keys in record-address-limits file are in the same format as keys in 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).
K	Key values are used to process the file. This entry is valid only for externally described files.

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 32), 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 31 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 31 and by the use of the RECNO keyword on the file description specifications.

A = Character Keys

The indexed file (I in position 32) 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 and 30 and starting position in positions 35 through 38).

The record-address-limits file (R in position 16) 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 31.

P = Packed Keys

The indexed file (I in position 32) 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 and 30 and starting position in positions 35 through 38).

Note: The sign of all decimal numeric input fields is forced to F or D. All numeric result fields specified by calculation specifications also have an F or D sign. There-

fore, if the sign of the key field in the file is not F or D, a record-not-found error occurs when you retrieve that file.

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 31.

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.

For more information on record address type, refer to the *RPG/400 User's Guide*.

Position 32 (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 32 to identify the organization of program described files.

Indexed Files

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.

Nonkeyed Program-Described File

A program-described file that is processed without keys can be processed:

- Randomly by relative-record numbers, positions 28 and 31 must be blank.
- Entry Sequence, positions 28 and 31 must be blank.
- As a record-address file, position 28 must be blank.

Record Address File

A record-address file (indicated by an R in position 16) that contains relative-record numbers must be identified by a T in position 32. (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 System/36 Environment record-address files, see the *RPG/400 User's Guide*.

Positions 33-34 (Overflow Indicator)

Entry	Explanation
Blank	No overflow indicator is used.
0A-0G, 0V	Specified overflow indicator conditions the lines to be printed when overflow occurs.
01-99	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.

Indicators 0A through 0G, and 0V are not valid for externally described files.

Use positions 33 and 34 to specify an overflow indicator to condition which lines in each PRINTER file will be printed when overflow occurs. This entry is valid only for a PRINTER device. Overflow only occurs if defined.

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.

Positions 35-38 (Key Field Starting Location)

Entry	Explanation
Blank	Key fields are not used for this program-described file, or the file is externally described.
1-9999	Record position in a program described indexed file in which the key field begins.

Use positions 35 through 38 to identify the record position in which the key field for a program described indexed file begins. An entry must be made in these positions for a program described indexed file. 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. The entry in these positions must be right-adjusted. Leading zeros can be omitted.

Position 39 (Extension Code)

Entry	Explanation
Blank	No extension or line-counter specifications are used.
E	Extension specifications further describe the file.
L	Line counter specifications further describe the file.

Use position 39 to indicate whether the program-described file is further described on the extension specifications or on the line counter specifications. An E in position 39 applies only to array or table files or to record-address files; an L in position 39 applies to files assigned to the PRINTER device.

Positions 40-46 (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 routine. The name of the routine must be specified in positions 54 through 59. A parameter list is created for use with this routine, including an option code parameter and a status code parameter. See "Special File" on page 431 for more information. The file must be a fixed unblocked format.
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 40 through 46 to specify the RPG/400 device name to be associated with the file. On the AS/400 system the file name in positions 7 through 14, rather than the device name specified in positions 40 through 46, is used to point to the file. The RPG/400 device name defines the RPG/400 functions that can be done on the associated file. Certain functions are valid only for a specific RPG/400 device name, such as the EXFMT operation for WORKSTN. The file name specified in positions 7 through 14 can be overridden at compilation time or run time, allowing you to change the input/output device used in the program.

Note that the RPG/400 device names are not the same as the system device names.

Positions 47-52 (Reserved)

Positions 47 through 52 must be blank.

Position 53 (Continuation Lines)

A K in position 53 indicates a continuation line. See "Continuation Line" on page 110 for more information.

Positions 54-59 (Routine)

When SPECIAL is the device entry (positions 40 through 46), the routine named in positions 54 through 59 handles the support for the special I/O device. The routine name must be left-adjusted. The name is used by the compiler to produce the linkage to the routine.

Positions 60-65 (Reserved)

Positions 60 through 65 must be blank.

Position 66 (File Addition)

Position 66 indicates whether records are to be added to a DISK file.

Entry	Explanation
Blank	No records can be added to an input or update file (I or U in position 15). For an output file (O in position 15), a blank is equivalent to an A.

A — Add records to the file. Positions 16 through 18 of the output record specifications for this file must contain ADD, or the WRITE operation code must be used in the calculation specifications.

See Table 14 for the relationship between position 15 and position 66 of the file-description specifications and positions 16 through 18 of the output specifications.

Table 14. Processing Functions for Files

Function	File Description Specifications Position 15	File Description Specifications Position 66	Output Specifications Positions 16-18
Create new file[1]	O	Blank	Blank
or			
Add records to existing file	O	A	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.

Positions 67-70 (Reserved)

Positions 67 through 70 must be blank.

Positions 71-72 (File Condition)

Entry	Explanation
Blank	The file can be used by the program, and, if it is an input file, the file is opened.
U1-U8	The file can be used by the program when the indicator is on; it is ignored when the indicator is off.
UC	Programmer control of first open. If a file is to be initially opened by the OPEN operation in the calculation specifications, then a UC entry causes the file not to be opened at program initialization. This entry is not valid for input files designated as primary, secondary, table, or record-address files, or for output files conditioned by the 1P indicator.

An entry of U1 through U8 in positions 71 and 72 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 U1 through U8 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 U1 through U8 indicators in this manner has no effect on file conditioning.

The UC entry is required for programmer control of only the first file opening. If a file is opened and later is closed by the CLOSE operation, the programmer can reopen the file (by the OPEN operation) and the UC entry is not required in positions 71 and 72.

Positions 73-74 (Reserved)

Positions 73 and 74 must be blank.

Positions 75-80 (Comments)

Positions 75 to 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

File Types and Processing Methods

Table 15 shows the valid entries for positions 28, 31, and 32 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 15 (Page 1 of 2). Processing Methods for DISK Files

Access	Method	Opcode	Position 28	Position 31	Position 32	Explanation
Random	RRN	CHAIN	Blank	Blank	Blank	Access by physical order of records
Seq	Key	READ READE READP REDPE cycle	Blank	Blank	I	Access by key sequentially

Table 15 (Page 2 of 2). Processing Methods for DISK Files

Access	Method	Opcode	Position 28	Position 31	Position 32	Explanation
Seq	Within Limits	READ READE READP REDPE cycle	L	A or P	I	Access by key sequentially controlled by record-address-limits file
Seq	RRN	READ READE READP REDPE cycle	Blank	Blank	T	Access sequentially restricted to RRN numbers in RAF file

Random-by-Key Processing

For the random-by-key method of processing, you specify a search argument that identifies the key of the record to be read in factor 1 of the calculation specifications for the CHAIN operation. See the section on "Keyed Processing Examples" in Chapter 7 of the *RPG/400 User's Guide* for an example of an externally described DISK file being processed randomly by key. The specified record can be read from the file either during detail calculations or during total calculations.

The random-by-key method of processing is valid for a full-procedural file designated as an input file or an update file.

For an externally described file, position 31 of the file description specifications must contain K, which indicates that the file is processed according to an access path that is built on keys. The data description specifications for the file specifies the field that contains the key value (the key field). Position 32 of the file-description specifications must be blank.

You must designate a program-described file as an indexed file (I in position 32), and position 31 of the file-description specifications must contain an A or a P. The length of the key field is identified in positions 29 and 30 of the file-description specifications, and the starting location of the key field is identified in positions 35 through 38. Data description specifications must be used to create the access path for a program-described input file. Refer to the section "Indexed File" in chapter 7 of the *RPG/400 User's Guide*.

Figure 24 on page 104 through Figure 27 on page 109 are processing charts for DISK files.

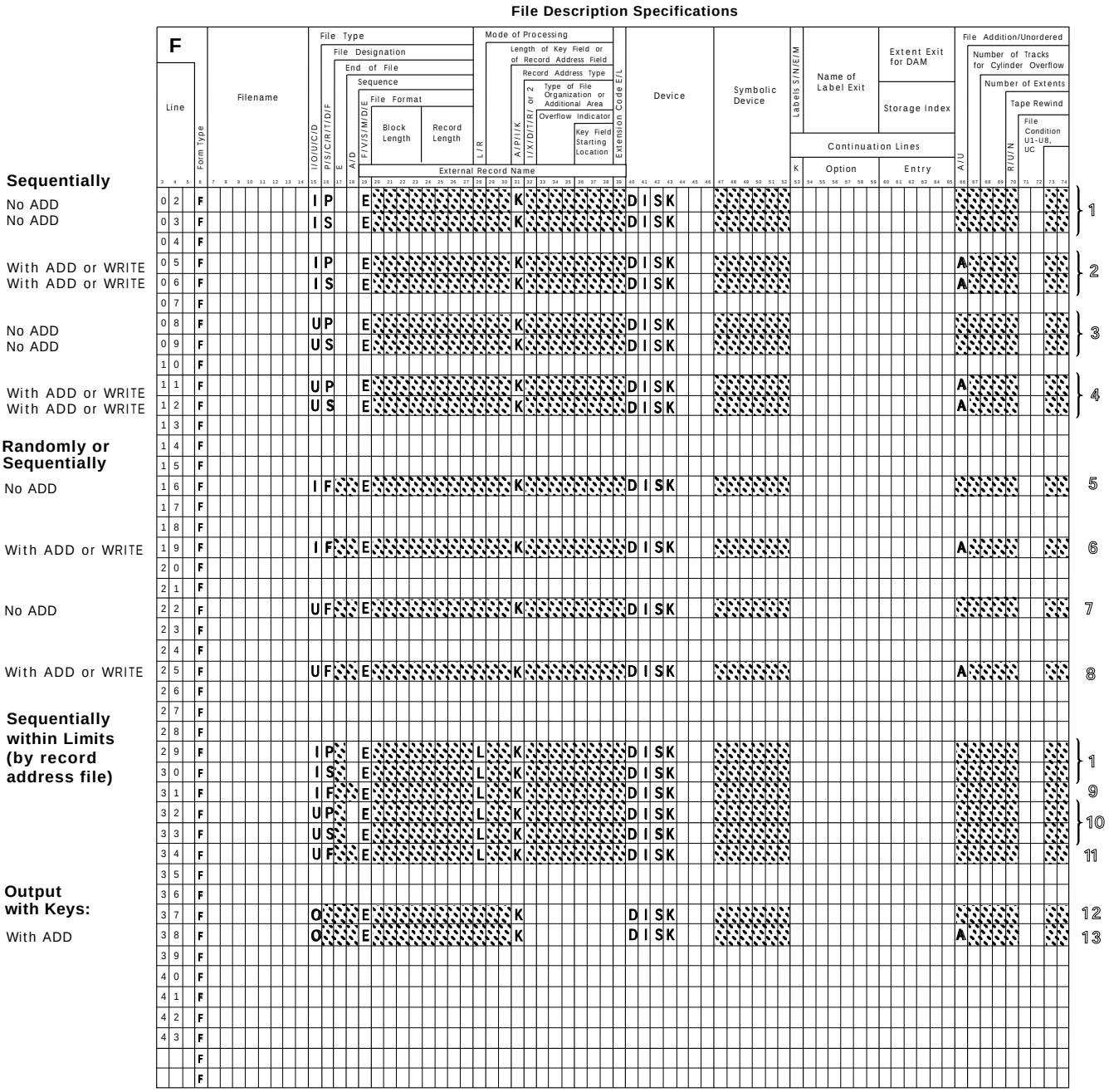


Figure 24. Processing Chart for Externally Described DISK Files (Processing with Keys)

Valid File Operations for Figure 24 on page 104

1. CLOSE, FEOD, FORCE
2. WRITE, CLOSE, FEOD, FORCE
3. UPDAT, DELET, CLOSE, FEOD, FORCE
4. WRITE, UPDAT, DELET, CLOSE, FEOD, FORCE
5. READ, READE, READP, REDPE, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
6. WRITE, READ, READE, READP, REDPE, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
7. READ, READE, READP, REDPE, SETLL, SETGT, CHAIN, UPDAT, DELET, OPEN, CLOSE, FEOD
8. READ, READE, READP, REDPE, SETLL, SETGT, CHAIN, WRITE, UPDAT, DELET, OPEN, CLOSE, FEOD
9. READ, OPEN, CLOSE, FEOD
10. UPDAT, CLOSE, FEOD, FORCE
11. READ, UPDAT, OPEN, CLOSE, FEOD
12. WRITE (initial load or extend), OPEN, CLOSE, FEOD
13. WRITE (add records), OPEN, CLOSE, FEOD

Note: Shaded positions must be blank. Positions without entries are program dependent.

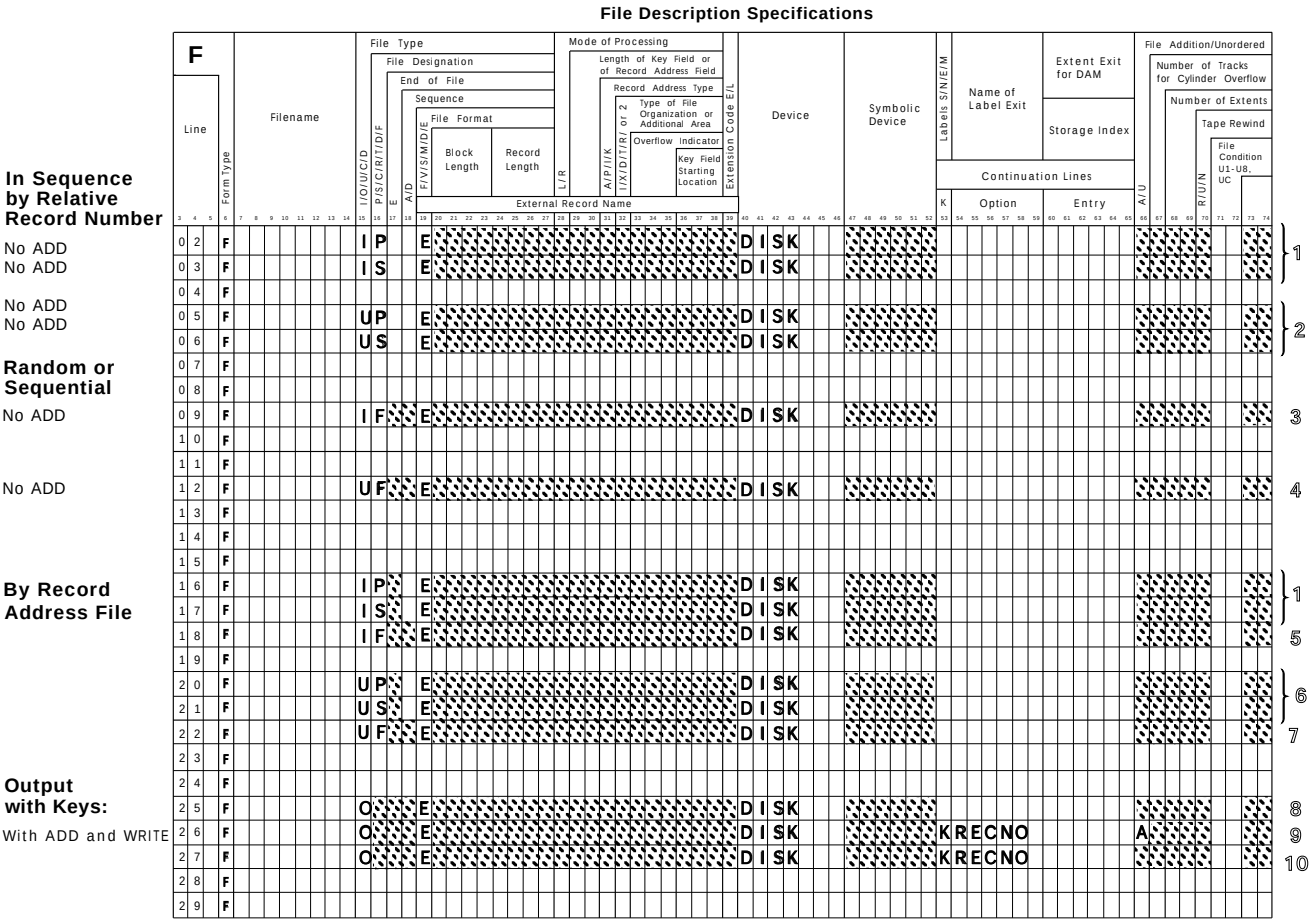


Figure 25. Processing Chart for Externally Described DISK Files (Processing without Keys)

Valid File Operations for Figure 25

1. CLOSE, FEOD, FORCE
2. UPDAT, DELET, CLOSE, FEOD, FORCE
3. READ, READP, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
4. READ, READP, SETLL, SETGT, CHAIN, UPDAT, DELET, OPEN, CLOSE, FEOD
5. READ, OPEN, CLOSE, FEOD
6. UPDAT, CLOSE, FEOD, FORCE
7. READ, UPDAT, OPEN, CLOSE, FEOD
8. WRITE (initial load or extend), OPEN, CLOSE, FEOD
9. WRITE (add records), OPEN, CLOSE, FEOD
10. WRITE (initial load), OPEN, CLOSE, FEOD

Notes:

1. Shaded positions must be blank. Positions without entries are program dependent.
2. The RECNO option must be specified in positions 54 through 59 for an output file that uses relative-record numbers. The option can be specified for input/output files. The RPG/400 compiler places the relative-record number of any retrieved record in the RECNO field.

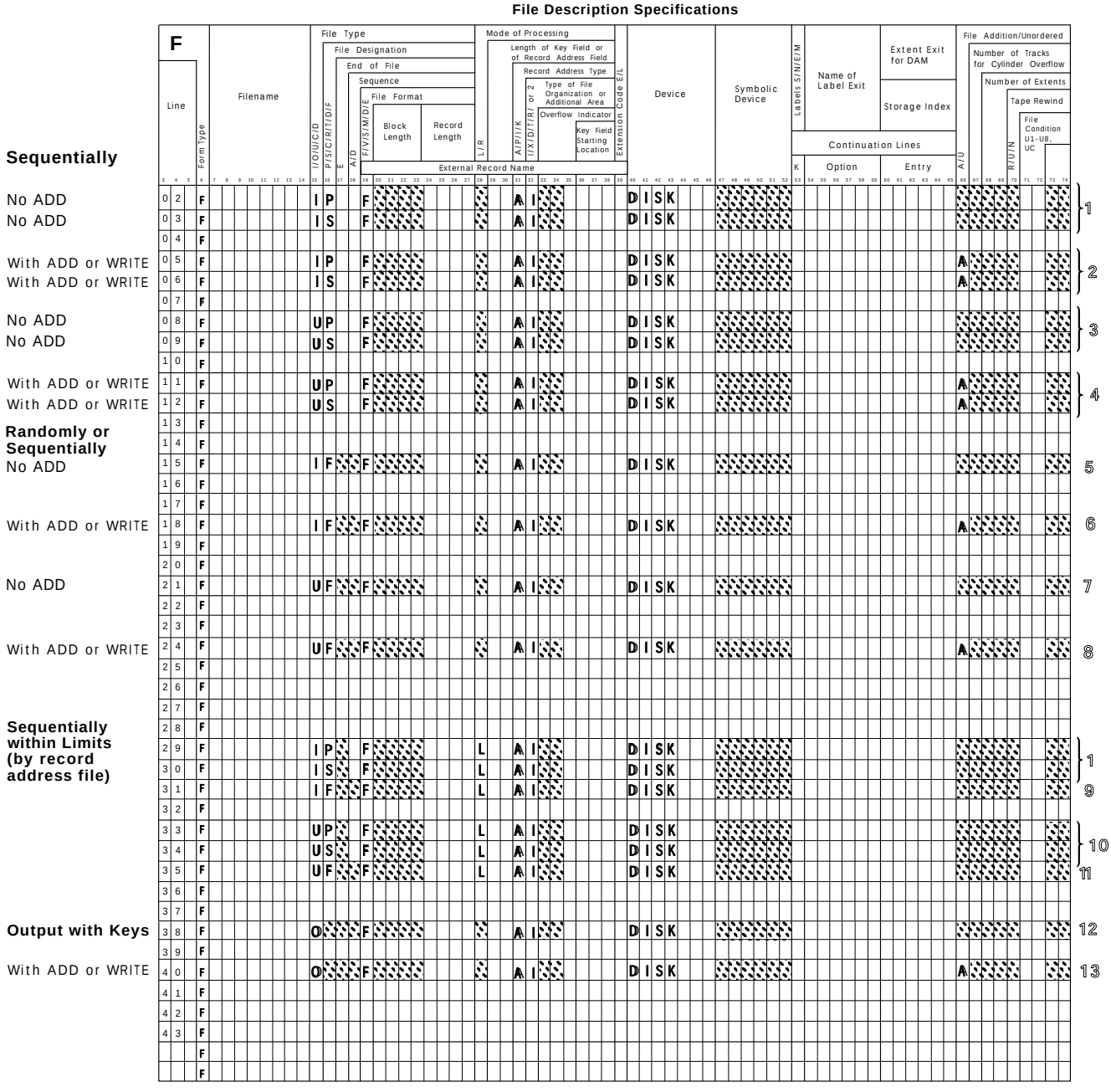


Figure 26. Processing Chart for Program-Described DISK Files (Processing with Keys)

Valid File Operations for Figure 26 on page 107

1. CLOSE, FEOD, FORCE
2. WRITE, CLOSE, FEOD, FORCE
3. UPDAT, DELET, CLOSE, FEOD, FORCE
4. UPDAT, DELET, WRITE, CLOSE, FEOD, FORCE
5. READ, READE, READP, REDPE, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
6. WRITE, READ, READE, READP, REDPE, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
7. READ, READE, READP, REDPE, SETLL, SETGT, CHAIN, UPDAT, DELET, OPEN, CLOSE, FEOD
8. WRITE, UPDAT, DELET, READ, READE, READP, REDPE, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
9. READ, OPEN, CLOSE, FEOD
10. UPDAT, CLOSE, FEOD, FORCE
11. READ, UPDAT, OPEN, CLOSE, FEOD
12. WRITE (initial load), OPEN, CLOSE, FEOD
13. WRITE (add records), OPEN, CLOSE, FEOD

Notes:

1. Shaded positions must be blank. Positions without entries are program dependent.
2. The entry in position 32 (A or P) depends on the format of the keys in the file.
3. All WRITE and UPDAT operations to a program-described file require a data-structure name in the result field.

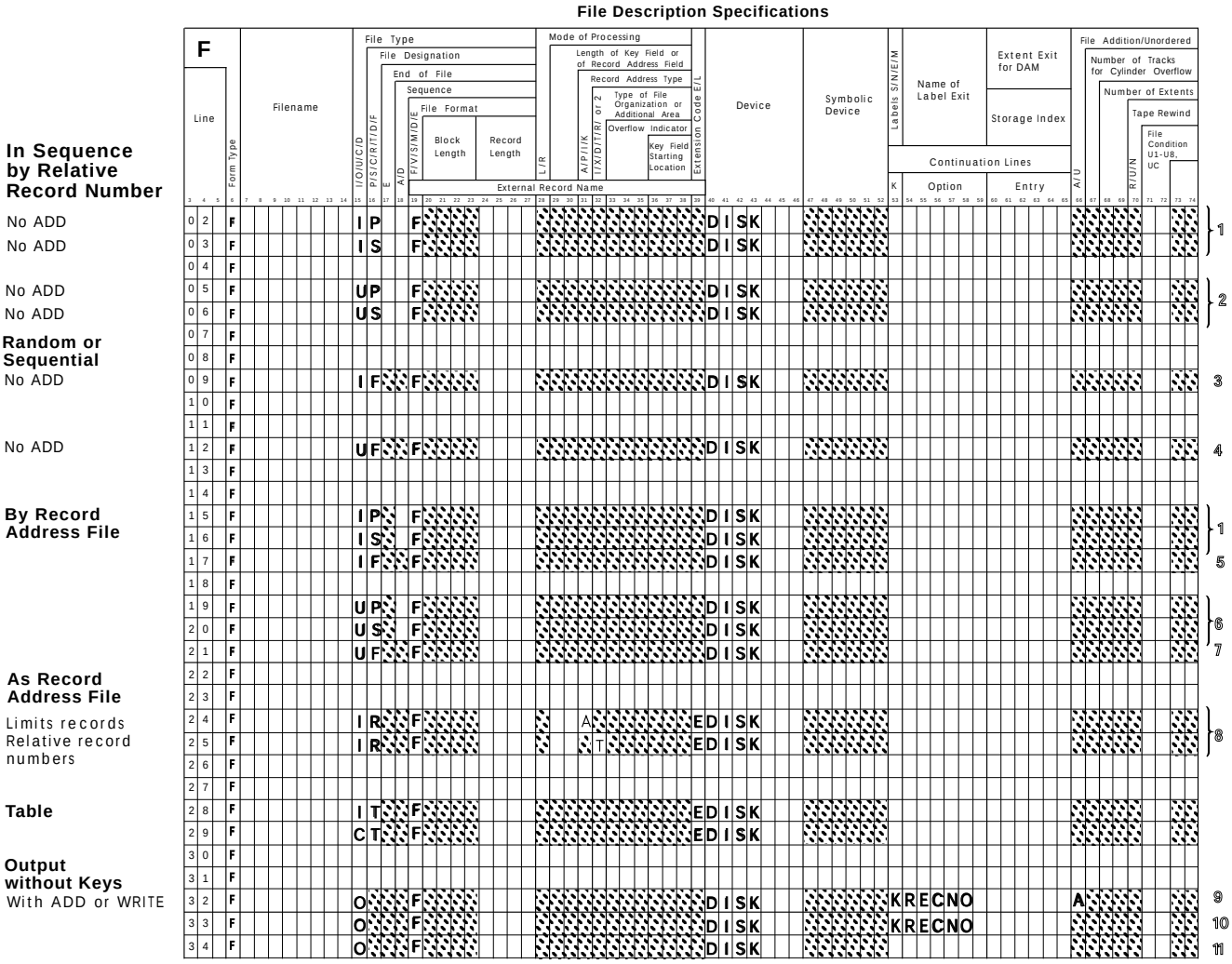


Figure 27. Processing Chart for Program-Described DISK Files (Processing without Keys)

Valid File Operations for Figure 27

1. CLOSE, FEOD, FORCE
2. UPDAT, DELET, CLOSE, FEOD, FORCE
3. READ, READP, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
4. READ, READP, SETLL, SETGT, CHAIN, UPDAT, DELET, OPEN, CLOSE, FEOD
5. READ, OPEN, CLOSE, FEOD
6. UPDAT, CLOSE, FEOD, FORCE
7. READ, UPDAT, OPEN, CLOSE, FEOD
8. OPEN, CLOSE, FEOD
9. WRITE (add records), OPEN, CLOSE, FEOD
10. WRITE (initial load), OPEN, CLOSE, FEOD
11. WRITE (initial load or extend), OPEN, CLOSE, FEOD

Notes:

1. Shaded positions must be blank. Positions without entries are program dependent.
2. The RECNO option must be specified in positions 54 through 59 for an output file that uses relative-record numbers. The option can be specified for input/update files. The RPG/400 compiler places the relative-record number of any retrieved record in the RECNO field.
3. All WRITE and UPDAT operations to a program-described file require a data-structure name in the result field.

Continuation Line

Continuation lines can be specified on the file description specification to provide additional information about the file being defined. Any number of continuation lines can be specified. A continuation line is indicated by a K in position 53 (see Figure 28).

A continuation line can be specified on the main file-description specification line if the functions use positions 54 through 65 for their definition; however, the keywords SFILE, RENAME, IGNORE, and PLIST cannot be defined there. To specify the continuation line information on the main file description specification line, enter K in position 53 and the valid entries in positions 54 through 67.

Valid entries for continuation lines are given in the two summary charts below.

Specifications–Continuation Line

Figure 28. RPG/400 Control and File Description

Continuation Line Summary Chart

Table 16 (Page 1 of 2). Continuation Line Summary Chart

Positions	Name	Entry	Explanation
1-2	Page	Page number	Entry assigns a page number to each specification form.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	F	Identification for a file description specification.
7-18		Blank	These positions must be blank for a separate continuation line.
19-28		External name of record format	These positions are used to specify the external name of the record format that is to be renamed (RENAME) or ignored (IGNORE).
29-46		Blank	These positions must be blank for a separate continuation line.

Table 16 (Page 2 of 2). Continuation Line Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
47-52	Record number field for SFILE	Numeric field name	For the SFILE options, these positions must specify the name of a Relative Record Number (RECNO) field. For other continuation line options, these positions must be blank.
53	Continuation line	K	Indicates a continuation line.
54-59, 60-67			These positions are used together. Positions 54 through 59 specify the option, while positions 60 through 67 provide further explanation of the option. See "Continuation Line Options Summary Chart" on page 111.
68-74		Blank	These positions must be blank for a separate continuation line.
75-80		Optional	This space is available for comments.

Continuation Line Options Summary Chart

The valid entries for positions 54 through 67 are:

Table 17 (Page 1 of 4). Continuation Line Options		
Option (54-59)	**Entry (60-67)**	**Explanation**
COMIT	Blank	This file is specified for commitment control. Use the COMIT and ROLBK operation codes to group changes to this file so that the changes all happen together, or do not happen at all.
ID	Field name	Positions 60-65 contain the left-justified name of a 10-character alphanumeric field which need not be further defined. This field 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. When moving a literal into the field, blank the field first, and use the MOVEL operation to place the literal left-justified in the field. 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 ID field is maintained for each call to a program. If you call program B from within program A, the ID field for program A is not affected. Program B uses a separate ID field. When you return to program A, its ID 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 ID but not NUM, the RPG/400 program assumes NUM is present with a value of 1. To determine the name of the requester device, you may look in the appropriate area of the file information data structure. Or, you may process one of the input or output operations described above with the ID field blank. After the operation, the ID field has the name of the requester device.

Table 17 (Page 2 of 4). Continuation Line Options

Option (54-59)	Entry (60-67)	Explanation
IGNORE	Blank	This option lets you ignore a record format from an externally described file. On the continuation line, positions 19 through 28 specify the external name of the record format to be ignored, and positions 60 through 67 must be blank. The program runs as if the record format did not exist.
IND	Indicator 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 01 through 99, right-justified, in positions 60 through 65. No indicators are saved and restored if IND is not specified or if the option NUM has the entry 1. If you specified the keyword INDARA, the number you specify for IND 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 IND is 54. IND must not be used with shared files. When you specify IND but not NUM, the RPG/400 program assumes NUM is present with a value of 1.
INFDS	Data structure name	This entry lets you define and name a data structure to contain the exception/error information. The data structure name is entered in positions 60 through 65 and left justified. If INFDS is specified for more than one file, each associated data structure must have a unique name.
INFSR	Subroutine name	The file exception/error subroutine named (left justified) in positions 60 through 65 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.
NUM	Maximum number of devices	The number specified must be greater than zero and right-justified in positions 60 through 65. The lesser of this number and the number of devices defined for the WORKSTN file on the create-file command is the maximum number of devices that this file can acquire. With a shared file, the NUM value is not used to restrict the number of acquired devices. When you specify ID, IND, or SAVDS but not NUM, the RPG/400 program assumes NUM is present with a value of 1.
PASS	*NOIND	Specify PASS *NOIND on the file description specification continuation line for a program described WORKSTN file if you are taking responsibility for passing indicators on input and output. With PASS *NOIND, the RPG/400 language does not pass indicators to data management on output and does not receive them on input. Pass indicators by describing them as fields (in the form *INxx, *IN, or *IN,xx) 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 you do not specify PASS *NOIND and you use the keyword INDARA in the DDS for the WORKSTN file, indicators are not passed to data management on output nor received from data management on input.

Table 17 (Page 3 of 4). Continuation Line Options

Option (54-59)	Entry (60-67)	Explanation
PLIST	Parameter list name	This entry is valid only when the device specified in positions 40 through 46 of the main file-description line is SPECIAL. Positions 60 through 65 give the left-justified name of the parameter list to be passed to the special routine. The parameters identified by this entry are added to the end of the parameter list passed by the program.
PRTCTL	Data structure name	The dynamic printer control option is being used. The data structure specified left-justified in positions 60 through 65 refers to the forms control information and line count value. The PRTCTL option is valid only for a program described file. See "PRTCTL Data Structure" on page 114 for a description of the predefined positions for this data structure.
RECNO	Field name	This entry is optional for disk files to be processed by relative-record number. A RECNO field 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. RECNO can be specified for input/update files. The relative-record number of the record retrieved is placed in the field named, left justified, in positions 60 through 65 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. RECNO is valid for DISK files only. The contents of positions 60 through 65 may be not valid when the RPG/400 compiler does the blocking and unblocking of records.
RENAME	Record format name	This entry, which is optional, allows you to rename record formats in an externally described file. Positions 19 through 28 of the continuation line specify the external name of the record format that is to be renamed. Positions 60 through 67 specify the left-justified name of the record as it is used in the program. The external name is replaced by this name in the program.
SAVDS	Data structure name	Positions 60-65 contain the left-justified name 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 SAVDS is not specified, no saving and restoring is done. SAVDS must not be specified for shared files. When you specify SAVDS but not NUM, the RPG/400 program assumes NUM is present with a value of 1.

Table 17 (Page 4 of 4). Continuation Line Options

Option (54-59)	Entry (60-67)	Explanation
SFILE	Record format name	If the main file-description line contains E in position 19 and WORKSTN in positions 40 through 46, this option must be used to define any subfiles to be used in the file. Positions 60 through 67 must specify, left justified the RPG/400 name of the record format to be processed as a subfile. Positions 47 through 52 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 in positions 47 through 52. This field is also used to specify the record number that RPG/400 uses for a WRITE operation to the subfile or for output operations that use ADD. The field name specified in positions 47 through 52 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 a separate continuation line. Do not use SFILE with SLN.
SLN	Field name	Positions 60-65 contain the left-justified name of a start line number (SLN) field. The SLN field determines where a record format is written to a display file. The main file-description line must contain WORKSTN in positions 40 through 46 and a C or O in positions 15. The data description specifications for the file must specify the keyword SLNO(*VAR) for one or more record formats. When you specify SLN on the continuation line, the SLN field will automatically be defined in the program as a numeric field with length of 2 and with 0 decimal positions. Do not use SLN with SFILE.

PRTCTL Data Structure

Data Structure Positions	Subfield Contents
1	A one-position character field that contains the space-before value
2	A one-position character field that contains the space-after value
3-4	A two-position character field that contains the skip-before value
5-6	A two-position character field that contains the skip-after value
7-9	A three-digit numeric field with zero decimal positions that contains the current line count value.

The values contained in the first four subfields of the data structure are the same as those allowed in positions 17 through 22 (space and skip entries) of the output specifications. If the space and skip entries (positions 17 through 22) 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 17 through 22. 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/400 compiler does not initialize subfield 5 until after the first output line is printed. The RPG/400 compiler then changes subfield 5 after each output operation to the file.

Chapter 6. Extension Specifications

Extension specifications describe all record address files, arrays, and tables. A maximum of 200 arrays and tables can be used in a program.

The following figure shows the valid positions for arrays, tables, and record address files.

Extension Specifications

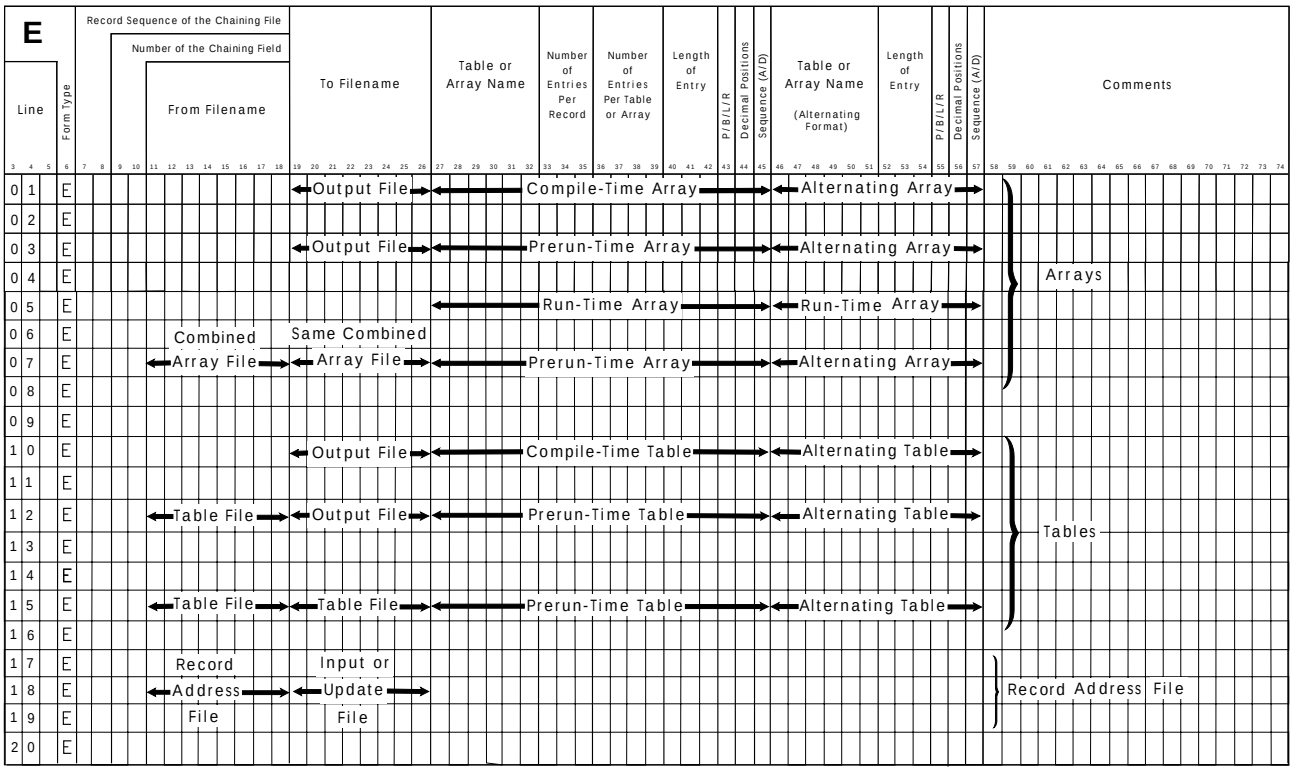


Figure 29. Possible Extension Specifications Entries

Enter these specifications on the RPG/400 Extension Specifications.

Extension Specification Summary Chart

Table 18 (Page 1 of 3). Extension Specifications Summary Chart

Positions	Name	Entry	Explanation
1-2	Page	Page Number	Entry assigns a page number to each specification.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	E	Identification for an extension specification.
7-10		Blank	

Table 18 (Page 2 of 3). Extension Specifications Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
11-18	From file name	Blank	The array or table is loaded at compilation time or by input or calculation specifications.
		Record address file name	Name of the record address file.
		Array or table file name	The array or table file loaded at prerun time.
19-26	To file name	Blank	The array or table is not written at the end of the program.
		Name of an input or update file containing data records	File processed with the record address file named in positions 11 through 18.
		Name of an output or combined file	File (output or combined) to which an array or table is to be written.
27-32	Array or table name	Array or table name	The name of array or table used in the program.
33-35	Entries per record	Blank	The array is loaded by input or calculation specifications.
		1-999	Number of array or table entries in each array or table input record.
36-39	Entries per array or table	1-9999	Maximum number of array or table entries.
40-42	Length of entry	1-256	Length of each element in the array or table named in positions 27 through 32.
43	Data format	Blank	The data for the array or table is in zoned decimal format or in character 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.
44	Decimal positions	Blank	Character array or table.
		0-9	Number of positions to the right of the decimal in numeric array or table elements.
45	Sequence	Blank	No particular sequence.
		A	Ascending sequence.
		D	Descending sequence.

Table 18 (Page 3 of 3). Extension Specifications Summary Chart

Positions	Name	Entry	Explanation
46-51	Array or table name	Array or table name	The name of the table or array used in the program as the alternate format.
52-54	Length of entry	1-256	Length of each element in the array or table named in positions 46-51.
55	Data format	Blank	The data for the array or table is in zoned decimal format or in character 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.
56	Decimal positions	Blank	Character array or table.
		0-9	Number of positions to the right of the decimal in numeric array or table elements.
57	Sequence	Blank	No particular sequence.
		A	Ascending sequence.
		D	Descending sequence.
58-74		Comments	Used to document the purpose of each specification line.
75-80		Optional	This space is available for comments.

Extension Specification Statement

Position 6 (Form Type)

An E must appear in position 6 to identify this line as an extension specifications statement.

Positions 7-10 (Reserved)

Positions 7 through 10 must be blank.

Positions 11-18 (From File Name)

Entry	Explanation
Blank	The array or table is loaded at compilation time, or the array is loaded by input or calculation specifications.
Record address file name	Name of the record address file.
Array or table file name	Name of the array or table file loaded at prerun time.

Use positions 11 through 18 to name an array file, table file, or record address file. File names must begin in position 11. The record address file name must always be entered in these positions. The file name of every prerun-time array or table used in the program must be entered in these positions. Leave positions 11 through 18 blank for compile-time arrays or tables and for run-time arrays loaded with input and/or calculation specifications.

Table 19 on page 121 shows the relationship between positions 11 through 18 and positions 19 through 26.

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. Only those arrays and tables that contain constant data should be compiled with the program.

Positions 19-26 (To File Name)

Entry	Explanation
Blank	The array or table is not written at end of program.
Input or update file with data	File processed with the record address file named in positions 11 through 18.
Output or combined file	Output file to which an array or table is to be written, or the same file name (must be a combined table file) specified in positions 11 through 18 if the output array or table is to replace input in the same file. The file should be externally described as a physical file.

If a record address file is named in positions 11 through 18, the name of the input or update file that contains the data records to be processed must be entered in positions 19 through 26.

If an array or table is to be written, enter the file name of the output or combined file in positions 19 through 26. This file must also be named in the file description specifications. An array or table can be written to only one output device. Leave positions 19 through 26 blank if the array or table is not to be written.

If an array or table is assigned to an output file, it is automatically written if LR is on. The array or table is written after all other records are written in the format used when it was entered.

If an array or table is to be written to the same file from which it was read, the same file name must be entered in positions 11 through 18 and in positions 19 through 26. This file must be specified as a combined file (C in position 15) in the file description specifications.

Table 19. From and To File Name Entries		
Type of File	**From File Name (Positions 11-18)**	**To File Name (Positions 19-26)**
Array or table files loaded at prerun time	If an array or table loaded at prerun time is being defined (positions 27 through 57), enter the name of the file that contains the array or table.[1]	If the array or table being defined is being written out after it is updated, enter the name of the output file or the combined array file if it is to be written to the same file that was assigned to it in the file description specifications.[1] If the array or table is not being written out, leave these positions blank.
Arrays or tables loaded at compile time	Blank.	Enter the name of the output file if the array or table is to be written out at the end of the program.[1]
Arrays loaded by input or calculation specifications	Blank.	Blank.
Record address file	Enter the name of the record address file.[1]	Enter the name of the file that contains the data records to be processed by the record address file.[1]
[1]These entries must be left-adjusted.		

Positions 27-32 (Array or Table Name)

Entry	Explanation
Array or table name	The name of the array or table used in the program.

Use positions 27 through 32 to name the array or table.

Positions 33-35 (Entries per Record)

Entry	Explanation
Blank	The array is loaded by input or calculation specifications.
1–999	Number of array or table entries in each array or table input record.

Use positions 33 through 35 to indicate the exact number of array or table entries in each array or table input record. The number must end in position 35. Every array or table input record except the last must contain the number of entries indicated in positions 33 through 35. The last record can contain fewer entries than indicated, but not more. Comments can be entered on table input records in the positions following the table entries.

If two arrays or tables are in alternating format in one file, each array or table input record must contain the corresponding entries from each array or table. The corresponding entries from the two arrays or tables are considered one entry and must be on the same record.

If positions 27 through 32 contain an array name, the following rules apply to the use of positions 11 through 18 and 33 through 35:

- For a prerun-time array, positions 11 through 18 must contain a file name and positions 33 through 35 must have an entry.
- For a compile-time array, positions 11 through 18 must be blank and positions 33 through 35 must have an entry.
- For an run-time array, positions 11 through 18 and positions 33 through 35 must be blank.

Positions 36-39 (Entries per Array or Table)

Entry	Explanation
1-9999	Maximum number of array or table entries.

Use positions 36 through 39 to indicate the maximum number of entries that can be contained in the array or table named in positions 27 through 32. This number applies to one array or table or to two arrays or tables in alternating format. The number entered must end in position 39.

Because the number of entries for two arrays or tables written in alternating format must be the same, the number in these positions also gives the number of entries in the second array or table specified in positions 46 through 51.

Positions 40-42 (Length of Entry)

Entry	Explanation
1-256	Length of each element in the array or table named in positions 27 through 32.

If L or R is specified in positions 43 or 55, the length includes the sign position.

If two arrays or tables are entered in alternating format, the specification in positions 40 through 42 applies to the array or table whose entry appears first in the record.

Position 43 (Data Format)

Entry	Explanation
Blank	The data for the array or table (1) is in zoned decimal format, or (2) is character data, or (3) is loaded through input or calculation specifications.
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.

The entry in position 43 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 array or table in the program.

Position 44 (Decimal Positions)

Entry	Explanation
Blank	Character array or table.
0-9	Number of positions to the right of the decimal in numeric array or table elements.

Use position 44 to indicate the number of decimal positions in a numeric array or table element. Position 44 must always have an entry for a numeric array or table. If the entries in an array or table have no decimal positions, enter a 0.

If two arrays or tables are entered in alternating format, the specification in this position applies to the array or table containing the entry that appears first on the record.

Position 45 (Sequence)

Entry	Explanation
Blank	No particular sequence
A	Ascending sequence
D	Descending sequence.

Use position 45 to describe the sequence (either ascending or descending) of the data in an array or table loaded at prerun time or compile time.

When an entry is made in position 45, the array or table is checked for the specified sequence at the time the array or table is loaded with data. If a prerun-time array or table is out of sequence, control passes to the RPG/400 exception/error handling routine.

Ascending sequence means that the array or table entries start with the lowest data entry (according to the collating sequence) and go to the highest. Descending sequence means that the array or table entries start with the highest data entry and go to the lowest. Items with equal values are allowed.

If two arrays or tables are entered in alternating format, the entry in position 45 applies to the array or table containing the entry that appears first on the record.

When the LOKUP operation is used to search an array or table for an entry to determine whether the entry is high or low compared with the search argument, a sequence must have been specified (A or D) for the array or table. See “LOKUP (Look Up)” on page 286 for more information.

A run-time array (loaded by input and/or calculation specifications) is not sequence checked. However, an A or D entry must be specified if a high or low LOKUP operation is processed. Sequence must be specified if the SORTA operation code is used with the array.

Positions 46-57 (Second Array Description)

The fields in positions 46 through 57 have the same significance and require the same type of entries as the fields with corresponding titles in positions 27 through 32 and 40 through 45.

Positions 46 through 57 can be used to describe a second array:

- For compile-time and prerun-time arrays, the array described in positions 46 through 57 is loaded in alternating format with the array named in positions 27 through 32.
- For run-time arrays, positions 46 through 57 can be used to describe a second run-time array that is loaded independently of the array named in positions 27 through 32.

See the discussion on positions 27 through 45 for information about correct specifications. Leave positions 46 through 57 blank for a single array or table.

Positions 58-74 (Comments)

Positions 58 through 74 can be used for comments.

Positions 75-80 (Comments)

Positions 75 through 80 can be used for comments, or left blank.

These positions are not printed contiguously with positions 6-74 on the compiler listing.

Chapter 7. Line Counter Specifications

Line counter specifications can be used for each program described PRINTER file to indicate the length of the form and the number of lines to print on a page. These entries are specified on the RPG/400 Extension and Line Counter Specifications. Line counter specifications may be used for each PRINTER file in your program. If line counter specifications are used, position 39 of the file description specifications for the PRINTER device must contain an L.

A form length and an overflow line specified by the OS/400 system override commands override any program specifications. If no override commands are used for the PRINTER file, the program specification of form length and overflow line is used. If there are no overrides and no program specifications, the form length and the overflow line specified in the device file are used.

Line Counter Specification Summary Chart

Table 20. Line Counter Specification Summary Chart

Positions	Name	Entry	Explanation
1-2	Page	Page number	Entry assigns a page number to each specification.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	L	Identification for a line counter specification.
7-14	File name	A valid file name	File name of the PRINTER file as previously specified on file description specification form.
15-17	Lines per page	2-112	The number of printing lines available is 2 through 112.
18-19	Form length	FL	Indicates that the entry in positions 15 through 17 is the form length. Positions 18 and 19 must contain FL if positions 15 through 17 contain an entry.
20-22	Overflow line number	2-112	The line number specified is the overflow line.
23-24	Overflow line	OL	Indicates that the preceding entry is the overflow line. Positions 23 and 24 must contain OL if positions 20 through 22 contain an entry.
25-74		Blank	
75-80		Optional	This space is available for comments.

Line Counter Specification Statement

Position 6 (Form Type)

An L must be entered in position 6 to identify this line as a line counter specifications statement.

Positions 7-14 (File Name)

Entry	Explanation
A valid file name	File name of the program described PRINTER file as previously defined on the file description specifications. The file name must begin in position 7.

Positions 15-17 (Lines Per Page)

Entry	Explanation
2-112	The number of printing lines available is 2 through 112.

Use positions 15 through 17 to specify the exact number of lines available on the form or page to be used. The entry must end in position 17. Leading zeros can be omitted.

Positions 18-19 (Form Length)

Entry	Explanation
FL	Form length.

Use positions 18 and 19 to indicate that the preceding entry (positions 15 through 17) is the form length. Positions 18 and 19 must contain the entry FL if positions 15 through 17 contain an entry.

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.

Positions 20-22 (Overflow Line Number)

Entry	Explanation
2-112	The line number specified is the overflow line.

Use positions 20 through 22 to specify the overflow line number. The overflow line number must be less than or equal to the form length. The entry must end in position 22. Leading zeros can be omitted. When the line that is specified as the overflow line is printed, the overflow indicator turns 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.

Positions 23-24 (Overflow Line)

Entry	Explanation
0L	Overflow line.

Use positions 23 and 24 to indicate that the preceding entry (positions 20 through 22) is the overflow line number. Positions 23 and 24 must contain 0L if positions 20 through 22 contain an entry.

Positions 25-74 (Reserved)

Positions 25 through 74 must be blank.

Positions 75-80 (Comments)

Positions 75 through 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

Chapter 8. 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/400 functions to the external description. Input specifications are also used to describe data structures and named constants. The RPG/400 Input Specification is shown in Figure 30 on page 130.

As Figure 30 on page 130 shows, for program described files, entries on input specifications are divided into the following categories:

- Record identification entries (positions 7 through 42), which describe the input record and its relationship to other records in the file.
- Field description entries (positions 43 through 70), which describe the fields in the records. Each field is described on a separate line, below its corresponding record identification entry.

For externally described files, entries on input specifications are divided into the following categories:

- Record identification entries (positions 7 through 14, and 19 and 20), which identify the record (the externally described record format) to which RPG/400 functions are to be added.
- Field description entries (positions 21 through 30, 53 through 62, and 65 through 70), which describe the RPG/400 functions to be added to the fields in the record. Field description entries are written on the lines following the corresponding record identification entries.

For data structures, entries on input specifications are divided into the following categories:

- Data structure statements (positions 7 through 12, 17 through 30, and 44 through 51), which define data structures.
- Data structure subfield specifications (positions 8, and 21 through 58), which describe the subfields of the data structures. Data structure subfield specifications are written on the lines following the data structure statements.

This chapter is organized in the following sequence:

- Input specifications summary charts
- Entries for program described files
- Entries for externally described files
- Entries for data structures
- Entries for named constants.

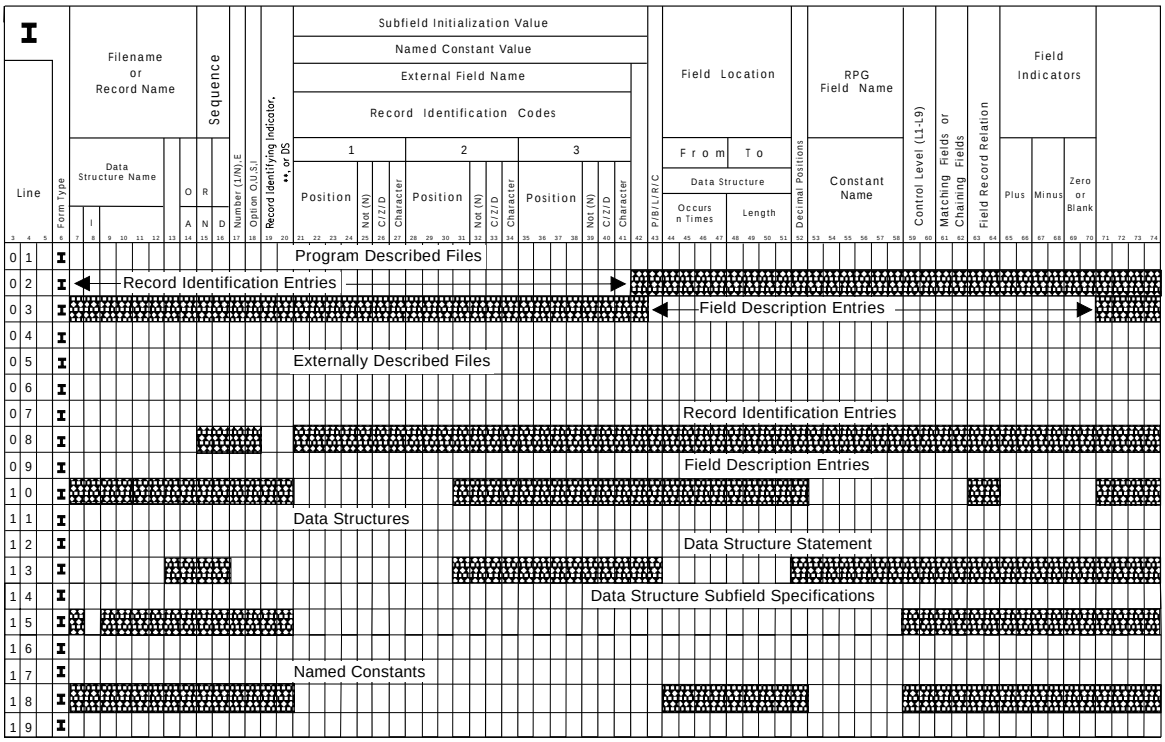


Figure 30. RPG/400 Input Specifications

Shaded areas must be blank for each specification.

Input Specifications Summary Charts

Program Described Files, Record Identification Entries

Table 21 (Page 1 of 2). Program Described Files, Record Identification Entries Summary Chart

Positions	Name	Entry	Explanation
1-2	Page	Page number	Entry assigns a page number to each specification.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	I	Identification for an input specification.
7-14	File name	A valid file name	Must be the same file name that appears on the file description specifications for the input file.
14-16	Logical relationship	AND or OR	Enter AND in positions 14 through 16 of the next specification line if more than three record identification codes are needed to identify the record. Enter OR in positions 14 and 15 if two or more record types have common fields.
15-16	Sequence	Any two alphabetic characters Any two-digit number	The program does not check for special sequence. The program checks for special sequence within the group.

Table 21 (Page 2 of 2). Program Described Files, Record Identification Entries Summary Chart

Positions	Name	Entry	Explanation
17	Number	Blank	The program does not check record types for a special sequence (positions 15 and 16 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.
18	Option	Blank	The record type must be present if sequence checking is specified.
		O	The record type is optional (it may or may not be present) if sequence checking is specified.
19-20	Record identifying indicator or **	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 only be used with a primary or secondary file.

Use Positions 21 through 41 to enter the record identification codes explained below:

Table 22 (Page 1 of 2). Program Described Files, Record Identification Codes Summary Chart

Positions	Name	Entry	Explanation
21-24, 28-31, 35-38	Position	Blank	No record identification code is present.
		1-9999	The position in the record that contains the record identification code.
25, 32, 39	Not	Blank	Record identification code must be present.
		N	Record identification code must not be present.
26, 33, 40	Code part	C	Entire character.
		Z	Zone portion of character.
		D	Digit portion of character.
27, 34, 41	Character	Any character	Enter the identifying character that is to be compared to the character in the position specified in the input record.
42-74		Blank	

Table 22 (Page 2 of 2). Program Described Files, Record Identification Codes Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
75-80		Optional	This space is available for comments.

Program Described Files, Field Description Entries

Table 23 (Page 1 of 2). Program Described Files, Field Description Entries Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
7-42		Blank	
43	Data format	Blank P B L R	The input field is in zoned decimal format or character format. The input field is in packed decimal format. The input field is in binary format. The numeric input field has a preceding (left) plus or minus sign. The numeric input field has a following (right) plus or minus sign.
44-47	From	1-9999	Specifies the beginning position of the field in the record.
48-51	To	1-9999	Specifies the end position of the field in the record.
52	Decimal positions	Blank 0-9	Character field. Number of decimal positions in numeric field.
53-58	Field name	Symbolic name	Field name, data structure name, subfield name, array name, array element, `PAGE`, `PAGE1-PAGE7`, `*IN`, or `*INxx`.
59-60	Control level	Blank `L1-L9`	This is not a control field. Control level indicators cannot be used with full procedural files. This field is a control field.
61-62	Match fields	Blank `M1-M9`	This is not a match field. This field is a match field. Match fields are valid only for primary and secondary files.

Table 23 (Page 2 of 2). Program Described Files, Field Description Entries Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
63-64	Field record relation	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.
65-70	Field indicators	Blank	No indicator specified.
		01-99	Field indicator.
		H1-H9	Halt indicator.
		U1-U8	External indicators.
		RT	Return indicator.
71-74		Blank	
75-80		Optional	This space is available for comments.

Externally Described Files, Record Identification Entries

Table 24 (Page 1 of 2). Externally Described Files, Record Identification Entries Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
1-2	Page	Page number	Entry assigns a page number to each specification form.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	I	Identification for an input specification.
7-14	Record name	Record format name	The RPG/400 name of the record format. A file name cannot be used.
15-18	Sequence	Blank	These positions must be blank.

Table 24 (Page 2 of 2). Externally Described Files, Record Identification Entries Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
19-20	Record identifying indicators	Blank	No record identifying indicator.
		01-99	General indicator.
		L1-L9, LR	Control level indicator used for record identifying indicator.
		H1-H9	Halt indicator.
		U1-U8	External indicator.
		RT	Return indicator.
21-41	Record identification code	Blank	Record format names are used to determine the record types used in the program.
42-74		Blank	
75-80		Optional	This space is available for comments.

Externally Described Files, Field Description Entries

Table 25 (Page 1 of 2). Externally Described Files, Field Description Entries Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
7-20		Blank	
21-30	External field name	Field name	If a field within a record in an externally described field is to be renamed, enter the external name of the field in these positions.
31-52		Blank	
53-58	RPG/400 field name	Field name	The name of the field as it appears in the external record description (if 6 characters or less) or the field name that replaces the externally defined field name in positions 21 through 30.
59-60	Control level	Blank	Field is not a control field.
		L1-L9	This field is a control field.
61-62	Match fields	Blank	Field is not a match field.
		M1-M9	The field is a match field.
63-64		Blank	
65-70	Field indicators	Blank	No indicator specified.
		01-99	General indicators.
		H1-H9	Halt indicators.
		U1-U8	External indicators.
		RT	Return indicator.

Table 25 (Page 2 of 2). Externally Described Files, Field Description Entries Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
71-74		Blank	
75-80		Optional	This space is available for comments.

Data Structure Statement Specifications

Table 26 (Page 1 of 2). Data Structure Statement Specifications Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
1-2	Page	Page number	Entry assigns a page number to each specification form.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	I	Identification for an input specification.
7-12	Data structure name	Blank Valid data structure name	The name of the data structure being defined may be omitted. Enter the name of the data structure being defined.
13-16		Blank	
17	External Description	Blank E	Subfield definitions for this data structure follow this specification. Subfield definitions are defined externally. The entry in positions 7 through 12 references the external definition.
18	Option	Blank I S U	Not a program status or data area data structure, or no data structure initialization. Data structure initialization. The entire data structure is initialized, characters to blank, numerics to zero, at the beginning of program initialization. A program status data structure. Only one data structure may be designated as the program status data structure. A data area data structure. RPG/400 retrieves the external data area (named in positions 7 through 12) at initialization and rewrites it at the end of the program. If you put blanks in positions 7 through 12, the local data area is used.
19-20	Record identifying indicator	DS	Indicates a data structure.
21-30	External file name	External name of data structure description	If an externally described data structure is to be renamed, enter the external file name for the subfield definitions and enter the name to be used in the program in positions 7 through 12.
31-43		Blank	

Table 26 (Page 2 of 2). Data Structure Statement Specifications Summary Chart

Positions	Name	Entry	Explanation
44-47	Occurrences	Blank 1-9999	This is not a multiple occurrence data structure. Entry indicates the number of occurrences of this multiple occurrence data structure.
48-51	Data structure length	Blank 1-9999	Length of the data structure is either the length specified on the input field specifications if the data structure is an input field or the highest To position specified for a subfield within the data structure if the data structure is not an input field. Length of the data structure.
52-74		Blank	
75-80		Optional	This space is available for comments.

Data Structure Subfield Specifications

Table 27. Data Structure Subfield Specifications Summary Chart

Positions	Name	Entry	Explanation
7		Blank	
8	Initialization option	Blank I	No subfield initialization. Subfield is initialized to the value specified in positions 21 to 42.
9-20		Blank	
21-30	External field name	Subfield name	If a subfield in an externally described data structure is to be renamed, enter the external name of the subfield in these positions.
21-42	Initialization value	Valid initial field	If a subfield is to be initialized, specify a literal value or named constant.
31-42		Blank	If a subfield in an externally described data structure is to be renamed to the entry specified in positions 21 to 30, positions 31 to 42 are left blank.
43	Internal data format	Blank P B	Subfield is in zoned decimal format or character format. Subfield is in packed decimal format. Subfield is in binary format.
44-47	From	1-9999	Specifies subfield's beginning position in data structure.
48-51	To	1-9999	Specifies subfield's end position in data structure.
44-51	Keywords	Valid keyword	Special keywords define the location of subfields in the program status data structure or the file information data structure. Keywords for the program status data structure are `*STATUS`, `*PROGRAM`, `*PARMS`, and `*ROUTINE`. Keywords for the file information data structure are `*FILE`, `*RECORD`, `*OPCODE`, `*STATUS`, and `*ROUTINE`.
52	Decimal positions	Blank 0-9	Character subfield. Number of decimal positions in a numeric subfield.
53-58	Subfield name	Valid subfield name	The subfield name or the external name of the subfield or the subfield name that replaces the external subfield name specified in positions 21 through 30.
59-74		Blank	
75-80		Optional	This space is available for comments.

Named Constant

Table 28. Named Constant Specifications Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
1-5		Blank	
6	Form type	I	Identification for an input specification.
7-20		Blank	
21-42	Constant	Constant	Any valid RPG/400 literal including transparent literals. The character constants may be continued on the next line if needed by coding a hyphen instead of a single quote as the last character.
43	Data type	C Blank	Indicates type of name is constant. Continuation line
44-52		Blank	
53-58	Constant name	Name	Name of constant. The normal rules for RPG/400 names apply. Reserved words cannot be used.
59-74		Blank	

Named Constant Continuation

Table 29. Named Constant Continuation Specifications Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
1-5		Blank	
6	Form type	I	Identification for an input specification.
7-20		Blank	
21-42	Constant	Constant	Any valid RPG/400 literal including transparent literals.
43-74		Blank	

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 42) for a program described file describe the input record and its relationship to other records in the file.

Positions 7-14 (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 14-16 (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 144 and “OR Relationship” on page 144.

Positions 15-16 (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 17) and option (position 18) 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/400 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 15 and 16 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 17 and 18.

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/400 exception/error handling routine.

Sequence numbers ensure only that all records of each record type precede the records of senior 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 61 and 62 (matching fields) to indicate that data in fields of a record should be checked for a special sequence.

Position 17 (Number)

Entry	Explanation
Blank	The program does not check record types for a special sequence (positions 15 and 16 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 15 and 16. If an alphabetic entry is made in positions 15 and 16, this entry must be blank.

Position 18 (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 15 and 16 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 15 and 16 or O entry in position 18).

Positions 19-20 (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 21 through 41).

Indicators

Positions 19 and 20 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 or as a data structure name on Input Specifications or as a Result Field on Calculation Specifications.

Positions 15 and 16 must contain an alphabetic entry. The lookahead fields are defined in positions 53 through 58 of the lines following the line containing ** in positions 19 and 20. Positions 59 through 74 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 16 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 21-41 (Record Identification Codes)

Entries in positions 21 through 41 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 19 and 20) and a sequence entry (positions 15 and 16) must be made.

Three sets of entries can be made in positions 21 through 41: 21 through 27, 28 through 34, and 35 through 41. 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	21-27	28-34	35-41
Position	21-24	28-31	35-38
Not	25	32	39
Code Part	26	33	40
Character	27	34	41

Entries in these sets need not be in sequence. For example, an entry can be made in positions 28 through 34 without requiring an entry in positions 21 through 27. 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/400 exception/error handling routine.

Positions 21-24, 28-31, and 35-38 (Position)

Entry	Explanation
Blank	No record identification code is present.
1-9999	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 25, 32, and 39 (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 26, 33, and 40 (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 (position 27) to be compared with the zone portion of the character in the record position specified in the position entry (positions 21 through 24). 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 27, 34, and 41 (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 14 through 16 for each additional line used. Positions 7 through 13, 17 through 20, and 42 through 74 of each line with AND in positions 14 through 16 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 14 and 15. Positions 7 through 13, 16 through 18, and 42 through 74 must be blank. A record identifying indicator can be entered in positions 19 and 20. If the indicator entry is made and the record identification codes on the OR line are satisfied, the indicator specified in positions 19 and 20 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.

Position 42 (Reserved)

Position 42 must be blank.

Field Description Entries

The field description entries (positions 43 through 74) must follow the record identification entries (positions 7 through 42) for each file.

Position 43 (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.
R	The number input field has a following (right) plus or minus sign.

The entry in position 43 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 *RPG/400 User's Guide* for information on internal field formats.

Positions 44-51 (Field Location)

Entry	Explanation
Two 1- to 4-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 44 through 47 specify the location of the field's beginning position; positions 48 through 51 specify the location of the field's end position. To define a single-position field, enter the same number in positions 44 through 47 and in positions 48 through 51. 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)
256	Character (256 characters)
31	Numeric with leading or trailing sign (30 digits)
9999	Data structure.

For arrays, enter the beginning position of the array in positions 44 through 47 and the ending position in positions 48 through 51. 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.

Position 52 (Decimal Positions)

Entry	Explanation
Blank	Character field
0-9	Number of decimal positions in numeric field.

This entry, used with the data format entry in position 43, describes the format of the field. This entry indicates whether the field described on this line is a character field or a numeric field. 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. For an array or an array element, this entry must be the same as the entry made on the extension specifications (position 44 or 56) for the array. (If this entry is blank for a numeric array or array element, the decimal positions specified in the extension specification are used.)

Positions 53-58 (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/400 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 53 through 58. If an array name is entered in positions 53 through 58, control

level (positions 59 and 60), matching fields (positions 61 and 62), and field indicators (positions 65 through 70) must be blank.

To refer to an element of an array, specify the array name, followed by a comma, followed by an index. 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 59-60 (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 59 and 60 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 61-62 (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 61 and 62 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.

Entries in position 26 (alternate collating sequence) and position 43 (file translation) of 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/400 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 63-64 (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/400 program only when the indicator coded in positions 63 and 64 is on or when positions 63 and 64 are blank. When positions 63 and 64 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 59 and 60) and matching fields (positions 61 and 62).

Positions 65-70 (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 65 through 70 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 65 and 66 (plus) and positions 67 and 68 (minus) are valid for numeric fields only. Positions 69 and 70 can be used to test a numeric field for zeros or a character 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.

Positions 71-74 (Reserved)

Positions 71 through 74 must be blank.

Positions 75-80 (Comments)

Positions 75 through 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

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 14 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 7-14 (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/400 name specified by the RENAME option on the file description specifications continuation line if the external record format was renamed. A record format name can appear only once in positions 7 through 14 of the input specifications for a program.

Positions 15-18 (Reserved)

Positions 15 through 18 must be blank.

Positions 19-20 (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 138earlier in this chapter, except for look-ahead specifications, which are not allowed for an externally described file.

Positions 21-41 (Record Identification Code)

Positions 21 through 41 must be blank. Record format names are used to determine the record types used in an externally described file.

Positions 42-74 (Reserved)

Positions 42-74 must be blank.

Positions 75-80 (Comments)

Positions 75-80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

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/400 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 external name is longer than 6 characters or because the name is the same as a field name specified in the program and two different names are required.

Positions 31-52 (Reserved)

Positions 31 through 52 must be blank.

Positions 53-58 (Field Name)

The field name entry is made only when it is required for the RPG/400 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 6 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 59-60 (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 61-62 (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 61-62 (Matching Fields)” on page 146 for more information on match fields.

Positions 63-64 (Reserved)

Positions 63 and 64 must be blank.

Positions 65-70 (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 65-70 (Field Indicators)” on page 147 for more information.

Positions 71-74 (Reserved)

Positions 71 through 74 must be blank.

Positions 75-80 (Comments)

Positions 75 through 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

Data Structure Specifications

A data structure can be used to:

- Allow the division of a field into subfields without using the MOVE or MOVEL operations.
- Operate on a subfield and change the contents of a subfield.
- Redefine the same internal area more than once using different data formats.

Data structures are defined on the input specifications the same way records are defined. The record specification line contains the data structure statement (DS in positions 19 and 20) and the data structure name (optional). The field specification lines contain the subfield specifications for the data structure.

Data structure specifications must follow the input specifications for records. All entries describing a data structure and its subfields must appear together.

Data Structure Specification Entries

Position 6 (Form Type)

Position 6 must contain an I for input specifications.

Positions 7-12 (Data Structure Name)

Positions 7 through 12 can contain the name of the data structure being defined. The data structure name is optional, and is limited to 6 characters. A data structure name must follow the rules for using symbolic names. A data structure name can be specified anywhere a character field can be specified. If the data structure is externally described and positions 21-30 are blank, this entry must contain the name of an externally described file.

Positions 13-16 (Reserved)

Positions 13 through 16 must be blank.

Position 17 (External Description)

Entry	Explanation
Blank	Subfield definitions for this data structure follow this specification.
E	Subfield definitions are described externally. Positions 7 through 12 must contain the name of an externally described file if positions 21 through 30 are blank. The file name must be limited to 6 characters.

Position 18 (Option)

Entry	Explanation
Blank	This data structure is not a program status or data area data structure, and this data structure is not globally initialized.
I	Data structure initialization. All subfields in the data structure are initialized; characters to blank, numerics to zero, in the order in which they are defined, during program initialization.
S	This data structure is the program status data structure. Only one data structure can be specified as the program status data structure.
U	This is a data area data structure. The external data area (named in positions 7 through 12) is retrieved when the program starts and rewritten when the program ends. If you put blanks in positions 7 through 12, the local data area is used. **Note:** The data area specified by the data structure is locked for the duration of the program.

Positions 19-20 (Record Identifying Indicator)

Positions 19 and 20 must contain DS to indicate a data structure.

Positions 21-30 (External File Name)

Entry	Explanation
Blank	The data structure subfields are defined in the program.
File name	This is the name of the file whose first record format contains the field descriptions used as the subfield descriptions for this data structure.

Positions 31-43 (Reserved)

Positions 31 through 43 must be blank.

Positions 44-47 (Data Structure Occurrences)

Entry	Explanation
Blank	This is not a multiple-occurrence data structure.
1-9999	The number (right-adjusted) indicating the number of occurrences of a multiple-occurrence data structure.

These positions must be blank if the data structure is the program status data structure (indicated by an S in position 18), a file information data structure (INFDS), or a data area data structure.

Positions 48-51 (Length)

Entry	Explanation
Blank	Length of the data structure is either the length specified on the input field specifications if the data structure is an input field or the highest To position specified for a subfield within the data structure if the data structure is not an input field.
1-9999	Length of the data structure.

The length of the data structure can be specified in positions 48 through 51. This entry is optional but, if used, must be right-adjusted. If this entry is not made, the length of the data structure is one of the following:

- The length specified on the input field specifications if the data structure name is an input field.
- The highest To position specified for a subfield within the data structure if the data structure name is not an input field.

Positions 52-74 (Reserved)

Positions 52 through 74 must be blank.

Positions 75-80 (Comments)

Positions 75 through 80 may be used for comments or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

Data Structure Subfield Specifications

Specifications for subfields, if used, must follow the data structure specification statement to which they apply.

Position 7 (Reserved)

Position 7 must be blank.

Position 8 (Initialization Option)

Entry	Explanation
Blank	No subfield initialization.
I	Subfield is initialized to value specified in positions 21 to 42.

Positions 9-20 (Reserved)

Positions 9 through 20 must be blank.

Positions 21-30 (External Field Name)

To rename a subfield in an externally described data structure, specify the external name in positions 21 through 30, and specify the name to be used in the program in positions 53 through 58. The remaining positions must be blank.

Positions 21-42 (Initialization Value)

If a subfield is to be initialized, specify a literal value or a named constant in these positions. If no value is specified and position 8 contains `I`, the subfield is initialized to zero or blanks, depending on the field type. The value may be continued on the next line. See “Named Constant Continuation Specifications” on page 155 for more information on continuation.

See the *RPG/400 User's Guide* for more information on how to specify the initialization value.

Positions 31-42 (Reserved)

Positions 31 through 42 must be blank, if an external field name is specified in positions 21 to 30.

Position 43 (Internal Data Format)

Entry	Explanation
Blank	Subfield is in zoned decimal format or is character data if position 52 is blank.
P	Subfield is in packed decimal format.
B	Subfield is in binary format.

Note: Unlike the external data format field, the entry determines the internal format of the data.

Positions 44-51 (Field Location)

Entry	Explanation
Two 1- to 4-digit numbers	Beginning of a subfield (from) and end of a subfield (to).
Keywords	For the program status data structure or for a file information data structure, special keywords define the location of the subfields in the data structures. Keywords for the program status data structure are *STATUS, *PROGRAM, *PARMS, and *ROUTINE. Keywords for the file information data structure are *FILE, *RECORD, *OPCODE, *STATUS, and *ROUTINE.

Positions 44-47 are the From position. Positions 48-51 are the To position. Both From and To must be right-justified, and leading zeroes may be omitted.

Position 52 (Decimal Positions)

Entry	Explanation
Blank	Character subfield
0-9	Number of decimal positions in a numeric subfield.

Position 52, along with position 43, determines the format of the subfield. An entry must be made in position 52 for a numeric subfield.

Positions 53-58 (Field Name)

In positions 53 through 58, enter the name of the subfield that is being defined. The name can be an array name, but cannot be an array element name.

Positions 59-74 (Reserved)

Positions 59 through 74 must be blank.

Positions 75-80 (Comments)

Positions 75 through 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

Named Constant Specifications

Positions 7-20 (Reserved)

Positions 7 through 20 must be blank.

Positions 21-42 (Constant)

In positions 21-42 enter the constant or edit word being declared. The constant may be continued on subsequent lines by coding a hyphen as the last character. For character named constants the hyphen replaces the ending quote. A continued numeric constant must result in a valid decimal number with at most 30 digits, a maximum of 9 being to the right of the decimal point. Named constants can be declared anywhere in the input specifications.

Position 43 (Data Type)

Entry	Explanation
C	Type of name is constant
Blank	Constant continuation line

Positions 44-52 (Reserved)

Positions 44-52 must be blank.

Positions 53-58 (Constant Name)

Positions 53-58 contain the name of the constant. The normal rules for RPG/400 names apply.

Positions 59-74 (Reserved)

Positions 59-74 must be blank.

Named Constant Continuation Specifications

Positions 7-20 (Reserved)

Positions 7 through 20 must be blank.

Positions 21-42 (Constant)

In positions 21-42 enter the constant or edit word being continued. A character or transparent literal constant may be continued over as many lines as desired so long as the total length of the constant does not exceed 256 characters. A continued numeric constant must result in a valid decimal number with at most 30 digits, a maximum of 9 being to the right of the decimal point.

Positions 43-74 (Reserved)

Positions 43-74 must be blank.

The following are examples of named constants:

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
I..............Namedconstant+++++++++C.........Fldnme.............
I*
I*The following is an example of a character constant:
I              'MICKEY'              C         MOUSE
I*
I*The following is an example of a continued character constant:
I*
I              'ABCDEF-              C         CHAR
I              'GHIJK'
I*
I*The following is an example of a numeric constant:
I*
I              123456789             C         INTEGER
I*
I*
I*The following is an example of a continued numeric constant:
I*
I              123456-               C         NUM
I              789
I*
I*The following is an example of a hexadecimal constant:
I*
I              X'010203'             C         HEX1
I*
I*The following is an example of a continued hexadecimal constant:
I*
I              X'010-                C         HEX2
I              '202'
I*
I*The following is an example of a continued transparent
I*constant.  The Shift Out (SO) and Shift In (SI) characters
I*are represented by o and i.  The value of the combined literal
I*is 'oK1K2K3K4K5i' if the transparent literal option is specified
I*(1 in position 57 of the control specification).
I*
I              'oK1K2K3i-            C         TRANS
I              'oK4K5i'
```

Figure 31. Named Constant Examples

Chapter 9. Calculation Specifications

Calculation specifications indicate the operations to be done on the data in a program. Two general rules govern the writing of calculation entries:

- Each operation is specified on one line, except when there are AND/OR lines in the calculation.
- 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.

Each calculation specifications statement is divided into three parts that specify the following:

- When calculations are to be done: The conditioning indicators specified in positions 7 through 17 determine when and under what conditions the calculations are to be done.
- What kind of calculations are to be done: The entries specified in positions 18 through 53 determine the kind of calculations to be done, specify the data (such as fields or files) upon which the operation is to be done, and specify the field that is to contain the results of the calculation.
- What tests are to be made on the results of the operation: Indicators specified in positions 54 through 59 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 11, "Operation Codes" on page 185.

The calculation specifications are entered on the RPG/400 Calculation Specifications. See Chapter 11, "Operation Codes" on page 185 for details on how these positions must be specified for individual calculation operations.

The calculation specification can also be used to enter SQL statements into an RPG/400 program. See *RPG/400 User's Guide* and *Programming: Structured Query Language Reference* for more information.

Calculation Specification Summary Chart

Table 30 (Page 1 of 3). Calculation Specification Summary Chart

Positions	Name	Entry	Explanation
1-2	Page	Page number	Entry assigns a page number to each specifications form.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	C	Identification for a calculation specification.

Table 30 (Page 2 of 3). Calculation Specification Summary Chart

Positions	Name	Entry	Explanation
7-8	Control level	Blank	The calculation operation is done at detail calculation time of each program cycle if the calculation is part of a subroutine, or if the calculation is a declarative statement.
		L0	The calculation operation is done at total calculation time of each program cycle.
		L1-L9	The calculation operation is done at total calculation time if the indicator is on (because a control break occurs or because the indicator is set on).
		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/400 subroutine. Optional.
		AN, OR	Conditioning indicators on more than one line.
9-17	Conditioning indicators	Blank	The operation is done if the condition specified in positions 7 and 8 is satisfied. An N in positions 9, 12, and 15 is used to check if the indicator is not on (SETOF or containing 0) to decide if calculations will occur. The following indicators are valid in positions 9 through 17:
		01-99	• General indicator
		KA-KN, KP-KY	• Function key indicator
		L1-L9	• Control level indicator
		LR	• Last record indicator
		MR	• Matching record indicator
		H1-H9	• Halt indicator
		RT	• Return indicator
		U1-U8	• External indicator
		OA-OG, OV	• Overflow indicator. If the conditions specified in positions 9 through 17 are satisfied, the operation is done.
18-27	Factor 1	Symbolic name or literal	Entry specifies a symbolic name or actual data on which an operation is to be done. Valid entries depend on the operation code.
28-32	Operation	Operation code	Entry specifies the operation to be done.

Table 30 (Page 3 of 3). Calculation Specification Summary Chart

Positions	Name	Entry	Explanation
33-42	Factor 2	Symbolic name or literal	Entry specifies a symbolic name or actual data on which an operation is to be done. Valid entries depend on the operation code.
43-48	Result field	Field name	The result field names the field that contains the result of the calculation operation specified in positions 28 through 32. Valid entries depend on the operation code.
49-51	Field length	Blank 1-30 1-256	The result field is defined elsewhere. Numeric field length. Character field length.
52	Decimal positions	Blank 0-9	The result field is character data or has been defined elsewhere in the program. Number of decimal positions in a numeric result field.
53	Operation extender	Blank H N P	No operation extension supplied. Half-adjust is done. Record is read but not locked. Pad the result field with blanks.
54-59	Resulting indicators	Blank 01-99 KA-KN, KP-KY H1-H9 L1-L9 LR OA-OG, OV U1-U8 RT	No resulting indicator. General indicators. Function key indicators. Halt indicators. Control level indicators. Last record indicator. Overflow indicators. External indicators. Return indicator. **Note:** The resulting indicator positions (54 and 55, 56 and 57, and 58 and 59) have different uses, depending on the operation code specified.
60-74	Comments	Comments	These positions can be used for comments to document the purpose of the calculation.
75-80		Optional	This space is available for comments.

Calculation Specification Statement

Position 6 (Form Type)

A C must appear in position 6 to identify this line as a calculation specifications 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 17 allow it; or the calculation is part of a subroutine.
L0	The calculation operation is done at total calculation time for each program cycle.
L1-L9	The calculation operation is done when the appropriate control break occurs at total calculation time, or when the indicator is set on.
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/400 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 to be done during total time and is not dependent on the occurrence of a control break.

If indicators L1 through L9 are specified in positions 7 and 8, the calculation is processed at total 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, is a statement within a subroutine, or is a declarative statement.

The following operations can be specified within total calculations with positions 7 and 8 blank: PLIST, PARM, KLIST, KFLD, TAG, DEFN, and ELSE. (Conditioning indicators in positions 9 through 17 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 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/400 programmer determines which files are to be checked for end-of-file by entering an E in position 17 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, the control level indicators L1 through L9 are also set on. If the indicators L1 through L9 have not been defined by an entry in positions 59 and 60 of the input specifications or by *INxx when used in a result field, the indicators are set on when LR is on, but they cannot be used in other specifications.

Subroutine Identifier

An SR entry in positions 7 and 8 indicates that the specification is part of a subroutine. The SR entry is not required. 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 17) for a calculation. Indicators in positions 9 through 17 contained in one line are always in an AND relationship. Indicators contained on more than one line can be a combination of AND and OR relationships. A maximum of seven AND/OR lines can be specified in one group.

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 9-17 (Indicators)

Entry	Explanation
Blank	The operation is processed on every record if the condition specified in positions 7 and 8 is satisfied.
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, 13 and 14, and 16 and 17 contain indicators that are tested to determine if a particular calculation is to be processed. A blank in positions 9, 12, and 15 designates that the indicator must be on for a calculation to be done. An N in positions 9, 12, and 15 designates that the associated indicator must be off for a calculation to be done.

One to three indicators can be entered in positions 9 through 17 on one line. Indicators on the same line are in an AND relationship. The AND relationship means that all three indicator conditions must be satisfied before the calculation can take place. If fewer than three indicators are specified, entries need not be made in sequence; that is, an indicator can be specified in positions 16 and 17, and positions 10

through 15 can be blank. When more than three indicators are needed to condition a calculation, AND/OR must be used. Seven is the maximum number of AND/OR lines that can be specified on calculation specifications. Positions 9 through 17 can contain a combination of the type of indicators discussed in the preceding text.

Positions 18-27 (Factor 1)

Factor 1 names a field or gives actual data (literals) or RPG/400 special words (*NAMVAR DEFN) on which an operation is to be done. The entry must begin in position 18. The entries that are valid for factor 1 depend on the operation code specified in positions 28 through 32. For the specific entries for factor 1 for a particular operation code, see Chapter 11, “Operation Codes” on page 185. With some operation codes, a colon can be used to separate parts of the factor, but must be preceded by and followed by a valid entry.

Positions 28-32 (Operation)

Positions 28 through 32 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 28. For further information on the operation codes, see Chapter 11, “Operation Codes” on page 185.

The program processes the operations in the order specified on the calculation specifications form.

Positions 33-42 (Factor 2)

Factor 2 names a field or gives the actual data (literals) on which a calculation is to be done. For the file operation codes, factor 2 names a file or record format to be used. The entry must begin in position 33. The entries that are valid for factor 2 depend on the operation code specified in positions 28 through 32. With some operation codes, a colon can be used to separate parts of the factor, but must be preceded by and followed by a valid entry. For the specific entries for factor 2 for a particular operation code, see Chapter 11, “Operation Codes” on page 185.

Positions 43-48 (Result Field)

The result field names the field that contains the result of the calculation operation specified in positions 28 through 32. Array elements are treated as fields. A look-ahead field, user date special word a literal, or a named constant cannot appear as the result field of a calculation. See Chapter 11, “Operation Codes” on page 185 for the result field rules for individual operation codes.

Positions 49-51 (Field Length)

Entry	Explanation
1-30	Numeric field length.
1-256	Character field length.
Blank	The result field is defined elsewhere.

Positions 49 through 51 specify the length of the result field. This entry is optional, but can be used to define a field not defined elsewhere in the program. These definitions of the field entries are allowed if the result field contains a field name.

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 length of an arithmetic result to the left of the decimal point exceeds the specified length, the excess leftmost digits and any excess decimal positions are dropped. No diagnostic message occurs if the result is truncated.

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.

If half-adjustment is specified in position 53 of the calculation specifications, the entries for field length (positions 49 through 51) and decimal positions (position 52) refer to the length of the result field after half-adjustment.

Position 52 (Decimal Positions)

Entry	Explanation
Blank	The result field is character data or has been defined elsewhere in the program.
0-9	Number of decimal positions in a numeric result field.

Position 52 indicates 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. This position can be left blank if the result field is numeric but was described by input or calculation specifications or in an external description. In this case, field length (positions 49 through 51) must also be left blank. The number of decimal positions specified cannot exceed the length of the field.

Position 53 (Operation Extender)

Entry	Explanation
Blank	No operation extension supplied.
H	Half adjust.
N	Record is read but not locked.
P	Pad the result field with blanks.

The operation extenders are single-character entries that provide additional attributes to the operations that they accompany. Operation extenders are specified in position 53 of calculation specifications.

An H indicates whether the contents of the result field are to be half adjusted (rounded). 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 can be specified only if the number of decimal positions in the generated result is greater than the number of decimal positions in the result field. 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, REDPE, 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, for CAT, SUBST, MOVEA, MOVEL, or XLATE, the result field is padded on the right after executing the instruction if the result field is longer than the result of the operation. Padding is done from the left for MOVE.

Positions 54-59 (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 an end-of-file, error, or record-not-found condition. The resulting indicator positions have different uses, depending on the operation code specified. See the individual operation codes in Chapter 11, "Operation Codes" on page 185 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 last calculation 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 unless the halt indicator is set off before the indicator is tested.

Positions 60-74 (Comments)

Positions 60 through 74 of each calculation specification line can be used for comments to document the purpose of that calculation.

Positions 75-80 (Comments)

Positions 75 through 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiled listing.

Chapter 10. 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 37), and field description and control (positions 23 through 70). These specifications are entered on the RPG/400 Output Specifications.

This chapter is organized in the following sequence:

- Output specifications summary charts
- Entries for program described files
- Entries for externally described files.

Output Specifications Summary Charts

Program Described Files, Record Identification and Control Entries (Record Line)

Table 31 (Page 1 of 3). Program Described Files, Record Identification and Control Entries (Record Line)

Positions	Name	Entry	Explanation
1-2	Page	Page number	Entry assigns a page number to each specification form.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	O	Identification for an output specification.
7-14	File name	Valid file name	Same file name that appears on the file description specifications for the output file.
14-16	Logical relationship	AND or OR	AND/OR indicates a relationship between lines of output indicators. Not valid for output fields.
15	Type	H or D T E	Detail records. Total records. Exception records.
16-18	Record addition deletion field	ADD DEL	Add a record to the file or subfile. Delete a record from a file.
16	Fetch overflow specifier Release	Blank F R	If this position is blank for a printer file, overflow is not fetched. Fetch overflow specified for printer files. Release a device (work station) after output.

Table 31 (Page 2 of 3). Program Described Files, Record Identification and Control Entries (Record Line)			
Positions	**Name**	**Entry**	**Explanation**
17	Space before	0 or Blank	Spaces zero lines before the line is printed.
		1	Spaces one line before the line is printed.
		2	Spaces two lines before the line is printed.
		3	Spaces three lines before the line is printed.
18	Space after	0 or Blank	Spaces zero lines after the line is printed.
		1	Spaces one line after the line is printed.
		2	Spaces two lines after the line is printed.
		3	Spaces three lines after the line is printed.
19-20	Skip before	0 or Blank	No skipping occurs.
		01-99	Skip to specified line number before printing line.
		A0-A9	Specifies numbers between 100 and 109. Skip to specified line number before printing line.
		B0-B2	Specifies numbers between 110 and 112. Skip to specified line number before printing line.
21-22	Skip after	0 or Blank	No skipping occurs.
		01-99	Skip to specified line number after printing the line.
		A0-A9	Specifies numbers between 100 and 109. Skip to specified line number after printing line.
		B0-B2	Specifies numbers between 110 and 112. Skip to specified line number after printing line.

Table 31 (Page 3 of 3). Program Described Files, Record Identification and Control Entries (Record Line)			
Positions	**Name**	**Entry**	**Explanation**
23-31	Output indicators	Blank	The line or field is written every time the type of record is checked for output.
			The line or field is written if the indicators in these positions are satisfied. The following indicators are valid in these positions:
		1-99	A general indicator used as a resulting indicator, field indicator, or record identifying indicator
		KA-KN, KP-KY	Function key indicator
		L1-L9	Control level indicator
		H1-H9	Halt indicator
		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).
32-37	EXCPT name	Record group name	A name placed in these positions specifies exception records to be written. Any number of exception output records can use the same EXCPT name, and they do not have to be consecutive.
38-74		Blank	
75-80			This space is available for comments.

Program Described Files, Field Description and Control Entries (Field Line)

Table 32 (Page 1 of 2). Program Described Files, Field Description and Control Entries (Field Line)			
Positions	**Name**	**Entry**	**Explanation**
7-22		Blank	
23-31	Field output indicators	See output indicators	Used to indicate whether fields within records are printed.

Table 32 (Page 2 of 2). Program Described Files, Field Description and Control Entries (Field Line)

Positions	Name	Entry	Explanation
32-37	Field name	Valid field name PAGE, PAGE1-PAGE7 UDATE, *DATE, UDAY, *DAY, UMONTH, *MONTH, UYEAR, *YEAR *PLACE	Any field name previously defined in the program. Automatic page numbering. Field output indicators control the resetting of the PAGE field to zero. UDATE or *DATE places the full date in the output record; UDAY or *DAY, only the day; UMONTH or *MONTH, only the month; UYEAR or *YEAR, only the year. Allows programmer to repeat data in an output record.
38	Edit codes	Blank 1-9, A-D, J-Q, X, Y, Z	No edit code is used. Numeric fields are zero-suppressed and punctuated according to a predefined pattern without the use of edit words. Edit codes 5 through 9 are defined externally by an OS/400 function.
39	Blank after	Blank B	The field is not reset. The field specified in positions 32 through 37 is reset to blank or zero after the field is written to the output record.
40-43	End position	Blanks, +nnn, –nnn, nnnn K1-K8	Defines the end position of a field or constant in the output record. The end position must not exceed the record length for the file. Length of the format name for a WORKSTN file.
44	Data format	Blank P B L R	The field is to be written in zoned decimal or character format; numeric data is edited or a constant is specified on this line. The field is to be written in packed decimal format. The field is to be written in binary format. The numeric output field is to have a preceding (left) plus or minus sign. The numeric output field is to have a following (right) plus or minus sign.
45-70	Constant or edit word	Constant or edit word Format name	Enter a constant or an edit word. Name of the external format to be used by the WORKSTN file.
70-74		Blank	
75-80		Optional	This space is available for comments.

Externally Described Files, Record Identification and Control Entries

Table 33 (Page 1 of 2). Externally Described Files, Record Identification and Control Entries

Positions	Name	Entry	Explanation
1-2	Page	Page number	Entry assigns a page number to each specification form.
3-5	Line	Line number	Entry numbers each specification line.
6	Form type	O	Identification for an output specification.
7-14	Record name	Valid record format name	Enter the record format name for externally described files.
14-16	Logical relationship	AND or OR	AND/OR specifies a relationship between two lines of output indicators.
15	Type	H or D T E	Detail records. Total records. Exception records.
16	Release	R	Release a device after output.
16-18	Record addition/deletion	ADD DEL	Add a record to the file or subfile. Delete an existing record from the file.
16-22	Space/Skip, Fetch overflow	Blank	Not used for externally-described files.
23-31	Output indicators	Blank 1-99 KA-KN, KP-KY L1-L9 H1-H9 U1-U8 MR LR RT 1P	The line or field is written every time the type of record is checked for output. The line or field is written if the indicators in these positions are satisfied. The following indicators are valid in these positions: A general indicator used as a resulting indicator, field indicator, or record identifying indicator Function key indicator Control level indicator Halt indicator External indicator set before running the program or set as a result of a calculation operation Matching record indicator Last record indicator Return indicator First-page indicator (valid only on heading or detail lines).

Table 33 (Page 2 of 2). Externally Described Files, Record Identification and Control Entries			
Positions	**Name**	**Entry**	**Explanation**
32-37	EXCPT name	Record group name	A name placed in these positions specifies exception records to be written. Any number of exception output records can use the same EXCPT name, and they do not have to be consecutive.
38-74		Blank	
75-80		Optional	This space is available for comments.

Externally Described Files, Field Description and Control Entries

Table 34. Externally Described Files, Field Description and Control Entries			
Positions	**Name**	**Entry**	**Explanation**
7-22		Blank	
23-31	Field output indicators	See output indicators	Used to indicate whether fields within records are written.
32-37	Field name	Valid field name *ALL	Either the externally described field name or the new name as renamed using positions 53-58 of the input specifications. All fields are written.
38		Blank	
39	Blank after	Blank B	The field is not reset. The field specified in positions 32 through 37 is reset to blank or zero after the field is written to the output record.
40-74		Blank	
75-80		Optional	This space is available for comments.

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 37 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 7-14 (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 14-16 (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 14 through 16 specify AND/OR lines for output operations. To specify this relationship, enter AND/OR in positions 14 through 16 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 16 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 17 through 22 (spacing and skipping) must be blank on an AND line. In an OR relationship, positions 17 through 22 can be used; if they are blank, the definitions from the preceding line are used. Positions 7 through 13 must be blank when AND/OR is specified.

An unlimited number of AND/OR lines can be specified on the output specifications.

Position 15 (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 EXCPT is used. See Chapter 11, “Operation Codes” for further information on the EXCPT operation code.

Position 15 indicates the type of record to be written. Position 15 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, all lines of each record type (H, D, T, or E) are handled at separate times within the program cycle.

Positions 16-18 (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 66.

This entry must appear on the same line that contains the record type (H, D, T, E) specification (position 15). If an OR line is used following an ADD or DEL entry, this entry applies to the OR line also.

Position 16 (Fetch Overflow/Release)

Entry	Explanation
Blank	Must be blank for all files except printer files (PRINTER specified in positions 40 through 46 of the file description specifications). If position 16 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 16 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 23 through 31 are satisfied. An overflow indicator cannot be specified on the same line as fetch overflow.

If an overflow indicator has not been specified in positions 33 and 34 of 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.

The form length and overflow line can be specified on the line counter 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 16 of the corresponding output specifications. See the “REL (Release)” on page 334 operation for further information on releasing devices.

Positions 17-22 (Space and Skip)

Use positions 17 through 22 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 PRTCTL (printer control option) is not specified on the file description specifications, an entry must be made in one of the following positions when the device is PRINTER: 17 (space before), 18 (space after), 19 and 20 (skip before), or 21 and 22 (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 position 17 (space before) or in positions 19 through 22 (skip before and skip after) and no entry is made in position 18 (space after), no space occurs after printing. When PRTCTL is specified, it is used only on records with blanks specified in positions 17 through 22.

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.

Position 17 (Space Before)

Entry	Explanation
0 or Blank	No spacing
1	Single spacing
2	Double spacing
3	Triple spacing.

Position 18 (Space After)

Entry	Explanation
0 or Blank	No spacing
1	Single spacing
2	Double spacing
3	Triple spacing.

Positions 19-20 (Skip Before)

Entry	Explanation
0 or Blank	No skipping occurs.
01-99	Skip to lines 1 to 99 before printing for printer files.
A0-A9	Skip to lines 100 to 109 before printing for printer files.
B0-B2	Skip to lines 110 to 112 before printing for printer files.

Positions 21-22 (Skip After)

Entry	Explanation
0 or Blank	No skipping occurs.
01-99	Skip to lines 1 to 99 after printing for printer files.
A0-A9	Skip to lines 100 to 109 after printing for printer files.
B0-B2	Skip to lines 110 to 112 after printing for printer files.

Positions 23-31 (Output 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 24 and 25, 27 and 28, and 30 and 31. When an N is entered in positions 23, 26, or 29, 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.

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 14 through 16 of the following line and specify the additional indicators in positions 23 through 31 on that line.

For an AND relationship, fetch overflow (position 16) can only be specified on the first line. Positions 17 through 22 (spacing and skipping) must be blank for all AND lines.

An overflow indicator must be defined on the file description specifications (positions 33 and 34) 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 14 and 15 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 17 through 22) 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 16) is used, it must be specified on each OR line.

Positions 32-37 (EXCPT Name)

When the record type is an exception record (indicated by an E in position 15), a name can be placed in these positions of the record line. The EXCPT operation can specify the name assigned to a group of the records to be output. This name is called an EXCPT name. An EXCPT name must follow the rules for using symbolic names. A group of any number of output records can use the same EXCPT name, and the records do not have to be consecutive records.

When the EXCPT operation is specified without an EXCPT name, only those exception records without an EXCPT name are checked and written if the conditioning indicators are satisfied.

When the EXCPT operation specifies an EXCPT name, only the exception records with that name are checked and written if the conditioning indicators are satisfied.

The EXCPT name is specified on the main record line and applies to all AND/OR lines.

If an exception record with an EXCPT name is conditioned by an overflow indicator, the record is written only during the overflow portion of the RPG/400 cycle or during fetch overflow. The record is not written at the time the EXCPT operation is processed.

An EXCPT operation with no fields can be used to release a record lock in a file. The UNLCK operation can also be used for this purpose. In Figure 32 on page 176, the record lock in file RCDA is released by the EXCPT operation.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C           KEY       CHAINRCDA
C                     EXCPTRELESE
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
O
O*
ORCDA       E              RELESE
O*                    (no fields)
```

Figure 32. Record Lock in File Released by EXCPT Operation

Field Description and Control Entries

Entries in positions 23 through 70 determine under what conditions and in what format fields of a record are to be written.

Each field is described on a separate line. No entries are permitted in positions 7 through 22 of a field description line. Field description and control information for a field begins on the line following the record identification line.

Positions 23-31 (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 177 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 23-31 (Output Indicators)" on page 174. Indicators used to condition field descriptions lines cannot be specified in an AND/OR relationship. Conditioning indicators cannot be specified on format name specifications for program described WORKSTN files.

Positions 32-37 (Field Name)

In positions 32 through 37, 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 45 through 70
- A table name, array name, or array element
- A named constant
- The RPG/400 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.

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 45 through 70. If a field name is entered in positions 32 through 37, positions 7 through 22 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 40 through 43. 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 "LOKUP (Look Up)" on page 286 operation is written. The first element of a table is written if no successful LOKUP operation was done.

The conditions for a field and the conditions for the record it is contained in must be satisfied before the field is written out.

PAGE, PAGE1-PAGE7

To use automatic page numbering, code PAGE in positions 32 through 35 as the name of the output field. Indicators specified in positions 23 though 31 do not condition the field, but rather control the resetting of the PAGE field. When indicators are specified and their conditions are met, the PAGE field is set to zero and incremented by 1 before output; when the conditions are not met, the PAGE field is incremented by 1 and then output. 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 Chapter 12, "RPG/400 Words with Special Functions."

*PLACE

*PLACE is an RPG/400 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 32 through 37, 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 39), editing (positions 38, 45 through 70), data format (position 44), 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/400 indicators as data.

Position 38 (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 38 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.

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/400 compiler unless the program is recompiled.

For more information on edit codes see Chapter 14, “Editing Numeric Fields” on page 403.

Position 39 (Blank After)

Entry	Explanation
Blank	The field is not reset.
B	The field specified in positions 32 through 37 is reset to blank or zero after the output operation is complete.

Position 39 is used to reset a numeric field to zeros or a character field to blanks. If the field is conditioned by indicators in positions 23 through 31, the blank after is also conditioned. This position must be blank for look-ahead, user date special words, *PLACE, named constants, and constants.

Resetting fields to zeros is useful 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 39) 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 may be set to blanks or zeros after the field is written to the output record.

Positions 40-43 (End Position)

Entry	Explanation
1-n	End position
K1-K8	Length of format name for WORKSTN file.

Positions 40 through 43 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 42 and 43 is K6. Leading zeros are permitted following the K, and the entry must be right-adjusted.

Valid entries for end positions are blanks, +nnn, –nnn, and nnnn. All entries in these positions must end in position 43. 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 40 through 43. 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 +nnn or –nnn entry specifies the placement of the field or constant relative to the end position of the previous field. The sign must be in position 40. The number (nnn) must be right-adjusted, but leading zeros are not required. To calculate the end position, use these formulas :

```
EP = PEP +nnn + FL
```

```
EP = PEP –nnn + 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 +nnn is equivalent to placing nnn positions between the fields. A -nnn causes an overlap of the fields by nnn positions. For example, if the previous end position (PEP) is 6, the number of positions to be placed between the fields (nnn) 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 +000. No positions separate the fields.

Position 44 (Data Format)

Entry	Explanation
Blank	The field is to be written in zoned decimal numeric or character format; numeric data is edited, or a constant is specified on this line.
P	The field is to be written in packed decimal format.
B	The field is to be written in binary format.
L	The numeric output field is to have a preceding (left) plus or minus sign.
R	The numeric output field is to have a following (right) plus or minus sign.

This position must be blank if editing is specified.

The entry in position 44 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.

Positions 45-70 (Constant or Edit Word)

Positions 45 through 70 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 of up to 24 characters can be placed in positions 45 through 70. The constant must begin in position 46 (apostrophe in position 45), 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 32 through 37) must be blank. Instead of entering a constant, you can use a named constant containing a maximum of 256 characters.

Edit Words

An edit word specifies the punctuation of numeric fields, including the printing of dollar signs, commas, periods, and sign status.

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 45 through 54. 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 40 through 43, where n is the length of the format name. For example, if the format name is '`CUSPMT`', enter `K6` in positions 42 and 43. A named constant can also be used.

Positions 71-74 (Reserved)

Positions 71 through 74 must be blank.

Positions 75-80 (Comments)

Positions 75 through 80 can be used for comments, or left blank.

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 37 of the record identification line identify the record format and determine under what conditions the records are to be written.

Positions 7-14 (Record Name)

Entry	Explanation
A valid record format name	A record format name must be specified for an externally described file.

Positions 14-16 (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 14-16 (Logical Relationship)” on page 171 for more information.

Position 15 (Type)

Entry	Explanation
H or D	Detail records
T	Total records
E	Exception records.

Position 15 indicates the type of record to be written. See “Position 15 (Type)” on page 171 for more information.

Position 16 (Release)

Entry	Explanation
R	Release a device after output.

See “Release” on page 173 for more information.

Positions 16-18 (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 16-18 (Record Addition/Deletion)” on page 172.

Positions 16-22 (Fetch Overflow/Space/Skip)

The fetch overflow, space, and skip entries for an externally described file must be blank. Space and skip entries for an externally described printer file are specified in the data description specifications.

Positions 23-31 (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 23-31 (Output Indicators)” on page 174.

Positions 32-37 (EXCPT Name)

An EXCPT name can be specified in these positions for an exception record line. See “Positions 32-37 (EXCPT Name)” on page 175 for more information.

Field Description and Control Entries

For externally described files, the only valid field description and control entries are conditioning indicators (positions 23 through 31), field name (positions 32 through 37), and blank after (position 39).

Positions 23-31 (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 23-31 (Output Indicators)” on page 174 for more information.

Positions 32-37 (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 39.

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, 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 38 (Edit Codes)

Position 38 must be blank because edit codes for externally described files are specified in the data description specifications.

Position 39 (Blank After)

Entry	Explanation
Blank	The field is not reset.
B	The field specified in positions 32 through 37 is reset to blanks or zeros after the output operation is complete. Do not specify *ALL in positions 32 through 37.

For more information on blank after, see “Position 39 (Blank After)” on page 178.

Positions 40-43 (End Position)

These positions must be blank because the field is placed in the output record in the positions specified by the data description specifications.

Position 44 (Data Format)

This position must be blank because the data format is defined in the data description specifications.

Positions 45-70 (Constant or Edit Word)

These positions must be blank because editing is specified in the data description specifications. If a constant is to be placed in an externally described file, it must be done by a calculation operation.

Positions 71-74 (Reserved)

Positions 71 through 74 must be blank.

Positions 75-80 (Comments)

Positions 75 through 80 can be used for comments, or left blank.

Chapter 11. Operation Codes

The RPG/400 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.

Many operation codes can be placed into categories. 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

(½)	Half adjust the result.
(n)	No lock.
(p)	Pad the result with blanks.
+	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 35 (Page 1 of 4). Operation Code Specifications Summary

Codes	Factor 1	Factor 2	Result Field	Resulting Indicators
ACQ	: QRGRCOPP	WORKSTN file		_ ER _
ADD(½)	Addend	Addend	Sum	+ – Z
ANDxx	Comparand	Comparand		
BEGSR	Subroutine name			
BITOF		Bit numbers	Character field	
BITON		Bit numbers	Character field	
CABxx	Comparand	Comparand	Label	HI LO EQ
CALL		Program name	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
CHEKR[2]	Comparator String	Base String:start	Right-most Position(s)	_ ER FD
CLEAR	*NOKEY	Structure or Variable		
CLOSE		File name		_ ER _
COMIT	Boundary			_ ER _
COMP[1]	Comparand	Comparand		HI LO EQ
DEBUG	Identifier	Output file	Debug info	
DEFN	*LIKE	Referenced field	Defined field	
DEFN	*NAMVAR	Internal program area	External data area	
DELET	Search argument	File name		NR ER _
DIV(½)	Dividend	Divisor	Quotient	+ – Z
DO	Starting value	Limit value	Index value	
DOUxx	Comparand	Comparand		
DOWxx	Comparand	Comparand		
DSPLY	Message identifier	Output queue	Response	_ ER _
DUMP	Identifier			
ELSE				
END		Increment value		
ENDCS				
ENDDO		Increment value		

[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.

Table 35 (Page 2 of 4). Operation Code Specifications Summary

Codes	Factor 1	Factor 2	Result Field	Resulting Indicators
ENDIF				
ENDSL				
ENDSR	Label	Return point		
EXCPT		EXCPT name		
EXFMT		Record format name		_ ER _
EXSR		Subroutine name		
FEOD		File name		_ ER _
FORCE		File name		
FREE		Program name		_ ER _
GOTO		Label		
IFxx	Comparand	Comparand		
IN	*LOCK	Data area name		_ ER _
ITER				
KFLD			Key field	
KLIST	KLIST name			
LEAVE				
LOKUP[1] (array) (table)	 Search argument Search argument	 Array name Table name	 Table name	 HI LO EQ HI LO EQ
MHHZO		Source field	Target field	
MHLZO		Source field	Target field	
MLHZO		Source field	Target field	
MLLZO		Source field	Target field	
MOVE (p)		Source field	Target field	+ – ZB
MOVEA (p)		Source	Target	+ – ZB
MOVEL (p)		Source field	Target field	+ – ZB
MULT (½)	Multiplicand	Multiplier	Product	+ – Z
MVR			Remainder	+ – Z
NEXT	Program device	File name		_ ER _
OCUR	Occurrence value	Data structure	Occurrence value	_ ER _
OPEN		File name		_ ER _
ORxx	Comparand	Comparand		
OTHER				
OUT	*LOCK	Data area name		_ ER _

[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.

Table 35 (Page 3 of 4). Operation Code Specifications Summary

Codes	Factor 1	Factor 2	Result Field	Resulting Indicators
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
REDPE (n)	Search argument	File name, Record name	Data structure	_ ER BOF
REL	Program device	File name		_ ER _
RESET	*NOKEY	Structure or Variable		
RETRN				
ROLBK				_ ER _
SCAN[2]	Comparator string:length	Base string:start	Left-most position(s)	_ ER FD
SELEC				
SETGT	Search argument	File name		NR ER _
SETLL	Search argument	File name		NR ER EQ
SETOF[1]				OF OF OF
SETON[1]				ON ON ON
SHTDN				ON _ _
SORTA		Array name		
SQRT (½)		Value	Root	
SUB(½)	Minuend	Subtrahend	Difference	+ – Z
SUBST (p)	Length to extract	Base string:start	Target string	_ ER _
TAG	Label			
TESTB[1]		Bit numbers	Character field	OF ON EQ
TESTN[1]			Character field	NU BN BL
TESTZ[1]			Character field	
TIME			Numeric field	
UNLCK		Data area, record, or file name		_ ER _
UPDAT		File name	Data structure	_ ER _
WHxx	Comparand	Comparand		
WRITE		File name	Data structure	_ ER EOF
XFOOT (½)		Array name	Sum	+ – 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.

Table 35 (Page 4 of 4). Operation Code Specifications Summary

Codes	Factor 1	Factor 2	Result Field	Resulting Indicators
XLATE (p)	From:To	String:start	Target String	_ ER _
Z-ADD (½)		Addend	Sum	+ – Z
Z-SUB (½)		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.

Arithmetic Operations

The arithmetic operations are:

- “ADD (Add)” on page 206
- “DIV (Divide)” on page 244
- “MULT (Multiply)” on page 306
- “MVR (Move Remainder)” on page 307
- “SQRT (Square Root)” on page 355
- “SUB (Subtract)” on page 356
- “XFOOT (Summing the Elements of an Array)” on page 375
- “Z-ADD (Zero and Add)” on page 378
- “Z-SUB (Zero and Subtract)” on page 379.

For examples of arithmetic operations, see Figure 33 on page 191.

Remember the following when specifying arithmetic operations:

- Arithmetic operations can be done only on numeric fields (including numeric subfields, numeric arrays, numeric array elements, numeric table elements, numeric named constants, numeric figurative constants, and numeric literals).
- Arithmetic operations are done on data in packed decimal format. Data maintained in other formats is converted to or from packed decimal format.
- 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.
- An arithmetic operation does not change factor 1 and factor 2 unless they are the same as the result field.
- Any data placed in the result field replaces the data that was there.
- The result field must be large enough to accommodate the results of the arithmetic operation because an RPG/400 program does not cause an error on an arithmetic overflow. If the result field is not large enough to accommodate the results, digits are dropped from either or both ends, depending on the location of the decimal point.
- If you use conditioning indicators, it is your responsibility to ensure that the DIV operation occurs immediately before the MVR operation. If the MVR operation occurs before the DIV operation, undesirable results occur. For example, error message RPG0907 may be issued.
- Half adjust (position 53) can be specified for all the arithmetic operations except for the MVR operation or for a DIV operation immediately followed by an MVR operation.

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.

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.

All arithmetic operations are done algebraically.

The results of all operations are signed (a plus sign is a hexadecimal F and a minus sign is a hexadecimal D) according to the following rules. The sign is in the zone portion of the low-order byte.

For information on using arrays with arithmetic operations, see “Specifying an Array in Calculations” on page 398.

Addition: If factor 1 and factor 2 have like signs, the result field sign is the same. If factor 1 and factor 2 have unlike signs, the result field sign is the same as the sign of the factor with the larger absolute value.

Subtraction: Change the sign of factor 2, and apply the rules for addition.

Multiplication: If factor 1 and factor 2 have like signs, the result field sign is plus (+). If factor 1 and factor 2 have unlike signs, the result field sign is minus (-).

Division: If factor 1 and factor 2 have like signs, the result field sign is plus (+). If factor 1 and factor 2 have unlike signs, the result field sign is minus (-). The sign of the remainder is the same as the factor 1 sign.

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.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++*
C*  Before the operations are processed, the field values are:
C*       A = 1.00       G = 2.77
C*       B = 10.0       H = 70
C*       C = 32         J = .6
C*       D = -20        K = 25
C*       E = 6.         L = 1.0, 1.7, -1.1             RESULTS
C*       F = 10.0                                      -------
C                   ADD  1         A       30         002
C             B     ADD  C         V       52         042.00
C             B     ADD  D         V                  -10.00
C*
C                   Z-ADDC         V                  032.00
C*
C                   SUB  1         E       30         005
C             C     SUB  B         W       51         0022.0
C             C     SUB  D         W                  0052.0
C*
C                   Z-SUBC         W                  -0032.0
C*
C                   MULT E         F       30         060
C             B     MULT G         X       84         0027.7000
C             B     MULT D         X                  -200.0000
C*
C                   DIV  B         H       30         007
C             C     DIV  J         Y       62         0053.33
C*
C                   MVR            Z       53         00.002
C*
C                   SQRT K         Z                  05.000
C*
C                   XFOOTL         Z                  01.600
```

Figure 33. Summary of Arithmetic Operations

Array Operations

The array operations are:

- “LOKUP (Look Up)” on page 286
- “MOVEA (Move Array)” on page 295
- “SORTA (Sort an Array)” on page 354
- “XFOOT (Summing the Elements of an Array)” on page 375.

See each operation for an explanation of its function.

Bit Operations

The bit operations are:

- "BITOF (Set Bits Off)" on page 209
- "BITON (Set Bits On)" on page 210
- "TESTB (Test Bit)" on page 361.

The BITOF and BITON operations allow you to turn off and on specific bits in a field specified in the result field. The specified 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.

In these operations, if factor 2 contains a 1-byte hexadecimal literal, the bits in the factor 2 entry will affect the result field entry in the same way a 1-byte character field would affect it.

Note: You can also format one or more characters at a time using move operations with hexadecimal literals. See "Move Operations" on page 198 for more detail.

Branching Operations

The branching operations are:

- "CABxx (Compare and Branch)" on page 212
- "ENDSR (End of Subroutine)" on page 259
- "EXCPT (Calculation Time Output)" on page 260
- "GOTO (Go To)" on page 271
- "ITER (Iterate)" on page 278
- "LEAVE (Leave a Do Group)" on page 284
- "TAG (Tag)" on page 360.

The GOTO operation (when used with a TAG operation) allows branching. For example:

- Several operations can be skipped when certain conditions occur.
- Certain operations can be done for several, but not all, record types.
- Several operations can be repeated.

The EXCPT operation allows records to be written during calculation time instead of at output time.

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 214
- “FREE (Deactivate a Program)” on page 269
- “PARM (Identify Parameters)” on page 318
- “PLIST (Identify a Parameter List)” on page 320
- “RETRN (Return to Caller)” on page 338.

The CALL, FREE, and RETRN operations allow an RPG/400 program to transfer control to other programs. With the CALL, FREE, and RETRN function, the PLIST and PARM operations can be used to allow the calling and called programs to address the same data.

See each operation for an explanation of its function.

Compare Operations

The compare operations are:

- “ANDxx (And)” on page 207
- “COMP (Compare)” on page 236
- “CABxx (Compare and Branch)” on page 212
- “CASxx (Conditionally Invoke Subroutine)” on page 218
- “DOUxx (Do Until)” on page 248
- “DOWxx (Do While)” on page 251
- “IFxx (If)” on page 273
- “ORxx (Or)” on page 315
- “WHxx (When True Then Select)” on page 371.

In the ANDxx, CABxx, CASxx, DOUxx, DOWxx, IFxx, ORxx, and WHxx 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 certain conditions. Resulting indicators assigned in positions 54 through 59 are set according to the results of the operation, or a branch occurs based on the results of the operation. No fields are changed by these operations.

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.

- If character 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.
- All numeric comparisons are algebraic. A plus (+) value is always greater than a minus (-) value.
- Blanks within numeric fields are assumed to be zeros.
- Numeric fields are converted to packed decimal format, if necessary, before they are compared.
- If an alternate collating sequence (position 26 of the control specification) has been specified for the comparison of character fields, the fields 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. However, the actual field value will not change.
- A character field cannot be compared with a numeric field.
- 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 with DOUxx, DOWxx, IFxx, and WHxx.

Data-Area Operations

The data area operations are:

- "IN (Retrieve a Data Area)" on page 276
- "OUT (Write a Data Area)" on page 317
- "UNLCK (Unlock a Data Area or Release a Record)" on page 368.

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.

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 or UNLCK operation, the exclusive allow read lock state is released.

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 the error message RPG0431 indicating that the data area is not available.

Remember the following when using the IN, OUT, and UNLCK 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 UNLCK operations can be done on a data area, you must specify that data area in the result field of an *NAMVAR DEFN statement. (For

further information on the DEFN statement, see "DEFN (Field Definition)" on page 239.)
- The data-area operations can be done on a data-area data structure that is implicitly retrieved only if the data-area data structure name is specified in the result field of an *NAMVAR DEFN statement.
- A locked data area cannot be updated or locked by another RPG/400 program; however, the data area can be retrieved in your own program 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 *NAMVAR DEFN statement.

A data structure defined with a U in position 18 of the input specifications form indicates that the data structure is a 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.

Specify *LDA in factor 2 of a *NAMVAR DEFN statement to define the LDA data structure.

Use the *NAMVAR DEFN operation with *PDA in factor 2 to define the name in the result field as the PDA data area. The result field follows the current conventions for *NAMVAR DEFN.

Declarative Operations

The declarative operations do not cause action to occur; they can be specified anywhere within calculations. 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 control level entry is for documentation purposes only. The declarative operations are:

- "DEFN (Field Definition)" on page 239
- "KFLD (Define Parts of a Key)" on page 281
- "KLIST (Define a Composite Key)" on page 282
- "PARM (Identify Parameters)" on page 318
- "PLIST (Identify a Parameter List)" on page 320
- "TAG (Tag)" on page 360.

The DEFN 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 operation to allow a called program access to parameters from a calling program.

The TAG operation names the destination of a branching operation such as GOTO or CABxx.

File Operations

The file operation codes are:

File operations can be used with both program described and externally described files (F or E respectively in position 19 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 command is used with the MBR (*ALL) parameter specified, the SETLL, SETGT and CHAIN operations only process the current file member. For more information, refer to the *Database Guide*.

The WRITE and UPDAT 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 REDPE 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 REDPE operations, data is transferred directly between the file and the data structure, bypassing the normal field extract function. 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. However, the data structure must be defined on the input specifications.

If an input operation (CHAIN, EXFMT, READ, READC, READE, READP, REDPE) 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 an N in position 53 of a CHAIN, READ, READE, READP, or REDPE operation for an update disk file, a record is read without locking. If no value is specified in position 53, 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/400 error handler.

Indicator-Setting Operations

The "SETON (Set On)" and "SETOF (Set Off)" operations set (on or off) indicators specified in positions 54 through 59. 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 SETOF operation.
- Setting L1 through L9 on or off with a SETON or SETOF operation does not automatically set any lower control level indicators.

Information Operations

The information operations are:

- "DEBUG (Debug Function)" on page 237
- "DUMP (Program Dump)" on page 256
- "SHTDN (Shut Down)" on page 353
- "TIME (Time of Day)" on page 366.

The DEBUG operation writes the following information that you can use to debug an RPG/400 program:

- The source statement sequence number of the DEBUG operation
- The contents of factor 1 (if factor 1 is specified) that identifies the DEBUG written information
- All indicators that are on at the time the operation is encountered
- The contents of the result field (if the result field contains an entry).

The information is sent to the file named in factor 2, or, if factor 2 is blank, the information is sent to a system defined file.

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 54 and 55 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 231
- “RESET (Reset)” on page 335.

The CLEAR operation sets all elements in a structure or variable to zero, blank, or '0', depending on the field type (numeric, character, or indicator). This allows you to clear structures globally, instead of element by element.

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 factor 2 of a RESET operation.

When these operation codes are applied to record formats, only fields which are output are affected. See “CLEAR (Clear)” on page 231 and “RESET (Reset)” on page 335 for more detail.

For more information see “Initialization” in Chapter 9 of the *RPG/400 User’s Guide*.

Message Operation

The message operation “DSPLY (Display Function)” on page 253 allows communication between the program and the system console or between the program and the display workstation that requested the program.

Move Operations

The move operations are:

- “MOVE (Move)” on page 292
- “MOVEA (Move Array)” on page 295
- “MOVEL (Move Left)” on page 302.

Move operations transfer all or part of factor 2 to the result field. Factor 2 remains unchanged. Factor 1 must be blank. Resulting indicators can be specified in positions 54 through 59., 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.

In the move operations, factor 2 and the result field are generally of the same type (both numeric or both character). However, you can use the move operations to change numeric fields to character fields and character fields to numeric fields. To change a numeric field to a character field, enter the name of the numeric field in factor 2 and specify a character result field. To change a character field to a numeric field, enter the name of the character field in factor 2 and specify a numeric result field.

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 69 on page 293 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 71 on page 304 for an example.)

If move operations are specified to move data into 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.

If you specify P in position 53 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, 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, only the resulting indicator in positions 58 and 59 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 (54-55)	Set on if the result field is greater than 0.
Low (56-57)	Set on if the result field is less than 0.
Equal (58-59)	Set on if the result field is equal to 0.

Move Zone Operations

The move zone operations are:

- “MHHZO (Move High to High Zone)” on page 288
- “MHLZO (Move High to Low Zone)” on page 289
- “MLHZO (Move Low to High Zone)” on page 290
- “MLLZO (Move Low to Low Zone)” on page 291.

The move zone operations move only the zone portion of a character.

A minus (-) sign in a move zone operation does not result in a negative character in the result field, because a minus sign is represented by a hexadecimal 60 internally and a D zone is required for a negative character. Characters J through R have D zones and can be used to obtain a negative value

(J = hexadecimal D1, ..., R = hexadecimal D9).

Note: 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.

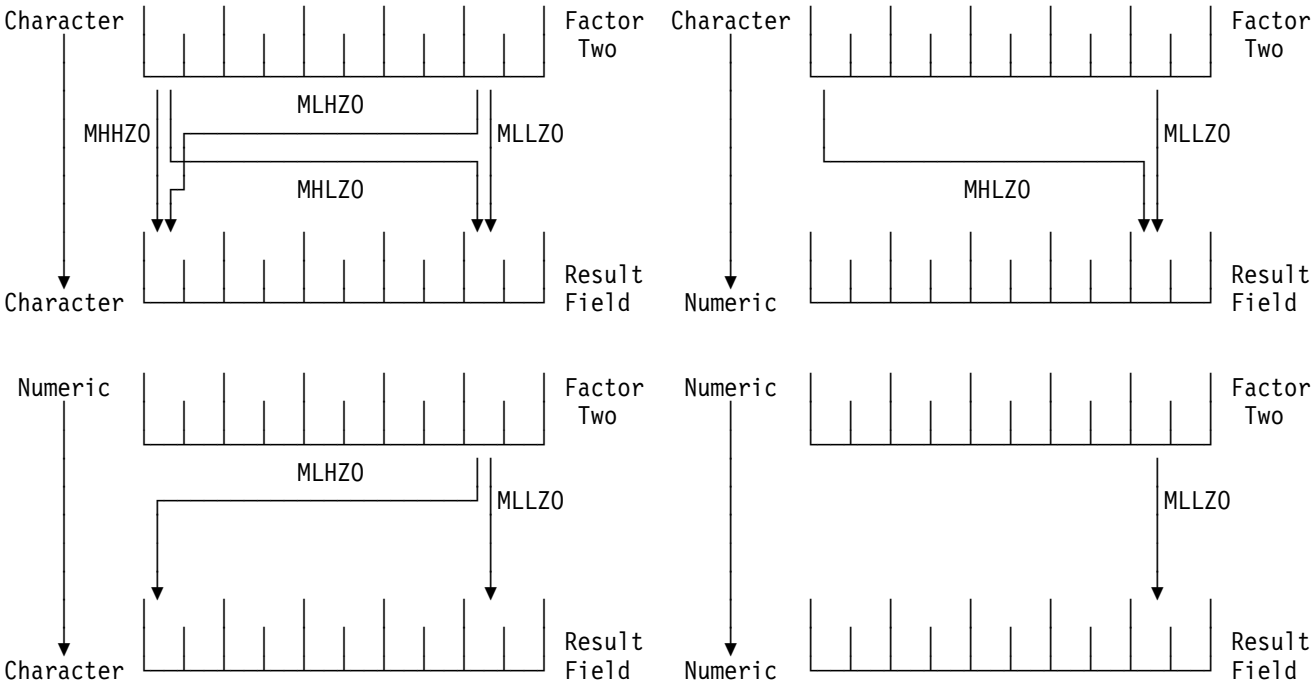


Figure 34. Function of MOVE Zone Operations

String Operations

The string operations include concatenation, scanning, substringing, translation, and verification. String operations can only be used on character fields.

The string operations are:

- “CAT (Concatenate Two Character Strings)” on page 220
- “CHECK (Check Characters)” on page 227
- “CHEKR (Check Reverse)” on page 229
- “SCAN (Scan Character String)” on page 340
- “SUBST (Substring)” on page 357
- “XLATE (Translate)” on page 376.

The CAT operation concatenates two character strings to form one.

The CHECK and CHEKR operations verify that each character in factor 2 is among the valid characters in factor 1. CHECK verifies from left to right and CHEKR from right to left.

The SCAN operation scans a base character string for occurrences of a second specified character string.

The SUBST operation extracts a specified character string from a base character string.

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, CHEKR, and SUBST operations (where it applies only to factor 2).

If you specify P in position 53 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.

Structured Programming Operations

The structured programming operations are:

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 SELEC, WHxx, 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 SELEC operation. The WHxx 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 WHxx 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, WHxx, 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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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* Algebraically, this would be represented as:
C* ( ( (FIELDA > FIELDB) & (FIELDA >= FIELDC) ) | (FIELDA < FIELDD) )
C*
C           FIELDA    IFGT FIELDB
C           FIELDA    ANDGEFIELDC
C           FIELDA    ORLT FIELDD
C                     SETON                     25
C                     ENDIF
```

Figure 35. Example of AND/OR Precedence

The CASxx operation allows a subroutine to be conditionally selected for processing. An ENDCS operation ends a CASxx group. For more information about the CASxx operation, see “Compare Operations” on page 193.

A DO, DOUxx, DOWxx, IFxx, or SELEC 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 SELEC 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 WHxx operation codes are the same as those given under “Compare Operations” on page 193.

In the ANDxx, CASxx, DOUxx, DOWxx, IFxx, ORxx, and WHxx operations, xx can be:

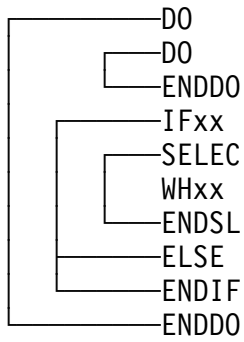
xx	Meaning
GT	Factor 1 is greater than factor 2.
LT	Factor 1 is less than factor 2.
EQ	Factor 1 is equal to factor 2.
NE	Factor 1 is not equal to factor 2.
GE	Factor 1 is greater than or equal to factor 2.
LE	Factor 1 is less than or equal to factor 2.
Blanks	Unconditional processing (CASxx only).

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 SELEC 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:



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 SELEC 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/400 subroutine is a group of calculation specification statements in a program that can be processed several times in that program. The RPG/400 subroutine operations are:

- "BEGSR (Beginning of Subroutine)" on page 208
- "ENDSR (End of Subroutine)" on page 259
- "EXSR (Invoke Subroutine)" on page 263
- "CASxx (Conditionally Invoke Subroutine)" on page 218.

RPG/400 subroutine specifications must follow all other calculation operations that can be processed for a program; however, the PLIST, PARM, KLIST, KFLD, and DEFN 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 from any point in the calculation operations. 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 264.

Test Operations

The test operations are:

- "TESTB (Test Bit)" on page 361
- "TESTN (Test Numeric)" on page 363
- "TESTZ (Test Zone)" on page 365.

The TESTx operations allow you to test character fields specified in the result field.

Operation Codes List

The remainder of 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 56 and 57, 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 must only be used with a multiple device file. See the section on "Multiple-Device Files" in the chapter about using WORKSTN files in the *RPG/400 User's Guide*.

ADD (Add)

Code	Factor 1	Factor 2	Result Field	Indicators
ADD(½)	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 189.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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 36. ADD Operations

ANDxx (And)

Code	Factor 1	Factor 2	Result Field	Indicators
ANDxx	Comparand	Comparand		

If you specify this optional operation, it must immediately follow a ANDxx, DOUxx, DOWxx, IFxx, ORxx, or WHxx operation. With ANDxx, you can specify a complex condition for the DOUxx, DOWxx, IFxx, and WHxx 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 WHxx operation. Conditioning indicator entries (positions 9 through 17) 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 either both character data or both numeric data. The comparison of factor 1 and factor 2 follows the same rules as those given for the compare operations. See “Compare Operations” on page 193.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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                     WRITERCRSN
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 37. 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/400 subroutine. Factor 1 contains the subroutine name. You must 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 continuation option if 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 60 on page 265 for an example of coding subroutines; see “Subroutine Operations” on page 203 for general information on subroutine operations.

BITOF (Set Bits Off)

Code	Factor 1	Factor 2	Result Field	Indicators
BITOF		Bit numbers	Character field	

The BITOF 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 BITOF to format a character, you should use both BITON and BITOF: BITON to specify the bits to be set on (=1), and BITOF 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.

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 38 on page 211 for an example of the BITOF 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 BITOF: BITON to specify the bits to be set on (=1), and BITOF 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.

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...
I..............Namedconstant+++++++++C.........Fldnme......... ....
I              '01234567'             C         BITNC
I              X'0F'                  C         HEXNC
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  Before the operations are processed, the bit settings are:
C*                FLDA = 00000000
C*                FLDB = 00000000
C*                FLDC = 11111111
C*                FLDD = 11000000
C*                FLDE = 11000000
C*                FLDF = 10000001
C*                FLDG = 11111111                      AFTER
C*                FLDH = 00000000                    OPERATIONS
C*                FLDI = 11001010                    ----------
C*
C                     BITON'04567'   FLDA             = 10001111
C                     BITON'3'       FLDB             = 00010000
C                     BITON'3'       FLDC             = 11111111
C                     BITON'3'       FLDD             = 11010000
C                     BITONFLDE      FLDF             = 11000001
C                     BITONX'01'     FLDH             = 00000001
C*
C                     BITOF'0'       FLDG             = 01111111
C*
C                     BITOFBITNC     FLDI             = 00000000
C                     BITONHEXNC     FLDI             = 00001111
```

Figure 38. BITON and BITOF 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 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 54-59) 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 character literal, a numeric literal, a named constant, a figurative constant, a field name, a table name, an array element, or a data structure name. Both the factor 1 and the factor 2 entries must be character data, or both must be numeric.

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/400 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 193 for the rules for comparing factor 1 with factor 2.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*          The field values are:
C*          FLDA = 100.00
C*          FLDB = 105.00
C*          FLDC = ABC
C*          FLDD = ABCDE
C*
C*           Branch to TAGX.
C            FLDA      CABLTFLDB      TAGX
C*
C*           Branch to TAGX.
C            FLDA      CABLEFLDB      TAGX
C*
C*           Branch to TAGX; indicator 16 is off.
C            FLDA      CABLEFLDB      TAGX       16
C*
C*           Branch to TAGX; indicator 17 is off, indicator 18 is on.
C            FLDA      CAB  FLDB      TAGX       1718
C*
C*           Branch to TAGX; indicator 19 is on.
C            FLDA      CAB  FLDA      TAGX         19
C*
C*           No branch occurs.
C            FLDA      CABEQFLDB      TAGX
C*
C*           No branch occurs; indicator 20 is on.
C            FLDA      CABEQFLDB      TAGX        20
C*
C*           No branch occurs; indicator 21 is off.
C            FLDC      CABEQFLDD      TAGX       21
C            TAGX      TAG
```

Figure 39. 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 8 characters. The total length of 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 blanks found before or after the slash are included in the program or library name. If the first or last character in the entry is a slash, a blank library or program name, respectively, is assumed. (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.) *LIBL and *CURLIB are not supported.

For System/38 Environment programs, if you specify the library name for the called program, the program name must be immediately followed by a period and then the library name.

Program references are grouped to avoid the overhead of resolving to the target program. All references (using a CALL or FREE 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. If your program depends on a resolve taking place on a reference by variable name, code a FREE operation using that variable name. This causes a subsequent reference, using that variable, to resolve to the program whether or not the value has changed. 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 40 on page 215 illustrates the grouping of program references.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
I..............Namedconstant+++++++++C.........Fldnme......... ....
I              'LIB1/PGM1'           C         CALLA
I              'PGM1'                C         CALLB
I              'LIB/PGM2'            C         CALLC
I*
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C                     CALL CALLA
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* CALLA above because CALLA has a different library name specified
C* (LIB1).
C*
C                     CALL 'PGM1'
C                     CALL CALLB
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                     FREE CALLC
C*
```

Figure 40 (Part 1 of 2). Example of Grouping of Program References

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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.  The call following
C* the free operation will result in a resolve taking place because
C* the free operation will force that to occur.  Note that the free
C* operation itself will not result in a resolve occurring. (That
C* is, that operation will use the current program object pointer
C* for calls using variable CALLV.)
C*
C                   MOVE 'PGM1'    CALLV  21
C                   CALL CALLV
C                   CALL CALLV
C                   FREE CALLV
C                   CALL CALLV
```

Figure 40 (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 54 and 55 must be blank. Any valid resulting indicator can be specified in positions 56 and 57 to be set on for an error returned from the called program and in positions 58 and 59 to be set on if the called program is an RPG/400 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. A referenced program is referenced on a CALL operation only. Using DSPPGMREF, you can query the names of programs called by way of named constants or literals. To make this information available to DSPPGMREF, recompile your program.

Figure 41 on page 217 illustrates the use of the CALL operation.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*  The CALL operation calls PROGA and allows PROGA to access
C*  FLDA and FLDB, defined elsewhere. PROGA is processed using the
C*  contents of FLDA and FLDB.  After PROGA is processed, control
C*  returns to the next statement to be processed after the last
C*  PARM statement.
C*
C*
C                   CALL 'PROGA'
C                   PARM           FLDA
C                   PARM           FLDB
```

Figure 41. CALL Operation

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 54 through 59). If factor 1 and factor 2 are not blanks, both must be character data, or both must be numeric. In a CASbb operation, factor 1 and factor 2 are required only if resulting indicators are specified in positions 54 through 59.

The result field must contain the name of a valid RPG/400 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 CASbb operation with no resulting indicators specified in positions 54 through 59 is functionally identical to an EXSR operation, because it causes the unconditional running of the subroutine named in the result field of the CASbb operation. Any CASxx operations that follow an unconditional CASbb operation in the same CAS group are never tested. Therefore, the normal placement of the unconditional CASbb 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 193 for further rules for the CASxx operation.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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    CASGEFIELDB    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    CASEQFIELDC    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 42. CASxx Operation

CAT (Concatenate Two Character 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 character string specified in factor 2 to the end of the character string specified in factor 1 and places it in the result field. If no factor 1 is specified, factor 2 is concatenated to the end of the result field string.

Factor 1 can contain a character 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 character string, and may contain the number of blanks to be inserted between the concatenated strings. Its format is the character string, followed by a colon, followed by the number of blanks. The character 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. If the number of blanks is specified, however, the trailing blanks of factor 1 are ignored and only as many blanks as specified are included in the result between the last nonblank character in factor 1 and the first character of factor 2. Leading blanks in factor 2 are always included. For example, if you have:

```
C      'bMIKEbb'  CAT  'bbSMITHb':1  Name
```

the value of the result field after this statement is executed is:

```
'bMIKEbbbSMITHb'
```

Note: The leading blanks in factors 1 and 2 and the trailing blanks in factor 2 are placed in the result unchanged. Since one was specified as the number of blanks, factor 1 was copied left justified to the result field, a blank was added following the rightmost nonblank character, and factor 2 was appended to the result. Since factor 2 had two leading blanks, the total number of blanks between the two now concatenated fields is three.

The result field must be character, 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 specified in the operation extender field (position 53) 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.

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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C* CAT concatenates LAST to NAME and inserts one blank as specified
C* in factor 2.  TEMP contains 'Mr.ЬSmith'.
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 in position 53
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/400ЬЬЬ'
C* after concatenation.
C                     MOVE *ALL'*'   TEMP   10
C                     MOVE '/400'    STRING  4
C           'RPG'     CAT  STRING    TEMP      P
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 'RPGIIIЬЬЬЬ'.
C                     MOVE 'RPG  '   NAME    5
C                     MOVE 'III  '   LAST    5
C                     Z-ADD0         NUM     10
C           NAME      CAT  LAST:NUM  TEMP    10P
```

Figure 43. CAT Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C* The following example shows leading blanks in factor 2.  After
C* the CAT, the RESULT contains 'MR.ßSMITH'.
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 'ABCßßßßßß'; FLD1 contains 'XYZ'.
C*  After the concatenation, FLD2 contains 'ABCßßXYZß'.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C                   MOVEL'ABC'     FLD2    9 P
C                   MOVE 'XYZ'     FLD1    3
C                   CAT  FLD1:2    FLD2
```

Figure 44. CAT Operation

CHAIN (Random Retrieval from a File)

Code	Factor 1	Factor 2	Result Field	Indicators
CHAIN (n)	Search argument	File name	Data structure	NR ER _

The CHAIN operation retrieves a record from a full procedural file (F in position 16 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 (Define a Composite Key)" 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 19 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 196 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 specified in the ID entry of the file specifications continuation line. If there is no such entry, data is read from the device for the last successful input operation to the file.

If the file is specified as INPUT, all records are read without locks and position 53 must be blank. If the file is specified as UPDATE, all records are locked if position 53 is blank.

If you are reading from an update disk file, you can specify an N in position 53 to indicate that no lock should be placed on the record when it is read. See the *RPG/400 User's Guide* for more information.

Positions 54 and 55 must contain an indicator that is set on if no record in the file matches the search argument. Positions 56 and 57 can contain an indicator to be

set on if the CHAIN operation is not completed successfully. Positions 58 and 59 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 next sequential record following 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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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       CHAINFILEX               60    INDICATOR 60
C*                                                   IF NOT FOUND
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 GOTO 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 GOTO operation.
C*
C   60                GOTO NOTFND
```

Figure 45. CHAIN Operation with a File Name in Factor 2

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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, the calculations after the GOTO
C*  operation are processed.
C*
C           KEY       CHAINREC1                 72    INDICATOR 72
C*                                                     IF NOT FOUND
C           KEY       KLIST
C                     KFLD           FLD1
C                     KFLD           FLD2
C           *IN72     IFEQ *OFF
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  The UPDAT operation modifies all the fields in the REC1 record.
C*
C                     UPDATREC1                       UPDATE
C                     ENDIF
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  The following example shows a CHAIN with no lock.
C*
C                     MOVE 3         KEY
C           KEY       CHAININPUT              N
```

Figure 46. 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). 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 character, 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 must be character, and 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 58-59.

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 17.

The indicator in positions 56-57 is turned on if an error occurs. The indicator in positions 58-59 is turned on if any incorrect characters are found.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C* Because factor 1 is a blank, CHECK indicates the position
C* of the first nonblank character.  If STRING contains 'bbbthe',
C* NUM will contain the value 4.
C*
C           ' '       CHECKSTRING    NUM     20
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
I..............Namedconstant+++++++++C.........Fldnme......... ....
I*
I* After the following example, N=6 and the found indicator 90
I* is on.  Because the start position is 2, the first nonnumeric
I* character found is the '.'.
I*
I              '0123456789'          C         DIGITS
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C                     MOVE '$2000.'  SALARY
C           DIGITS    CHECKSALARY:2  N             90
```

Figure 47. CHECK Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
I..............Namedconstant+++++++++C.........Fldnme......... ....
I*
I* The following example checks that FIELD contains only the letters
I* A to J.  As a result, ARRAY=(136000) after the CHECK operation.
I* Indicator 90 turns on.
I*
I              'ABCDEFGHIJ'          C         LETTER
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C                     MOVE '1A=BC*'  FIELD   6
C           LETTER    CHECKFIELD     ARRAY         90
C*
C* In the following example, because FIELD contains only the
C* letters A to J, ARRAY=(000000).  Indicator 90 turns off.
C*
C                     MOVE 'FGFGFG'  FIELD   6
C           LETTER    CHECKFIELD     ARRAY         90
```

Figure 48. CHECK Operation

CHEKR (Check Reverse)

Code	Factor 1	Factor 2	Result Field	Indicators
CHEKR	Comparator string	Base string:start	Right-position	_ ER FD

The CHEKR operation verifies that each character in the base string (factor 2) is among the characters indicated in the comparator string (factor 1). 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 character, 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 must be character, and 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. The result field is an optional field; if you do not specify it, you must specify the found indicator in position 58-59.

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 17.

The indicator in positions 56-57 is turned on if an error occurs. The indicator in positions 58-59 is turned on if any incorrect characters are found.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C* Because factor 1 is a blank character, CHEKR indicates the
C* position of the first nonblank character.  This use of CHEKR
C* allows you to determine the length of a string.  If STRING
C* contains 'ABCDEF   ', NUM will contain the value 6.
C*
C           ' '       CHEKRSTRING    NUM     20
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
I..............Namedconstant+++++++++C.........Fldnme......... ....
I*
I* After the following example, N=1 and the found indicator 90
I* is on. Because the start position is 5, the operation begins
I* with the rightmost 0 and the first nonnumeric found is the '$'.
I*
I              '0123456789'          C         DIGITS
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C                     MOVE '$2000.'  SALARY  6
C           DIGITS    CHEKRSALARY:5  N             90
E*
E* The following example checks that FIELD contains only the letters
E* A to J.  As a result, ARRAY=(876310) after the CHEKR operation.
E* Indicator 90 turns on.
E*
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments
E                     ARRAY       6  1 0
I..............Namedconstant+++++++++C.........Fldnme......... ....
I              'ABCDEFGHIJ'          C         LETTER
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C                     MOVE '1A=BC***'FIELD   8
C           LETTER    CHEKRFIELD     ARRAY         90
```

Figure 49. CHEKR Operation

CLEAR (Clear)

Code	Factor 1	Factor 2	Result Field	Indicators
CLEAR	*NOKEY	Structure or Variable		

The CLEAR operation sets elements in a structure (record format, data structure, array, or table) or a variable (field, subfield, or indicator), to zero, blank or '0', depending on field type (numeric, character, or indicator). 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 factor 2 contains a DISK record format name; in which case, it can contain *NOKEY to indicate that all fields except key fields are to be cleared.

Factor 2 contains the structure or variable that is set to zero, blank, or '0'. 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. If you have partially overlapping fields of different definitions, data that is not valid could exist in numeric 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 factor 2 using an array index, only the element specified is cleared.

Note that when the CLEAR operation is applied to a record-format name, 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 affect 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.

| For more information see “Initialization” in Chapter 10 of the *RPG/400 User’s Guide*.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
I.................................PFromTo++DField+L1M1FrPlMnZr...
I*
I*  In the following example, CLEAR sets all subfields in the data
I*  structure DS1 to their defaults, CHAR to blank, NUM to zero.
IDS1         DS
I                                        2   50NUM
I                                       20  30 CHAR
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C                    CLEARDS1
```

Figure 50. CLEAR Operation

Figure 51 on page 232 shows an example of the field initialization for the CLEAR record format.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
A* FLD1 and FLD2 are defined as output 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.
A*
AAN01N02N03T.Name++++++RLen++TDpBLinPosFunctions++++++++++++++++++++*
A          R FMT01
A  10        FLD1          10A  I  2 30
A            FLD2          10A  O  3 30
A            FLD3          10A  B  4 30
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilenameIPEAF....RlenLK1AIOvKlocEDevice+......KExit++Entry+A....U1.
FWORKSTN CF  E                    WORKSTN
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
I..............Namedconstant+++++++++C.........Fldnme......... ....
I              'INPUT DATA'           C         IN
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C                    CLEARFMT01
C                    WRITEFMT01
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                    MOVELFLD003    FLD002
C                    MOVELFLD001    FLD003
C                    CLEAR*IN04
C                    ENDIF
C                    MOVELIN        FLD001
C*
```

```
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                     RESETFMT01
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                     CLEARFMT01
C                     CLEAR*IN12
C                     ENDIF
C  N03                WRITEFMT01
C                     ENDDO
C                     SETON                     LR
```

Figure 51 (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 16 of the file description specifications) in factor 2.

You can specify a resulting indicator in positions 56 and 57 to be set on if the CLOSE operation is not completed successfully. Positions 54, 55, 58, and 59 must be blank.

Multiple CLOSE operations to a file already closed are valid. A second close to the same file has no effect on that file.

If an array or table is to be written to an output file (specified in positions 19 through 26 of the extension specifications), the array or table dump does not occur if the file is closed (by a CLOSE operation) at LR time when the file is written. If the file is closed, it must be reopened for the dump to occur.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  The explicit CLOSE operation closes FILEB.
C*
C                   CLOSEFILEB
C*
C*  The explicit CLOSE *ALL operation closes all files in the
C*  program.  To reopen the files, you must specify an explicit
C*  OPEN for each file.  If the CLOSE operation is not completed
C*  completed successfully, indicator 17 is set on and the
C*  program branches to the label specified in the GOTO statement.
C                   CLOSE*ALL                  17
C   17              GOTO NOCOMP                    CLOSE FAILED
C                   :
C                   :
C                   :
C           NOCOMP  TAG
```

Figure 52. CLOSE Operation

COMIT (Commit)

Code	Factor 1	Factor 2	Result Field	Indicators
COMIT	Boundary			_ ER _

The COMIT operation:

- Makes all the changes to your files that have been specified in output operations since the previous COMIT or "ROLBK (Roll Back)" operation (or since the beginning of operations under commitment control if there has been no previous COMIT or ROLBK operation).
- 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 COMIT operation, or by another program in the same routing step. The program issuing the COMIT operation does not need to have any files under commitment control. The COMIT operation does not change the file position.

Commitment control starts when the COMIT operation is executed or when the CL command STRCMTCTL is executed. See the chapter on "Commitment Control" in the *RPG/400 User's Guide* for more information.

In factor 1, you can specify a literal, named constant, array element, table name, data structure, or data structure subfield to identify the boundary between the changes made by this COMIT operation and subsequent changes. If you leave factor 1 blank, the identifier is null.

The optional indicator in positions 56 and 57 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 be either both character or both numeric. As a result of the comparison, indicators are set on as follows:

High: (54-55) Factor 1 is greater than factor 2.
Low: (56-57) Factor 1 is less than factor 2.
Equal: (58-59) Factor 1 equals factor 2.

You must specify at least one resulting indicator in positions 54 through 59. 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 193.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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                  141516
C*
C*  Indicator 19 is set on; indicator 17 is set off.
C           FLDA      COMP FLDC                  171819
C*
C*  Indicator 21 is set on; indicators 20 and 22 are set off
C           FLDD      COMP FLDE                  202122
```

Figure 53. COMP Operation

DEBUG (Debug Function)

Code	Factor 1	Factor 2	Result Field	Indicators
DEBUG	Identifier	Output file	Debug info	

The DEBUG operation helps the programmer debug a program that is not working properly. One or more records containing information helpful for finding programming errors are written as a result of this operation. You can use the DEBUG operation independently of or in combination with the OS/400 testing and debugging functions.

You can specify the operation at any point or at several points in the calculation specifications. Whenever the program encounters the DEBUG operation, one or more records are written. The first record contains the source-statement sequence number of the DEBUG operation in the program, factor 1 if any, and a list of all indicators that were on at the time the DEBUG operation was encountered. It can contain the contents of a field, an array element, a table element, or a literal that identifies the written information that describes the DEBUG operation. The contents of factor 1 identify the written information that describes the DEBUG operation. It can contain a field name, an array element, a table name, a named constant, or a literal. The length of the specified field can be from 1 to 8 characters. Factor 1 cannot contain a figurative constant. If factor 1 contains an entry, the sequence number and the contents of the entry are written to the first record. If factor 1 is not used, the source statement sequence number of the DEBUG operation is written to the first record.

Factor 2 can contain the name of the output file to which the DEBUG output is written. This file must have a record length of at least 80 positions. Only program-defined output files are allowed with the DEBUG operation; externally described files are not permitted. If factor 2 is blank, the output goes to the display work station that requested the program (the requester). The same entry must appear in factor 2 of all DEBUG operations in a program.

The contents of the result field are written to a separate record. The result field can contain a field name, an array name, a table name, or a “KLIST (Define a Composite Key)” name.

If factor 1 and the result field are not specified, only the indicators and the source statement sequence number of the DEBUG operation are written.

Note: If a KLIST is specified in the result field of a DEBUG operation, all numeric fields in the KLIST are printed or displayed in zoned decimal format.

The DEBUG operation is performed only if a 1 is specified in position 15 of the control specification. Otherwise, the DEBUG statement is checked for errors at compile time, but the DEBUG operation is not processed at run time.

Records Written for DEBUG

For a DEBUG operation, one record is always written in the following format:

Positions	Information
1-8	DEBUG=

9-16	Source statement sequence number of the DEBUG operation code in the program. This entry permits you to identify the individual DEBUG operation (if more than one is used) without making an entry in factor 1.
17-18	Blank.
19-26	Contents of factor 1, if specified.
27	Minus (-) sign if factor 1 contains a negative value. Blank if factor 1 contains a positive value.
28	Blank.
29-43	The words INDICATORS ON=
44	Blank.
45-47, ...	The names of all indicators that are on, each separated by a blank. If more than one record is needed to write all the indicators, positions 1 through 43 are used only by the first record. The indicators are written starting in position 45 of the remaining records. **Note:** If an indicator contains a character that is not valid (not '0' or '1'), the indicator is listed followed by the hexadecimal representation of the value in the indicator. For example, if indicators 01 and 88 are on, and indicator 33 contains the character A, the indicator line appears as: ON = 01 33 (C1) 88.

When the result field contains an entry, a separate record is written in the following format:

Positions	Information
1-14	The words FIELD VALUE=
15- (any position)	The contents of the result field. The first 66 characters of the result field are written in positions 15 through 80 of this record. If the field is greater than 66 characters, the additional characters are written in positions 15 through 80 of additional records.

Numeric fields are written out in unpacked format and are zero-suppressed. The sign is always written to the right of the field; a minus (-) sign is written to the right of negative fields, and a blank is written to the right of positive fields. Zero fields are written with the last zero and the sign. No other editing is done. If the result field is an array name, the elements are written in order, one record for each element. When a multiple occurrence data structure is specified, only the current occurrence is written.

DEFN (Field Definition)

Code	Factor 1	Factor 2	Result Field	Indicators
DEFN	*LIKE	Referenced field	Defined field	
DEFN	*NAMVAR	External data area	Internal program area	

Depending on the factor 1 entry, the declarative DEFN 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 DEFN 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 17) are not permitted.

*LIKE DEFN

The "DEFN (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 49 through 51 (field length) to make the result field entry longer or shorter than the factor 2 entry. A plus sign (+) in position 49 indicates a length increase; a minus sign (-) in position 49 indicates a length decrease. Positions 50 and 51 can contain the increase or decrease in length (right-adjusted) or can be blank. If positions 49 through 51 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.

See Figure 54 on page 241 for examples of *LIKE DEFN.

*NAMVAR DEFN

The "DEFN (Field Definition)" operation with *NAMVAR in factor 1 associates a field, a data structure, a data-structure subfield, or a data-area data structure (within your RPG/400 program) with an AS/400 data area (outside your RPG/400 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/400 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/400 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 file, 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 appears on a *NAMVAR DEFN statement.

In positions 49 through 52, 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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
*
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     DEFN FLDA      FLDP
C*
C*  FLDQ is a 9-position character field.
C           *LIKE     DEFN FLDA      FLDQ  + 2
C*
C*  FLDR is a 6-position character field.
C           *LIKE     DEFN FLDA      FLDR  - 1
C*
C*  FLDS is a 5-position numeric field with 2 decimal positions.
C           *LIKE     DEFN FLDB      FLDS
C*
C*  FLDT is a 6-position numeric field with 2 decimal positions.
C           *LIKE     DEFN FLDB      FLDT  + 1
C*
C*  FLDU is a 3-position numeric field with 2 decimal positions.
C           *LIKE     DEFN FLDB      FLDU  - 2
C*
C*  FLDX is a 3-position numeric field with 2 decimal positions.
C           *LIKE     DEFN FLDU      FLDX
```

Figure 54 (Part 1 of 2). DEFN Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  If specified, 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           *NAMVAR   DEFN           TOTGRS 102
C*
C*  The result field entry (TOTNET) is the name of the data area to
C*  be used within the RPG/400 program.  The factor 2 entry (TOTAL)
C*  is the name of the data area as defined to the system.
C           *NAMVAR   DEFN TOTAL     TOTNET
C*
C*  The result field entry (SAVTOT) is the name of the data area to
C*  be used within the RPG/400 program.  The factor 2 entry (*LDA)
C*  indicates the use of the local data area.
C           *NAMVAR   DEFN *LDA      SAVTOT
```

Figure 54 (Part 2 of 2). DEFN Operation

DELET (Delete Record)

Code	Factor 1	Factor 2	Result Field	Indicators
DELET	Search argument	File name		NR ER _

The DELET operation deletes a record from a database file. The file must be an update file (identified by a U in position 15. of the file description specifications) The deleted record can never be retrieved.

If factor 1 contains no entry, the DELET 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 an integer literal or a numeric field 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 54 and 55. 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 56 and 57; it is set on if the DELET operation is not completed successfully. (For example, an unauthorized user tries to delete the record) Leave positions 58 and 59 blank.

Under the OS/400 operating system, if a read operation is done on the file specified in factor 2 after a successful DELET operation to that file, the next record after the deleted record is obtained.

DIV (Divide)

Code	Factor 1	Factor 2	Result Field	Indicators
DIV(½)	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/400 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 189.

Figure 33 on page 191 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 201.

In factor 1, specify a starting value with no 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 no 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.

You have the option of indenting DO statements, IF-ELSE clauses, and SELEC-WHxx-OTHER clauses for readability in the compiler listing. See the section on Structured Programming in the *RPG/400 User's Guide* for an explanation of how to indent statements in the compiler listing. Figure 55 shows how the INDENT compiler option can make the compiler printout more readable. In this example, a vertical bar is used for indentation. Note that underscore characters are used to show the resulting indicators that are missing from the compiler printout.

```
1....+....2....+..<      28 - 32      >..+....4....+....5....*.
                  MOVE                5          AMOUNT  50
                  MOVE                100        TOTAL   50
                  MOVE                1          FACTOR  20
         AMOUNT   DOWLE               TOTAL
                  | Z-ADD             1          COUNT   50
         COUNT    | DOUEQ             10
         AMOUNT   | | IFEQ            10
                  | | | ADD           1          FACTOR
         AMOUNT   | | | MULT          FACTOR     AMOUNT
         COUNT    | | | IFEQ          2
                  | | | | Z-ADD       2          FACTOR
                  | | | | SETON                           __02__
                  | | | ELSE
                  | | | | Z-ADD       4          FACTOR
                  | | | | SETON                           __03__
                  | | | ENDIF
                  | | ENDIF
                  | ENDDO
                  | SETON                                  __04__
                  ENDDO
```

Figure 55. Source Listing Indentation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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   17               DO   10        X       30       DO 10 TIMES
C                     :
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       30       DO 10 TIMES
C                     :
C                     :
C                     :
C   50               ENDDO  2
```

Figure 56. DO Operation

DOUxx (Do Until)

Code	Factor 1	Factor 2	Result Field	Indicators
DOUxx	Comparand	Comparand		

The DOUxx operation begins a group of operations you want to process more than once (but always at least 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 201.

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 either both character or both numeric.

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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  The DOUEQ operation runs the operation within the DO group at
C*  least once.
C           FLDA      DOUEQFLDB
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                     SUB  1         FLDA
C                     ENDDO
C*
C*  The combined DOUEQ ANDEQ OREQ operation processes the operation
C*  within the DO group at least once.
C           FLDA      DOUEQFLDB
C           FLDC      ANDEQFLDD
C           FLDE      OREQ 100
```

Figure 57 (Part 1 of 2). DOUxx Operations

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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                     SUB  1         FLDA
C                     ADD  1         FLDC
C                     ADD  5         FLDE
C                     ENDDO
```

Figure 57 (Part 2 of 2). DOUxx Operations

DOWxx (Do While)

Code	Factor 1	Factor 2	Result Field	Indicators
DOWxx	Comparand	Comparand		

The DOWxx operation begins a group of operations you want to process while the relationship xx exists between factor 1 and factor 2. 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 201.

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 either both character or both 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 193.

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 DO 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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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           FLDA      DOWLTFLDB
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                     MULT 2.08      FLDA
C                     ENDDO
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           FLDA      DOWLTFLDB
C           FLDA      ORLT FLDC
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                     MULT 2.08      FLDA
C                     ENDDO
```

Figure 58. DOWxx Operations

DSPLY (Display Function)

Code	Factor 1	Factor 2	Result Field	Indicators
DSPLY	Message identifier	Output queue	Response	_ ER _

The DSPLY operation allows the program to communicate with the display work station that requested the program. The operation can display a message and accept a response.

The value in factor 1 is 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. QUSERMSG must be in a library in the library list of the job receiving the message.

The message identifier can be from 1 to 7 characters in length and can include an optional alphabetic prefix of 3 characters followed immediately by 1 to 4 digits. If the alphabetic prefix is not specified, the default is USR. The numeric portion of the message identifier (1) must immediately follow either *M or the optional alphabetic prefix, (2) cannot include embedded blanks, and (3) must be left-adjusted (high-order zeros can be omitted). If the numeric portion of the message identifier is not specified, the default is 0000. Valid examples are:

Factor 1 Entry	Message Identifier Used
*M	USR0000
*MABC	ABC0000
*M1	USR0001
*MABC5	ABC0005

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 specified in factor 2. (For information on how to create a queue, see the *CL Programmer's Guide*.) 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 requester.

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 56 and 57 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 56 and 57, if specified, is set on. If this indicator is not specified, the exception/error handling routine receives control.

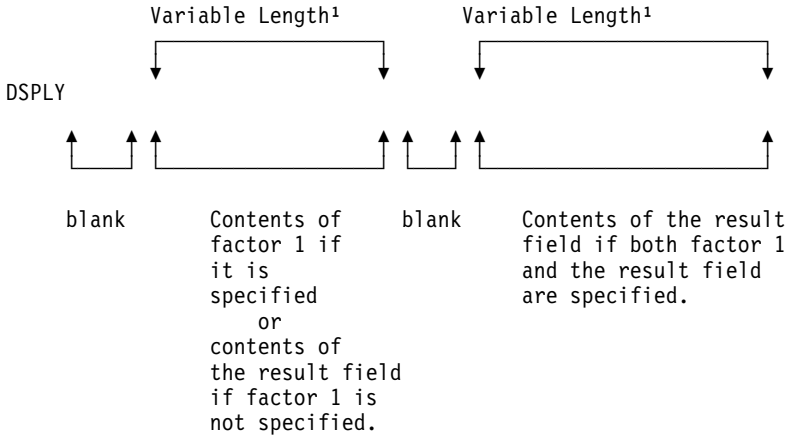
When you specify the DSPLY operation *with no message identifier in factor 1*, the operation functions as follows:

- 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.

  The maximum length of information that can be displayed is 52 characters.

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 characters.*



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 Guide*.

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 keyed 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.
- 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 keyed.
- Numeric fields are right-adjusted and padded on the left with zeros after all characters are keyed.
- Lowercase characters are not converted to uppercase.

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 of or in combination with the OS/400 testing and debugging functions.

The contents of factor 1 identify the DUMP operation. It must contain a character entry that can be one of: a field name, literal, named constant, table name, or array element whose contents identify the dump. 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 a 1 is specified in position 15 of the control specification. If the control specification entry is not made, the DUMP operation is checked for errors and the statement is printed on the listing, but the DUMP operation is not processed.

If you have specified a POST operation code, with factor 1 blank, anywhere in your program, no file information data structures (INFDS) are updated until you do an appropriate POST operation. For up-to-date information, do a POST operation for each file before you do the DUMP operation. (This action is not required for a dump that the system does in response to an inquiry message.)

ELSE (Else)

Code	Factor 1	Factor 2	Result Field	Indicators
ELSE				

The ELSE operation is an optional part of the IFxx operation. 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 17) are not permitted.

To close the IFxx/ELSE group use an ENDIF operation.

Figure 63 on page 274 shows an example of an ELSE operation with an IFxx operation.

ENDyy (End a 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, DOUxx, DOWxx, IFxx, or SELEC group of operations.

The ENDyy operations are listed below:

END End a CASxx, DO, DOUxx, DOWxx, IFxx, or SELEC group
ENDCS End a CASxx group
ENDDO End a DO, DOUxx, or DOWxx group
ENDIF End an IFxx group
ENDSL End a SELEC 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 no 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.

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 SELEC 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/400 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 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/400 error handler receives control.

See “File Exception/Error Subroutine (INFSR)” on page 37 for more detail on return points.

See Figure 60 on page 265 for an example of coding an RPG/400 subroutine.

EXCPT (Calculation Time Output)

Code	Factor 1	Factor 2	Result Field	Indicators
EXCPT		EXCPT name		

The EXCPT operation has two major functions:

- It allows records to be written during calculation time.
- It allows a variable number of records to be written in one program cycle at either detail calculation or total calculation time.

See Figure 59 on page 261 for examples of the EXCPT operation.

When specifying the EXCPT operation remember:

- The exception records that are to be written during calculation time are indicated by an E in position 15 of the output specifications. An EXCPT name, which is the same name as specified in factor 2 of an EXCPT operation, can be specified in positions 32 through 37 of the output specifications of the exception records.
- Only exception records, not heading, detail, or total records, can contain an EXCPT name.
- When the EXCPT operation with a name in factor 2 is processed, only those exception records with the same EXCPT 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 32 through 37 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/400 cycle or during fetch overflow. The record is not written at the time the EXCPT 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 UNLCK operation.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  When the EXCPT operation with HDG specified in factor 2 is
C*  processed, all exception records with the EXCPT name HDG are
C*  written.  In this example, UDATE and PAGE would be printed
C*  and then the printer would space 2 lines.
C                   EXCPTHDG
C*
C*  When the EXCPT operation with no entry in factor 2 is
C*  processed, all exception records that do not have an EXCPT
C*  name specified in positions 32 through 37 are written if the
C*  conditioning indicators are satisfied.  Any exception records
C*  without conditioning indicators and without an EXCPT name
C*  are always written by an EXCPT 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                   EXCPT
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
OName++++DFBASbSaN01N02N03Excnam.................................
O*
O       E  1     10
O                          TITLE
O                          AUTH
O       E  2               HDG
O                          UDATE
O                          PAGE
O       E  1               DETAIL
O                          AUTH
O                          VERSNO
```

Figure 59. EXCPT 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 16 of the file description specifications) combined file (C in position 15 of the file description specifications) that is externally described (E in position 19. 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 56 and 57 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 54, 55, 58, and 59 must be blank.

For the use of EXFMT with multiple device files, see the descriptions of the READ (by format name) and WRITE operations.

EXSR (Invoke Subroutine)

Code	Factor 1	Factor 2	Result Field	Indicators
EXSR		Subroutine name		

The EXSR operation causes the RPG/400 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/400 subroutine can be processed from any point in the calculation operations. All RPG/400 operations can be processed within a subroutine, and these operations can be conditioned by any valid indicators in positions 9 through 17. 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.

You can include a maximum of 254 subroutines in a program; however, 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 can 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 outside the subroutine cannot be issued to a BEGSR, ENDSR, or TAG within a subroutine. A GOTO within a subroutine can be issued to a TAG within either detail or total calculations.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C* For a subroutine, positions 7 and 8 can be blank or contain SR.
C*
C                         :
C                         :
C                     EXSR SUBRTB
C                         :
C                         :      CALCULATIONS
C                         :
CL2                   EXSR SUBRTA
C                         :
C                         :      CALCULATIONS
C                         :
C           SUBRTA    BEGSR
C                         :
C                         :      CALCULATIONS
C                         :
C*
C*  One subroutine can call another subroutine.
C*
C                     EXSR SUBRTC
C                         :
C                         :      CALCULATIONS
C                         :
C                     ENDSR
C           SUBRTB    BEGSR
C                         :      CALCULATIONS
C                         :
C                         :
C*
```

Figure 60 (Part 1 of 2). Example of Coding Subroutines

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  GOTO and TAG operations can be used within a subroutine.
C*
C           START     TAG
C                       :
C                       :      CALCULATIONS
C                       :
C   23                  GOTO END
C                       :
C                       :      CALCULATIONS
C                       :
C   24                  GOTO START
C           END         ENDSR
C           SUBRTC      BEGSR
C                       :
C                       :      CALCULATIONS
C                       :
C                       ENDSR
C*
```

Figure 60 (Part 2 of 2). Example of Coding Subroutines

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 *Database Guide*.)

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 56 and 57 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 4 on page 14.

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.

FREE (Deactivate a Program)

Code	Factor 1	Factor 2	Result Field	Indicators
FREE		Program name		_ ER _

The FREE operation removes a program from the list of activated programs, frees static storage, and ensures program initialization (first cycle processing) the next time the program is called. It does not close files or unlock data areas.

Factor 2 contains the name of the program to be deactivated. It must contain the name of a field, named constant, literal, or array element that contains the name of the program to be deactivated. The entry in factor 2 must be character data; it can include a qualified name such as LIB/PGM. The name preceding the slash is the library that contains the program to be freed from the activated program list. Specify only the program name if you want to search the library list. The RPG/400 language uses the program name exactly as specified in the literal, field, or array element to determine the program to be called. (Lowercase characters are not shifted to uppercase, and a name enclosed in quotation marks (for example 'ABC') always includes the quotation marks as part of the name of the program to be freed.) *LIBL and *CURLIB are not supported (for example, '*LIBL/PROG').

You can specify any valid resulting indicator in positions 56 and 57 to be set on if FREE is not completed successfully. No error occurs if the program to be freed is not active (for example, the program does not exist).

For programs that are to run under System/38 Environment, if you specify the library name, the program name must be immediately followed by a period and then the library name.

See the “CALL (Call a Program)” operation for details on how program references are grouped.

Note: Issuing a FREE operation and then a CALL operation to the same program reopens the program's files and may use additional temporary storage. Repeatedly issuing FREE and CALL operations to a program, without closing the program's files in between, may use enough temporary storage to degrade the system's performance, and ultimately cause an AS/400 machine check. This problem can be avoided if the files have shared open data paths SHARE(*YES). However, using SHARE(*YES) can cause other problems. Read the section on "Sharing an Open Data Path" in the *RPG/400 User’s Guide* for complete details.

Figure 61 on page 270 shows the FREE operation being used with the CALL operation.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  When the CALL operation is processed, the data in the result
C*  fields of the parameter list can be accessed by PROGA.  The
C*  parameter list ends when the first calculation operation other
C*  than a PARM operation is encountered.
C                   CALL 'PROGA'
C                   PARM          FLDA    30
C                   PARM          FLDB    30
C                   PARM          FLDC    50
C*
C*  When the FREE operation is processed, PROGA is removed from the
C*  list of activated programs.  Removing it from the list ensures
C*  a fresh copy of all fields in PROGA the next time the program is
C*  called.  Indicator 55 is set on if the FREE operation is not
C*  completed successfully.
C*
C                   FREE 'PROGA'               55  55 = NO SUCCESS
```

Figure 61. CALL/FREE Operations

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 (Tag)" operation names the destination of a GOTO operation. Use a GOTO operation to 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 to a total calculation line.

A GOTO operation outside a subroutine cannot specify a branch to a TAG or ENDSR operation within that subroutine.

A branch cannot be made from a total calculation line to a detail calculation line. (A total calculation line is defined as one that is conditioned by a control level indicator in positions 7 and 8 of the calculation specifications.)

Branching from one part of the RPG/400 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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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                      :         Calculations
C                      :
C
C   20                GOTO END
C*
C                      :
C                      :         Calculations
C                      :
C           END       TAG
C*  A branch within total calculations.
CL1                   GOTO RTN2
CL1
CL1         RTN2      TAG
```

Figure 62. GOTO and TAG 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 201.)

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 character, or both must be numeric.

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 SELEC-WHxx-OTHER clauses for readability. See also the section on "Structured Programming" in the *RPG/400 User's Guide* for an explanation of how to indent statements in the source listing.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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           FLDA      IFEQ FLDB                       IF EQUAL
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                       IF EQUAL
C                       :
C                       :
C                       :
C                     ELSE                            IF NOT EQUAL
C                       :
C                       :
C                       :
C                     ENDIF
```

Figure 63 (Part 1 of 2). IFxx/ENDIF and IFxx/ELSE/ENDIF Operations

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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           FLDA      IFEQ FLDB
C           FLDA      ANDGTFLDC
C           FLDD      OREQ FLDE
C           FLDD      ANDGTFLDF
C                       :
C                       :
C                       :
C                     ENDIF
```

Figure 63 (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 *NAMVAR DEFN statement. (See “DEFN (Field Definition)” on page 239 for information on *NAMVAR DEFN.)

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 “UNLCK (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/400 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 *NAMVAR. When *NAMVAR 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/400 exception/error handling routine receives control. If a program exception/error subroutine (*PSSR) is specified, the program status data structure contains information on the data area in error. If a message is issued to the requester, the message identifies the data area in error.

You can specify a resulting indicator in positions 56 and 57 to be set on if an error occurs during the operation. Positions 54-55 and 58-59 must be blank.

For further rules for the IN operation, see “Data-Area Operations” on page 194.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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   *NAMVAR
C*
C                     ADD  AMOUNT    TOTAMT
C                     ADD  GROSS     TOTGRS
C                     ADD  NET       TOTNET
C*
CLR                   OUT  *NAMVAR
C*
C           *NAMVAR   DEFN           TOTAMT  82
C           *NAMVAR   DEFN           TOTGRS 102
C           *NAMVAR   DEFN           TOTNET 102
```

Figure 64. 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, DOUxx, 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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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           FLDA      DOUEQFLDB                     Outer loop
C                     :
C           NUM       DOWLT10                       Inner loop
C           *IN01     IFEQ *ON
C                     LEAVE
C                     ENDIF
C                     EXSR PROC1
C           *IN12     IFEQ *OFF
C                     ITER                          ITER
C                     ENDIF
C                     EXSR PROC2
C                     ENDDO                         Inner ENDDO
C                     Z-ADD20         RSLT    20    Z-ADD
C                     :
C                     ENDDO                         Outer ENDDO
C                     :
C*
```

Figure 65 (Part 1 of 2). ITER Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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* control to the innermost ENDDO and the condition on the DOW is
C* evaluated again.
C                     :
C           FLDA      DOUEQFLDB
C                     :
C           NUM       DOWLT10
C           *IN01     IFEQ *ON
C                     MOVE 'UPDATE'  FIELD  20
C                     LEAVE
C                     ELSE
C                     ITER
C                     ENDIF
C                     ENDDO                          Inner ENDDO
C                     Z-ADD20        RSLT    20
C                     :
C                     ENDDO
C                     :
```

Figure 65 (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 17) 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 (character or numeric), 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 66 on page 283 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 17) 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, DELET, READE, REDPE, 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*
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  The KLIST operation indicates the name, FILEKY, by which the
C*  search argument can be specified.
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.

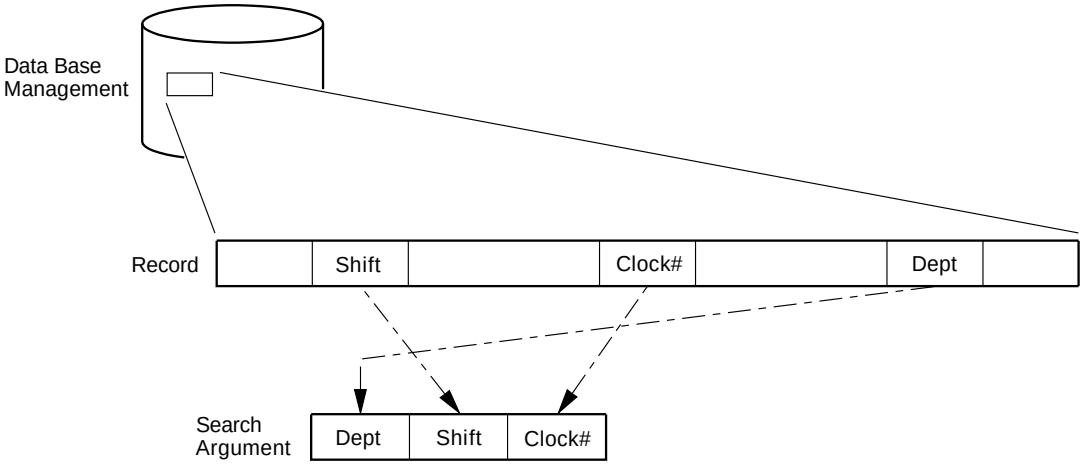


Figure 66. KLIST and KFLD Operations

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, DOUxx, 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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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         DOWNE1
C                      :
C           ANSWER    IFEQ 'q'
C                     LEAVE
C                     ENDIF
C                      :
C                     ENDDO
C                     Z-ADDA         B
C*
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           FLDA      DOUEQFLDB
C           NUM       DOWLT10
C           *IN01     IFEQ *ON
C                     SETON                     99
C                     LEAVE
C                       :
C                     ENDIF
C                     ENDDO
C  99                 LEAVE
C                       :
C                     ENDDO
C                       :
```

Figure 67. LEAVE Operation

LOKUP (Look Up)

Code	Factor 1	Factor 2	Result Field	Indicators
LOKUP				
(array)	Search argument	Array name		HI LO EQ
(table)	Search argument	Table name	Table name	HI LO EQ

The LOKUP 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 character or numeric literal, a field name, an array element, a table name, a named constant, or a figurative constant.

If a table is named in factor 1, the search argument used is the element of the table last selected in a LOKUP operation, or it is the first element of the table if a previous LOKUP has not been processed. The array or table to be searched is specified in factor 2.

For a table LOKUP, 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.

Decimal alignment is not processed for LOKUP operations.

Resulting indicators specify the search condition for LOKUP. One must be specified in positions 54 through 59 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 LOKUP operation searches for an entry that satisfies either condition with equal given priority.

High (54-55): 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 (56-57): 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 (58-59): 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 LOKUP operation, remember:

- The search argument and array element or table element must have the same length and the same format (character or numeric).
- When LOKUP is processed on an array and an index is used, the LOKUP 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 in the extension specifications. For further information on the sequence entry, see “Position 45 (Sequence)” on page 123.
- No resulting indicator is set on if the search is not successful.
- If only an equal indicator (positions 58-59) is used, the LOKUP 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 LOKUP operation can produce unexpected results when the array is not strictly in ascending or descending sequence.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  In this example, the programmer wants to know which element in
C*  ARY the LOKUP operation locates.  The Z-ADD operation sets the
C*  field X to 1.  The LOKUP 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                   Z-ADD1        X       30
C           SRCHWD    LOKUPARY,X                    26      EQUAL
C*
C*  In this example, the programmer wants to know if an element
C*  is found that is equal to SRCHWD.  LOKUP searches ARY until it
C*  finds the first element equal to SRCHWD.  When this occurs,
C*  indicator 26 is set on.
C           SRCHWD    LOKUPARY                      26      EQUAL
C*
C*  The LOKUP starts at a variable index number specified by field
C*  X.  Field X does not have to set to 1 before the LOKUP
C*  operation.  When LOKUP 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           SRCHWD    LOKUPARY,X                    26      EQUAL
```

Figure 68. LOKUP 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 199.

The function of the MHHZO operation is shown in Figure 34 on page 200.

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 199.

The function of the MHLZO operation is shown in Figure 34 on page 200.

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 199.

The function of the MLHZO operation is shown in Figure 34 on page 200.

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 199.

The function of the MLLZO operation is shown in Figure 34 on page 200.

MOVE (Move)

Code	Factor 1	Factor 2	Result Field	Indicators
MOVE (p)		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.

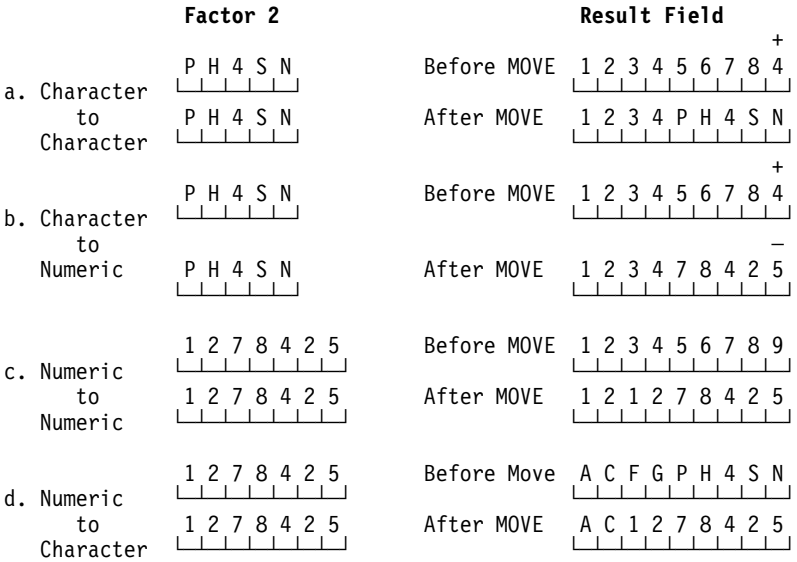
If factor 2 is longer than the result field, the excess leftmost characters of factor 2 are not moved. If the result field is longer than factor 2, the excess leftmost characters in the result field are unchanged, unless padding is specified.

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 (position 53) 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 198.

Factor 2 Shorter Than Result Field



Factor 2 Longer Than Result Field

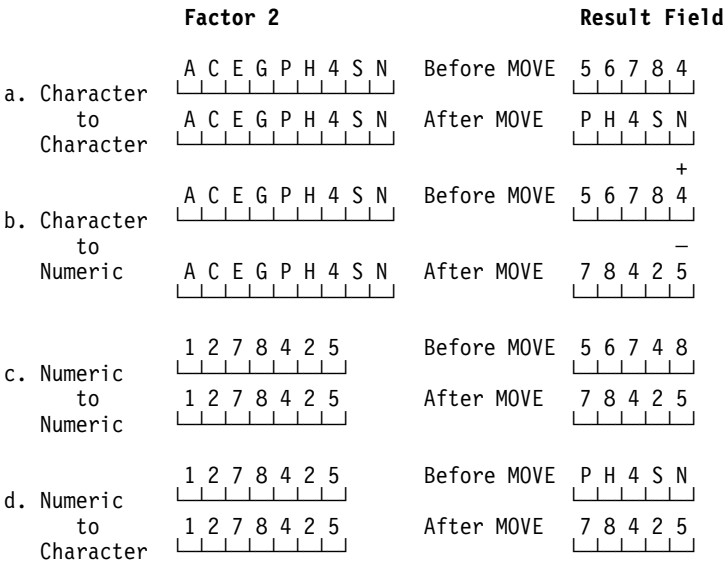
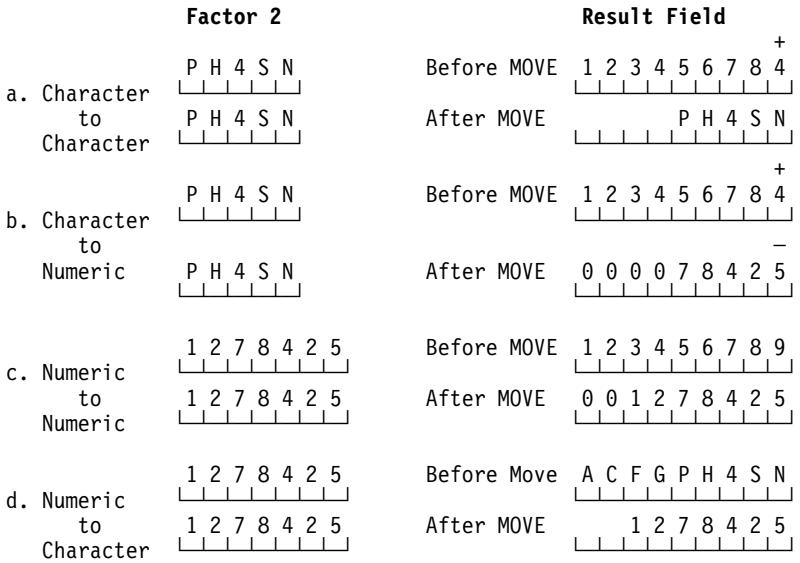
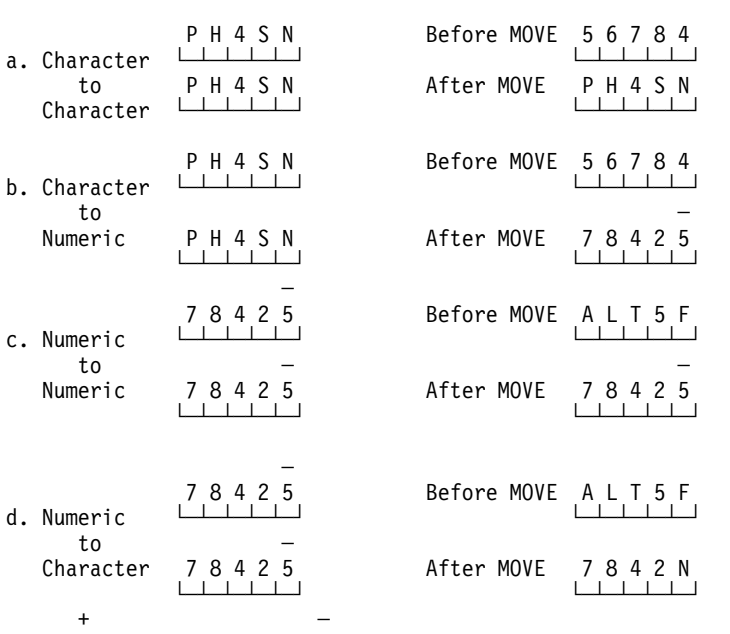


Figure 69 (Part 1 of 2). MOVE Operation

Factor 2 Shorter Than Result Field
With P in Operation Extender Field



Factor 2 and Result Field Same Length



Note: 4 = letter D , and 5 = letter N.

Figure 69 (Part 2 of 2). MOVE Operation

MOVEA (Move Array)

Code	Factor 1	Factor 2	Result Field	Indicators
MOVEA (p)		Source	Target	+ – ZB

The MOVEA operation transfers character 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, or character array. You can:

- Move several contiguous character array elements to a single character field
- Move a single character field to several contiguous character 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 70 on page 296.

Character MOVEA Operations

Both factor 2 and the result field must be defined as character.

On a character 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 character 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. . .'.

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

For both character and numeric MOVEA operations, you can specify a P in position 53 to pad the result from the right.

For further information on the MOVEA operation, see "Move Operations" on page 198.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C                     MOVEAARRX      ARRY
C*  Array-to-array move.  No indexing; different length array,
C*  same element length.
```

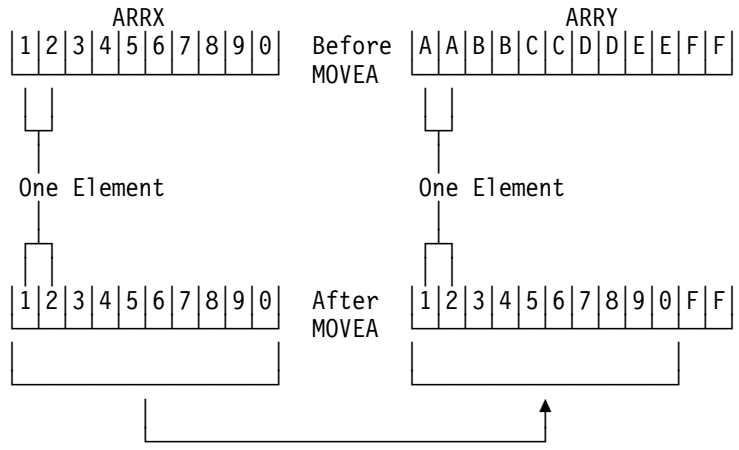
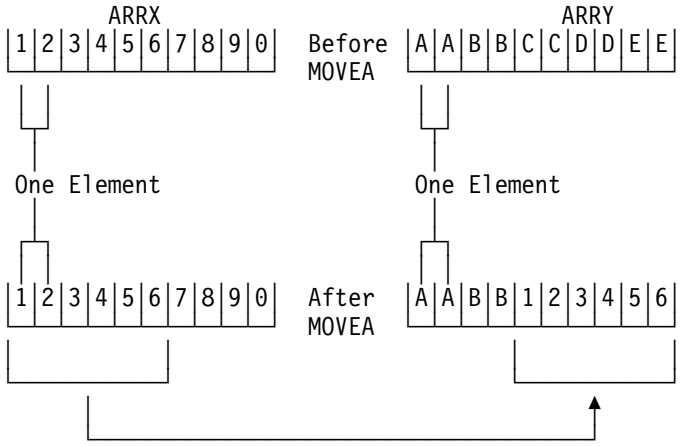


Figure 70 (Part 1 of 6). MOVEA Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C                     MOVEAARRX     ARRY,3
C*  Array-to-array move with index result field.
```




```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C                     MOVEAARRX     ARRY
C*  Array-to-array move, no indexing and different length array
C*  elements.
```

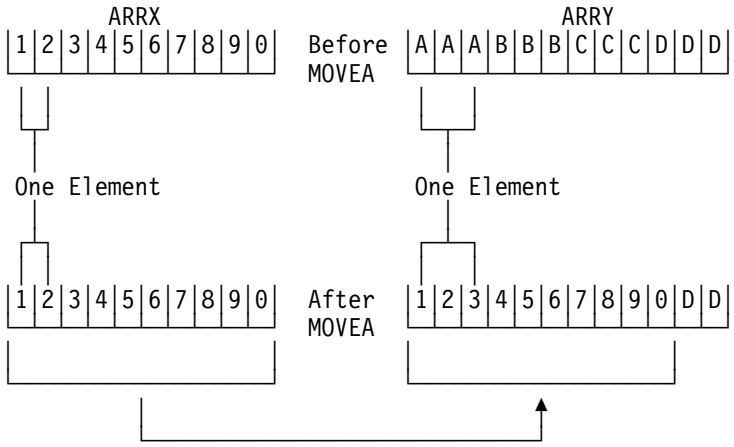
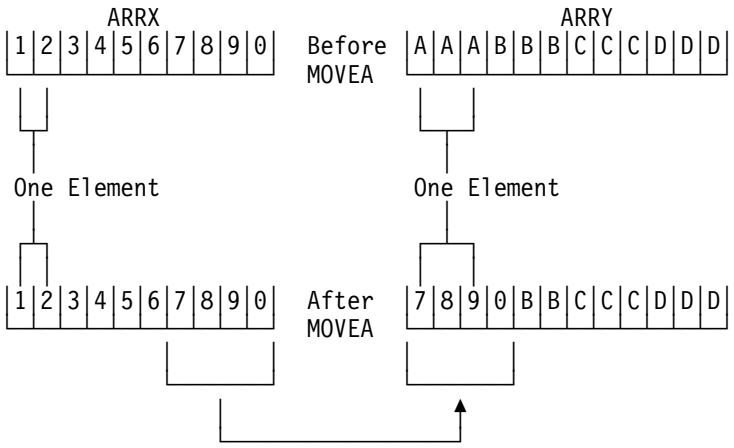


Figure 70 (Part 2 of 6). MOVEA Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C                     MOVEAARRX,4    ARRY
C*  Array-to-array move, index factor 2 with different length array
C*  elements.
```




```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C                     MOVEAFIELDA    ARRY
C*  Field-to-array move, no indexing on array.
```

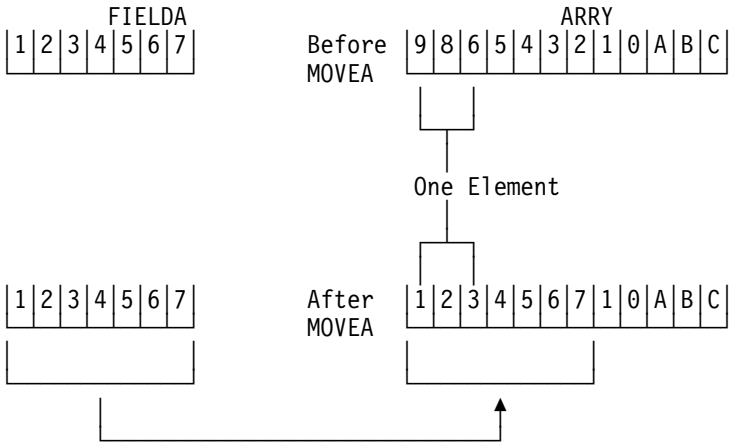
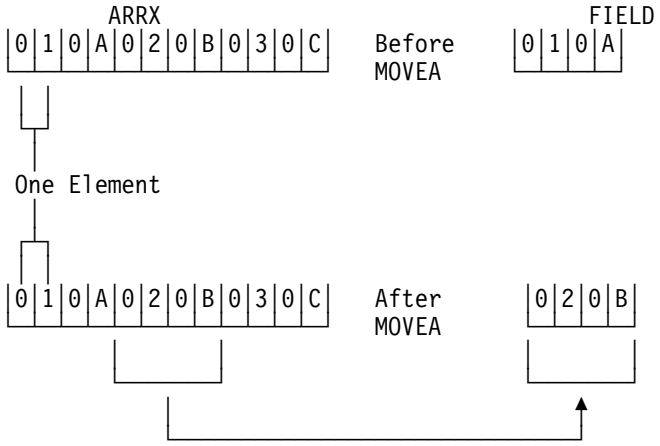


Figure 70 (Part 3 of 6). MOVEA Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C* In the following example, N=3.  Array-to-field move with variable
C* indexing.
C                     MOVEAARRX,N    FIELD
C*
```




```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C                     MOVEAARRB      ARRZ
C*
C* An array-to-array move showing numeric elements.
```

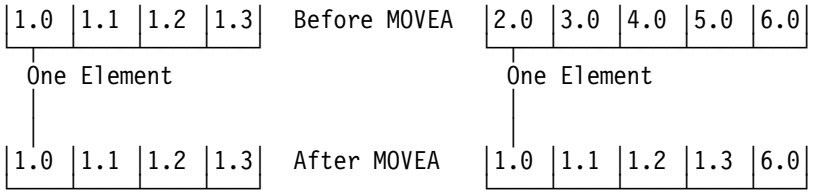
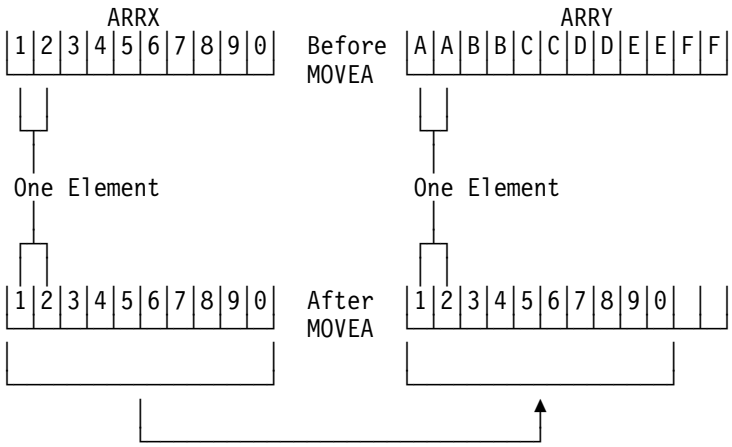


Figure 70 (Part 4 of 6). MOVEA Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C                     MOVEAARRX     ARRY      P
C*  Array-to-array move.  No indexing; different length array with
C*  same element length.
```




```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C                     MOVEAARRB     ARRZ      P
C*
C* An array-to-array move showing numeric elements with padding.
```

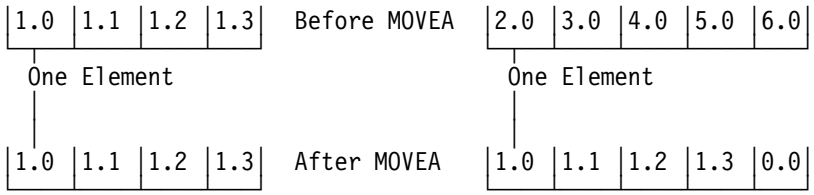


Figure 70 (Part 5 of 6). MOVEA Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C                   MOVEAARRX,3    ARRY      P
C*  Array-to-array move.  No indexing; different length array with
C*  same element length.
```

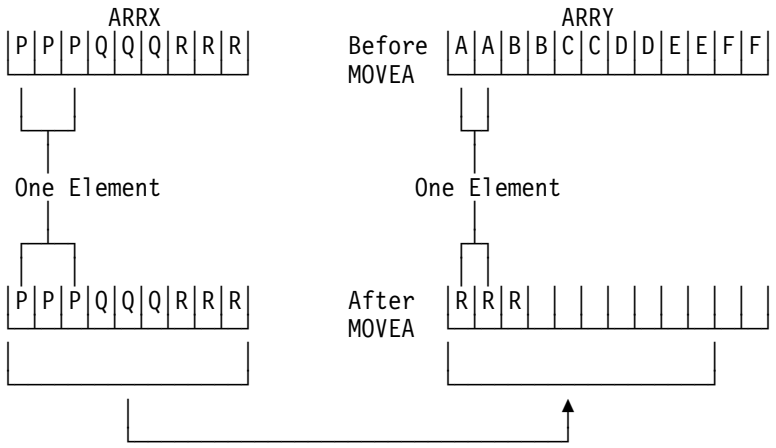


Figure 70 (Part 6 of 6). MOVEA Operation

MOVEL (Move Left)

Code	Factor 1	Factor 2	Result Field	Indicators
MOVEL (p)		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.

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 71 on page 304.

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 with the rightmost position.
   b. If factor 2 is numeric and the result field is character, the sign is moved with 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.
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.

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.
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 198 for more information on the rules for padding.

For further information on the MOVEL operation, see “Move Operations” on page 198.

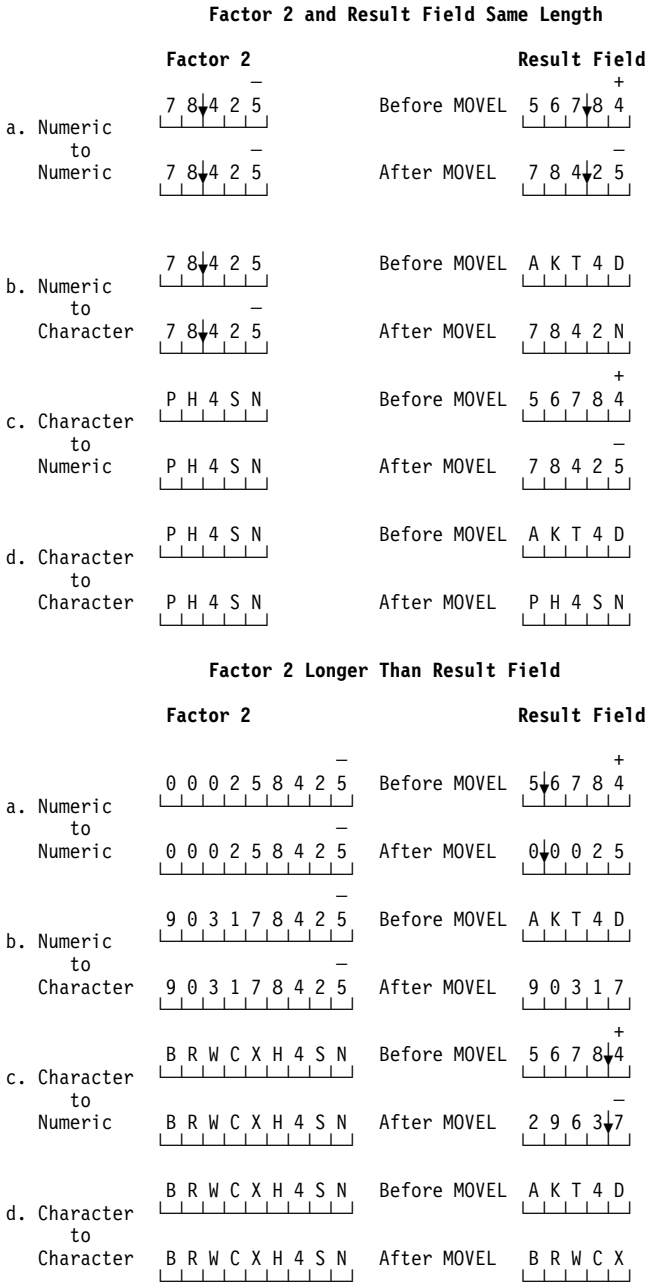
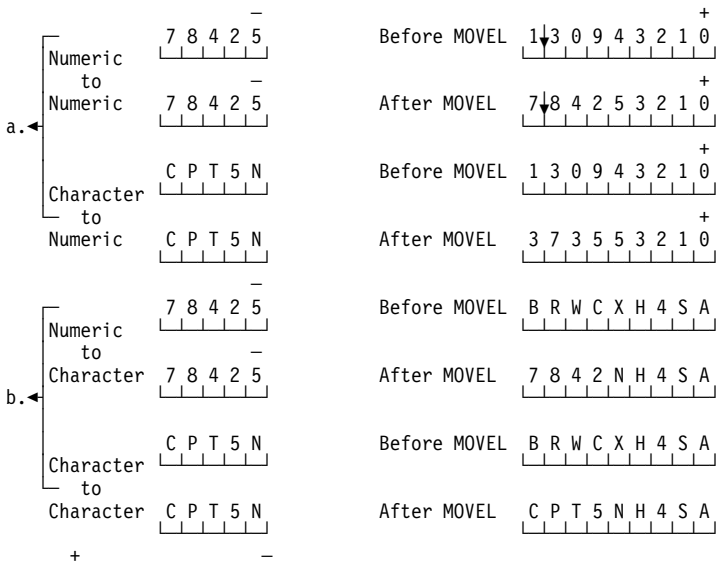


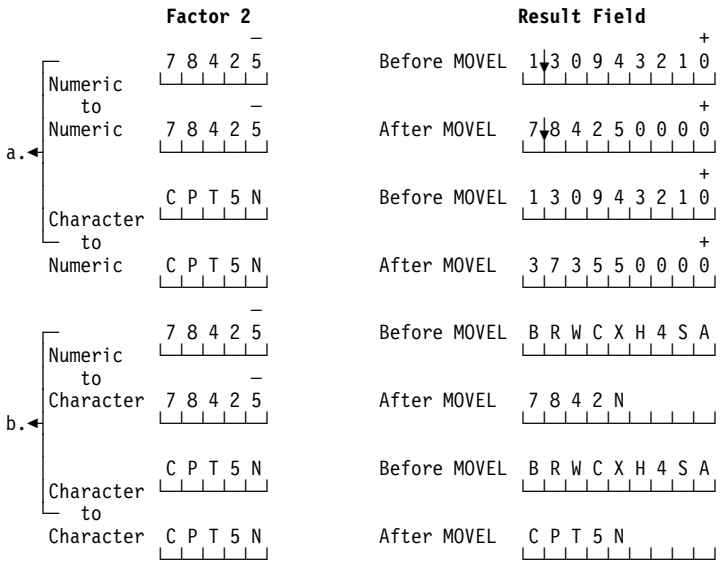
Figure 71 (Part 1 of 2). MOVEL Operation

Factor 2 Shorter Than Result Field



Note: 4 = letter D, and 5 = letter N; arrow ↓ is decimal point.

Factor 2 Shorter Than Result Field With P in Operation Extender Field



Note: 4 = letter D, and 5 = letter N; arrow ↓ is decimal point.

Figure 71 (Part 2 of 2). MOVEL Operation

MULT (Multiply)

Code	Factor 1	Factor 2	Result Field	Indicators
MULT (½)	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. You can specify half adjust (position 53).

For further information on the MULT operation, see “Arithmetic Operations” on page 189.

See Figure 33 on page 191 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 (position 53) 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 189.

See Figure 33 on page 191 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 56 and 57. It is set on if an exception/error occurs on the NEXT operation. If the INFSR subroutine is specified and positions 56 and 57 do not contain an indicator, the subroutine automatically receives control when an exception/error occurs. If the INFSR subroutine is not specified and positions 56 and 57 do not contain an indicator, the default error handler takes control.

OCUR (Set/Get Occurrence of a Data Structure)

Code	Factor 1	Factor 2	Result Field	Indicators
OCUR	Occurrence value	Data structure	Occurrence value	_ ER _

The OCUR operation code specifies the occurrence of the data structure that is to be used next within an RPG/400 program. 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 OCUR operation is specified. After an OCUR operation is specified, the occurrence of the data structure that was established by the OCUR 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 OCUR 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 OCUR operation.

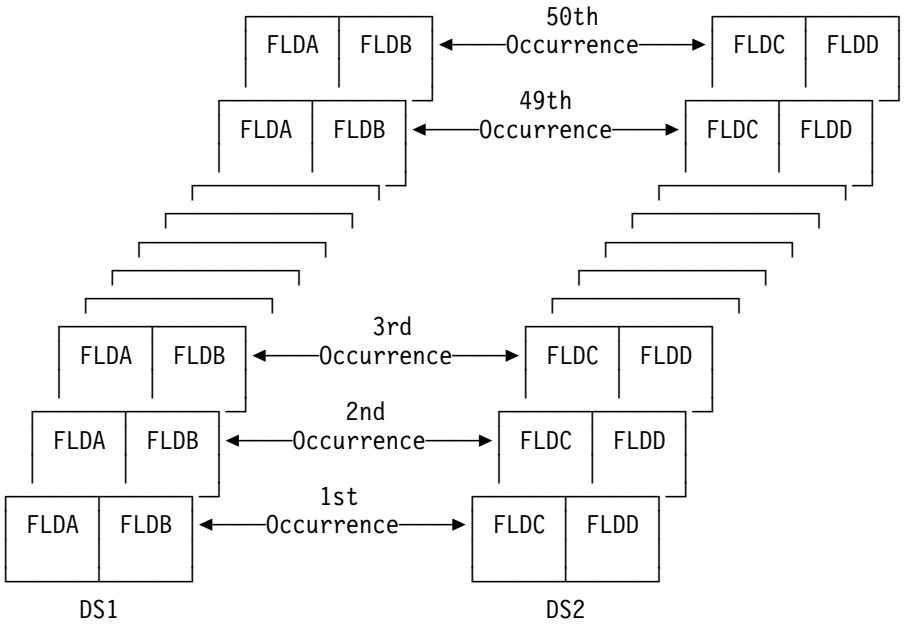
If factor 1 is a data structure name, it must be a multiple occurrence data structure. The current occurrence of the data structure in factor 1 is used to set the occurrence of the data structure in factor 2.

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 OCUR 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.

You can specify a resulting indicator in positions 56 and 57 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 OCUR operation was processed.

The OCUR 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.




```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
I.....................................PFromTo++DField+L1M1FrPlMnZr...
I*
I*  DS1 and DS2 are multiple occurrence data structures.
I*  Each data structure has 50 occurrences.
IDS1         DS                          50
I                                         1   5 FLDA
I                                         6   80FLDB
I*
IDS2         DS                          50
I                                         1   6 FLDC
I                                         7  11 FLDD
```

Figure 72 (Part 1 of 3). Uses of the OCUR Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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           3         OCUR DS1
C                     MOVE 'ABCDE'   FLDA
C                     Z-ADD22        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           4         OCUR DS1
C                     MOVE FLDA      FLDX
C                     Z-ADDFLDB      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         OCUR 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       OCUR 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                     OCUR DS1       Z
```

Figure 72 (Part 2 of 3). Uses of the OCUR Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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           DS2       OCUR 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           X         OCUR DS1                     20
C*
C           *IN20     IFEQ *ON
C                     SETON                     LR
C                     END
```

Figure 72 (Part 3 of 3). Uses of the OCUR Operation

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 56 and 57 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 UC (user control) in positions 71 and 72 of the file description specifications. (See Chapter 5, “File Description Specifications” for restrictions when using the UC entry.)

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 UC entry is not required in positions 71 and 72. When UC is not specified, 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 ID option specified (on the file description specifications continuation line), the ID field is set to blanks. See the description of the ID option, in Chapter 5, “File Description Specifications.”

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilenameIPEAF....RlenLK1AIOvKlocEDevice+......KExit++Entry+A....U..
FEXCPTN  O   E                    DISK                           UC
FFILEX   I   E                    DISK
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  The explicit OPEN operation opens the EXCPTN file for
C*  processing if indicator 97 is on and indicator 98 is off.
C*  Note that the EXCPTN file on the file description
C*  specifications has UC specified in positions 71 and 72.
C*
C   97N98             OPEN EXCPTN                99  99=NO SUCCESS
C   97N98N99          WRITEERREC
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 FILEX is opened
C*  at program initialization, UC is not specified in positions 71
C*  and 72 of the file description specifications.
C*
C                     CLOSEFILEX
C                     CALL 'RTNX'
C                     OPEN FILEX
```

Figure 73. 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, WHxx, and ANDxx operations. ORxx is specified immediately following a DOUxx, DOWxx, IFxx, WHxx, ANDxx or ORxx statement. Use ORxx to specify a more complex condition for the DOUxx, DOWxx, IFxx, and WHxx 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 WHxx operation. Conditioning indicator entries (positions 9 through 17) 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 either both character data or both numeric data. The comparison of factor 1 and factor 2 follows the same rules as those given for the compare operations. See “Compare Operations” on page 193.

Figure 57 on page 249 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 WHxx condition is satisfied in a SELEC 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 SELEC group.
- Only one OTHER operation can be specified in a SELEC group.
- No WHxx operation can be specified after an OTHER operation in the same SELEC 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 17) are not allowed.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C* Example of a SELEC group with WHxx 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                   SELEC
C           X       WHEQ 1
C                    :                              seq 1
C           Y       WHEQ 2
C                    :                              seq 2
C                   OTHER
C                    :                              seq 3
C                   ENDSL
```

Figure 74. OTHER Operation

For more details and examples, see the SELEC and WHxx 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 *NAMVAR DEFN statement.
- 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 18 of the input specifications. (The RPG/400 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 *NAMVAR. When *NAMVAR 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/400 exception/error handling routine receives control. If you specify a program exception/error subroutine (*PSSR), the program status data structure contains information on the data area in error. If a message is issued to the requester, the message identifies the data area in error.

You can specify a resulting indicator in positions 56 and 57 to be set on if an error occurs during the operation. Positions 54-55 and 58-59 must be blank.

For further rules for the OUT operation, see “Data-Area Operations” on page 194.

See Figure 64 on page 277 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 or CALL operation they refer to. PARM statements must be in the order expected by the called program. One PARM statement, or as many as 255 PARM statements, can follow a PLIST or CALL.

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 17) are not allowed.

Factor 1 and factor 2 entries are optional. If specified, the entries must be the same type (character or numeric) 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 field, data structure, or array that is to be the parameter. Also, the result field of a non-*ENTRY PLIST can contain an array element. The result field can be numeric or character. The result field cannot contain *IN, *INxx, *IN,xx, a label, a literal, 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, or a table name. In addition, an array element, a data-structure subfield name, the name of a compile-time array and the name of a program status or file information data structure (INFDS) are not allowed in the result field of PARM specified for an *ENTRY PLIST. 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. When a data structure with multiple occurrences is passed to the called program, 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.

Each parameter field has only one storage location; it is in the calling program. The address of the storage location of the result field is passed to the called program on a PARM operation. If the called program changes the value of a parameter, it changes the data at that storage location. When control returns to the calling program, the value of the parameter in the calling program (that is, the result field) has changed. Even if the called program ends in error after it changes the value of a parameter, the changed value exists in the calling program. To preserve the information passed to the called program for later use, specify in factor 2 the name of the field that contains the information you want to pass to the called program. Factor 2 is copied into the result field, and the storage address of the result field is passed to the called program.

Because the program accesses the parameter fields 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 should be the same. If they are not, undesirable results may occur.

When a CALL operation runs, the following occurs:

1. In the calling program, 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 called program, 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.
3. In the called program, when control is returned to the calling program, 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 program ends abnormally.
4. Upon return to the calling program, the contents of the result field of a PARM operation in the calling program are copied into the factor 1 field (receiver field) of the same PARM operation. This move does not occur if the called program ends abnormally or if an error occurs on the CALL operation.

Note: If the receiver field is numeric, the value of the field to be placed in the receiver field is added to a field of zeros and the sum is placed in the receiver field. If the receiver field is character, the field to be placed in the receiver field is moved (left-adjusted) into the receiver field, and the receiver field is padded with blanks. For a discussion of how to call and pass parameters to a program through CL, see the *CL Programmer's Guide*.

Figure 75 on page 320 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 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 17) are not allowed.

Factor 1 must contain the name of the parameter list. If the parameter list is the entry parameter list of a called program, factor 1 must contain *ENTRY. Only one *ENTRY parameter list can occur in a program. A parameter list is ended when an operation other than PARM is encountered.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  In the calling program, the CALL operation calls PROG2 and
C*  allows PROG2 to access the data in the parameter list fields.
C                    CALL 'PROG2'   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 PROG2 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           AMT     52
C           *IN30     PARM *IN27     BYTE    1
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C                    CALL 'PROG2'
C*  In this example, the PARM operations immediately follow a
C*  CALL operation instead of a PLIST operation.
C                    PARM           AMT     52
C           *IN30     PARM *IN27     BYTE    1
```

Figure 75 (Part 1 of 2). PLIST/PARM Operations

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*  In the called program, 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 (FLDC and FLDG) of the parameter list are placed in
C*  the factor 1 fields (FLDA and FLDD).  When the called program
C*  returns, the contents of the factor 2 fields of the parameter
C*  list (FLDB and FLDE) are placed in the result fields (FLDC
C*  and FLDG).  All of the fields are defined elsewhere in the
C*  program.
C           *ENTRY    PLIST
C           FLDA      PARM FLDB      FLDC
C           FLDD      PARM FLDE      FLDG
```

Figure 75 (Part 2 of 2). 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). The information is either on the status of a specific program device or I/O feedback associated with a file.

In factor 1, you can specify a program device name to get information about that specific program device. Use either a character field of length 10 or less, a character literal, or a character named constant. If you leave factor 1 blank, you get I/O 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 program device in factor 1, the file must be defined as a WORKSTN file.

If you specify a file in factor 2, you can leave the result field blank. The INFDS associated with this file 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 an INFDS in the result field; information from the file associated with this INFDS will be posted.

In positions 56 and 57, 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/400 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/400 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 (except for the area *STATUS which is always updated). To ensure that the DUMP operation code provides up-to-date information in this case, issue a POST operation before the DUMP operation.

If a program has no POST operation code, or if it has only POST operation codes with factor 1 specified, the INFDS is updated with each input/output operation or block of operations. If RPG is blocking records, the information in the INFDS will be valid for the last block of records processed. If you require more accurate information, do not use record blocking. See “File Information Data Structure” on page 25 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 16 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 19 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 19 of the file description specifications). See “File Operations” on page 196 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 (by a “CHAIN (Random Retrieval from a File),” “SETLL (Set Lower Limit),” or “SETGT (Set Greater Than)” operation).

If the file from which you are reading is an update disk file, you can specify an N in position 53 to indicate that no lock should be placed on the record when it is read. See the *RPG/400 User's Guide* for more information.

You can specify an indicator in positions 56 and 57 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 58 and 59 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 76 on page 324 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 Programmer's Guide* 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 56 and 57 is set on. See the *ICF Programmer's Guide* 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 ID entry on the file continuation 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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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 END OF FILE
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 in position 53 indicates that the record is not locked.
C*
C                   READ REC1               N    64 END OF FILE
C   64              GOTO EOF
C*
C           EOF     TAG
```

Figure 76. 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 “Continuation Line Summary Chart” 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 ID entry (on the file specifications continuation line). 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 56 and 57 to be set on if an error occurs while the operation is running. A resulting indicator in positions 58 and 59 is required; it is set on when there are no more changed records in the subfile.

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 16 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 58 and 59 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 factor 1 equals the key of the current record or if factor 1 is not specified, the key comparison takes place at the Data Management level; otherwise, it takes place within the RPG/400 program. If the file being read is defined as update and the compare is by RPG, 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 in position 53) 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 19 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 19 of the file description specifications). See “File Operations” on page 196 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 in position 53 to indicate that no lock should be placed on the record when it is read. See the *RPG/400 User's Guide* for more information.

You can specify a resulting indicator in positions 56 and 57 to be set on if the operation does is completed successfully. You must specify a resulting indicator in positions 58 and 59. 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 one or more key fields in the file being read from are defined with ALTSEQ, ABSVAL, DIGIT, or ZONE, the result of the read operation may differ from that expected. The reason is the content of the field on which the

access path is built may not be the same as the content that the READE operation is using for its internal comparison.

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 56 and 57 is set on. No further I/O operations can be issued against the file until it is successfully closed and reopened.

Note: If the key used contains a numeric (packed or zoned) field, the search argument must exactly match the key field. For example, if the physical file uses a packed key of X'123C' for +123. and the search argument is 123, READE will use X'123F' and EOF will be returned.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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    READEFILEA                    55 NOT EQUAL
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    READEREC1                     56 NOT EQUAL
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                     READEFILEA                    55 NOT EQUAL
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 in position 53 indicates that the record is not locked.
C                     READEREC1               N     56 NOT EQUAL
```

Figure 77. 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 16 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 19 of the file description specifications). See “File Operations” on page 196 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 (Random Retrieval from a File)” or “SETLL (Set Lower Limit)” operation). You can specify an indicator in positions 56 and 57 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 in position 53 to indicate that no lock should be placed on the record when it is read. See the *RPG/400 User’s Guide* for more information.

You must specify an indicator in positions 58 and 59 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 be reposition the file after the indicator is set on to process any further successful sequential operation (for example, “READ (Read a Record)”) to the file.

Figure 78 on page 330 shows READP operations with a file name and record format name specified in factor 2.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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                     READPFILEA                    71 BOF
C   71                GOTO BOF                         BEG OF FILE
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                     READPREC1                     7272 = BOF
C   72                GOTO BOF
C*
C           BOF       TAG
```

Figure 78. READP Operation

REDPE (Read Prior Equal)

Code	Factor 1	Factor 2	Result Field	Indicators
REDPE (n)	Search argument	File name, Record name	Data structure	_ ER BOF

The REDPE 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 58-59 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 19 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 19 of the file description specifications). See “File Operations” on page 196 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 in position 53 to indicate that no lock should be placed on the record when it is read. See the *RPG/400 User's Guide* for more information.

You can specify a resulting indicator in positions 56 and 57 to be set on if the operation is not completed successfully. You must specify a resulting indicator positions 58 and 59. 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 REDPE operation is not successful, you must reposition the file (for example, by a “CHAIN (Random Retrieval from a File)” or “SETLL (Set Lower Limit)” operation).

Note: If factor 1 equals the key of the current record or if factor 1 is not specified, the key comparison takes place at the Data Management level; otherwise, it takes place within the RPG/400 program. If the file being read is defined as update and the compare is by RPG, 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 in position 53) is specified, a temporary lock is not requested.

If factor 1 is specified and one or more key fields in the file being read from are defined with ALTSEQ, ABSVAL, DIGIT, or ZONE, the result of the read operation may differ from that expected. The reason is that the content of the field on which the access path is built may not be the same as the content that the REDPE operation is using for its internal comparison.

A REDPE (with factor 1 specified) that immediately follows an OPEN operation or a BOF condition returns BOF. A REDPE (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 56 and 57 is set on. The file *must* be positioned (using "CHAIN (Random Retrieval from a File)," "SETLL (Set Lower Limit)," "READ (Read a Record)," "READP (Read Prior Record)," or "READE (Read Equal Key)" (with factor 1 specified)) prior to issuing a REDPE operation with factor 1 blank. A SETGT operation code should not be used to position the file prior to issuing a REDPE 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).

If the key used contains a numeric (packed or zoned) field, the search argument must exactly match the key field. For example, if the physical file uses a packed key of X'123C' for +123. and the search argument is 123, REDPE will use X'123F' and BOF will be returned.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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    REDPEFILEA                    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    REDPEFILEB     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    REDPERECA                   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                     REDPEFILEA                    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                     REDPEFILEB     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                     REDPERECA                   8899
```

Figure 79. REDPE 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 multiple device 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 56 and 57 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.

To release a record lock, use the UNLCK operation. See the UNLCK operation for more information about unlocking data areas and releasing record locks for update disk files.

The REL operation is valid only for multiple device WORKSTN files.

RESET (Reset)

Code	Factor 1	Factor 2	Result Field	Indicators
RESET	*NOKEY	Structure or Variable		

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 operation of the program. This value can be set using data structure initialization, or you can use the initialization subroutine to assign an initial value to the structure or variable. When RESET is specified for a structure or variable, a snapshot of the initial value of that variable or structure is taken at the end of the *INIT operation after *INZSR (initialization subroutine) is processed. See Figure 4 on page 14 for more information. This value is then used to reset the structure or variable.

Factor 1 must be blank unless factor 2 contains a record format name. In this case, factor 1 can contain *NOKEY, which indicates that all fields except key fields are to be reset to their initial values.

Factor 2 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 factor 2 using an array index, only the element specified is reset.

When the RESET operation is applied to a record format name, only those fields that are output in that record format are affected. For WORKSTN (positions 40-46) file record formats, only those fields with a usage of output or both are affected. All field-conditioning indicators 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.

The RESET operation is used in conjunction 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 field or structure. The value is then used to reset the field or structure if it appears in factor 2 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 because the initial values of all structures and variables that are reset must be saved. However, the amount of storage declared is reduced where possible. For example, if a data structure is reset and a subfield within that data struc-

ture is also reset, the save area for the subfield is based on the same storage as the data structure save area. Note that, if a single occurrence of a multiple-occurrence data structure is reset, a save area for the entire data structure (all occurrences) is declared. A save area is created for a structure or variable only if it appears in factor 2 of a RESET operation. If no RESET operations are coded in the program, then no additional storage is required. 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 *RPG/400 User’s Guide* and the “CLEAR (Clear)” operation code.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilenameIPEAF....RlenLK1AIOvKlocEDevice+......KExit++Entry+A....U..
FEXTFILE O   E                    DISK
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
I..............Namedconstant+++++++++C.........Fldnme.............
I*
I* The file EXTFILE contains one record format RECFMT containing
I* the character fields CHAR1 and CHAR2 and the numeric fields
I* NUM1 and NUM2.
IDS1        IDS
I I            'MONDAY'                1   8 DAY1
I I            'THURSDAY'              9  16 DAY2
I                                     17  22 JDATE
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C* The following operation blank DAY1, DAY2, and JDATE.
C                     CLEARDS1
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                     RESETDS1
C* The following operation will set CHAR1 and CHAR2 to blanks and
C* NUM1 and NUM2 to zero.
C                     CLEARRECFMT
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                     RESETRECFMT
C*
C           *INZSR    BEGSR
C                     MOVELUDATE     JDATE
C                     MOVEL'NAME    'CHAR1
C                     MOVEL'ADDRESS 'CHAR2
C                     Z-ADD1         NUM1
C                     Z-ADD2         NUM2
C                     ENDSR
```

Figure 80. RESET Operation

RETRN (Return to Caller)

Code	Factor 1	Factor 2	Result Field	Indicators
RETRN				

The RETRN operation causes a return to the caller as follows:

1. The halt indicators are checked. If a halt indicator is on, the program ends abnormally. (All open files are closed, an error return code is set to indicate to the calling routine that the program 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 program returns to the calling routine. Data is preserved for the next time the program is run. Files and data areas are not written out.

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 COMIT or ROLBK operation (or since the beginning of operations under commitment control if there has been no previous COMIT 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 COMIT operation (or at the time of the file OPEN, if there has been no previous COMIT operation.)

The file changes and the record-lock releases apply to all the files under commitment control in your routing step, whether the changes have been requested by the program issuing the ROLBK operation or by another program in the same routing step. 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 56 and 57 is set on if the operation is not successfully completed.

SCAN (Scan Character String)

Code	Factor 1	Factor 2	Result Field	Indicators
SCAN	Comparator string:length	Base string:start	Left-most position	_ ER FD

The SCAN operation scans a character 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.

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 must be character, and 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 must be character, and 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 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 58 and 59 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 character field length of the base string specified in factor 2.

Note: The strings are indexed from position 1. 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. Figurative constants cannot be used in the factor 1, factor 2, or result fields. 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 58 and 59 can be specified to be set on if the string being scanned for is found. A resulting indicator in positions 56 and 57 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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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 operation places the values 5 and 6 in the first and
C* second elements of the array.  Indicator 90 is set on.
C                     MOVE 'ARRAYY'  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                     MOVE 'TESTING' FIELD1   7
C                     Z-ADD2         X        10
C                     MOVEL'TOOL'    FIELD2   5
C           FIELD2:4  SCAN FIELD1:X  INT      20    90
C*
```

Figure 81. SCAN Operation

SELEC (Begin a Select Group)

Code	Factor 1	Factor 2	Result Field	Indicators
SELEC				

The select group conditionally processes one of several alternative sequences of operations. It consists of:

- A SELEC statement
- Zero or more WHxx groups
- An optional OTHER group
- ENDSL or END statement.

After the SELEC operation, control passes to the statement following the first WHxx condition that is satisfied. All statements are then executed until the next WHxx operation. Control passes to the ENDSL statement (only one WHxx is executed). If no WHxx condition is satisfied and an OTHER operation is specified, control passes to the statement following the OTHER operation. If no WHxx 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 SELEC operation. If they are not satisfied, control passes immediately to the statement following the ENDSL operation of the select group.

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 SELEC operation is specified inside a select group, the WHxx and OTHER operations apply to the new select group until an ENDSL is specified.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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 WHxx); 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                   SELEC
C           X       WHEQ 1
C                   Z-ADDA        B
C                   MOVE C        D
C                    :                              seq 1
C           Y       WHEQ 2
C           X       ANDLT10
C                    :                              seq 2
C                   OTHER
C                    :                              seq 3
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     CHAINFILE                   10
C  N10              SELEC
C           X       WHEQ 1
C                    :                            seq 1
C           Y       WHEQ 2
C                    :                            seq 2
C                   ENDSL
C                   ADD  1        N
```

Figure 82. SELEC 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 16 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 (Define a Composite Key)" 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, in the OS/400 system, 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 54 and 55 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 56 and 57 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. 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 numeric keys:

- 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 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.

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 *CL Reference*.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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           KEY       SETGTFILEA                     GREATER THAN
C                     READ FILEA                   64READ NEXT
C*
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           KEY       SETGTFILEB                     GREATER THAN
C                     READPFILEB                   64READ LAST
```

Figure 83 (Part 1 of 4). SETGT Operation

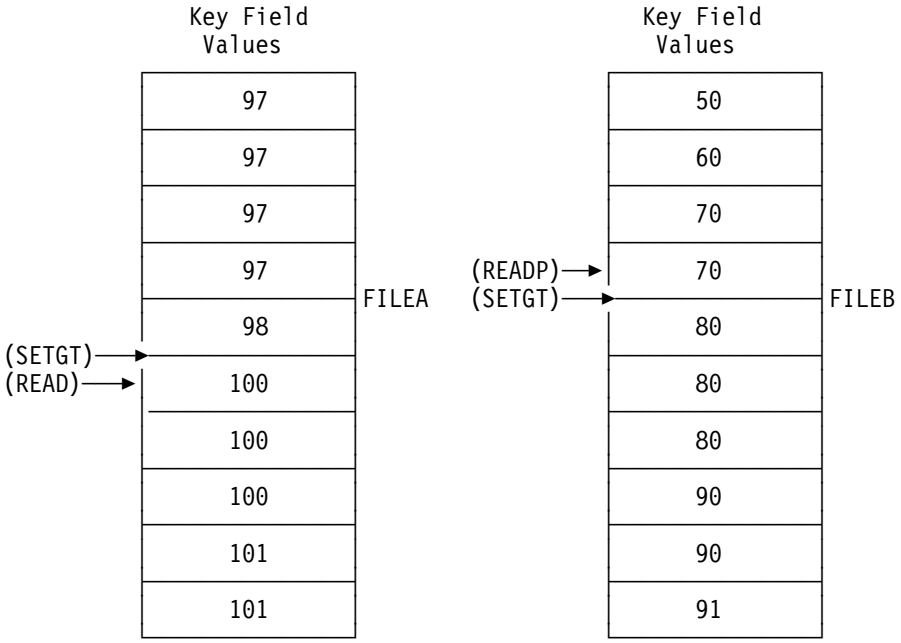


Figure 83 (Part 2 of 4). SETGT Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  This example shows the use of *LOVAL.  The SETGT 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           *LOVAL    SETGTRECDA                     GREATER THAN
C                     READ RECDA                   64READ NEXT
C*
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           *HIVAL    SETGTRECDB                     GREATER THAN
C                     READPRECDB                   64READ LAST
```

Figure 83 (Part 3 of 4). SETGT Operation

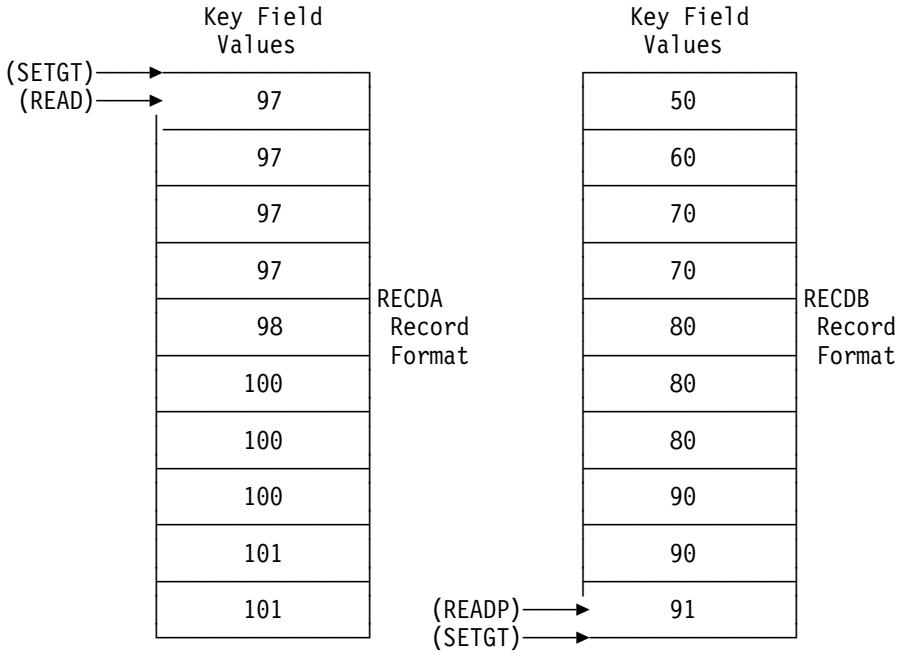


Figure 83 (Part 4 of 4). SETGT Operation

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 16 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 54 and 55, 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 56 and 57, it is set on when an error occurs during running of the operation. If an indicator is specified in positions 58 and 59, 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. 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 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 numeric keys:

- 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 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.

Figure 83 on page 345 (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 58 and 59 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 *CL 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 58-59) 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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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 101 record.  Because there are
C*  101 records, indicator 55 is set on and the GOTO operation is
C*  not processed.
C           ORDER     SETLLORDFIL                   55 101 PRESNT
C  N55                GOTO NOTFND
C           LOOP      TAG
C*  The READE operation reads the first 101 record.  Because
C*  indicator 56 is not on, the lines conditioned by N56 are
C*  processed.
C           ORDER     READEORDFIL                   56 END OF GRP
C*  The EXCPT operation is processed, and the program branches to the
C*  label specified in the GOTO operation.
C  N56                EXCPTDETAIL                    PRINT A LINE
C  N56                GOTO LOOP
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 and the two following operations are
C*  bypassed.
C           NOTFND    TAG
```

ORDFIL

	ORDER	Other Fields
	100	1st record of 100
	100	2nd record of 100
	100	3rd record of 100
(SETLL)→	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 84. SETLL Operation

SETOF (Set Off)

Code	Factor 1	Factor 2	Result Field	Indicators
SETOF				OF OF OF

The SETOF operation sets off any indicators specified in positions 54 through 59. You must specify at least one resulting indicator in positions 54 through 59. 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 85 on page 352 illustrates the SETOF operation.

SETON (Set On)

Code	Factor 1	Factor 2	Result Field	Indicators
SETON				ON ON ON

The SETON operation sets on any indicators specified in positions 54 through 59. You must specify at least one resulting indicator in positions 54 through 59. 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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  The SETON and SETOF operations set from one to three indicators
C*  specified in positions 54 through 59 on and off.
C*  The SETON operation sets indicator 17 on.
C                   SETON                      17
C*  The SETON operation sets indicators 17 and 18 on.
C                   SETON                      1718
C*  The SETOF operation sets indicator 21 off.
C                   SETOF                      21
```

Figure 85. SETON and SETOF 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 54 and 55 is set on. Positions 54 and 55 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 *CL Reference*.

Positions 56 through 59 must be blank.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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 and the GOTO END operation is processed.
C                     SHTDN                     27
C   27                GOTO END
```

Figure 86. 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 in position 45 of the extension specifications. 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. A related array, such as a second array defined on the same extension specification, is not sorted. Only the array specified in factor 2 is sorted.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments
E                    ARRY       8  1  A
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*  The SORTA operation sorts ARRY into ascending sequence because A
C*  is specified in position 45 of the extension specifications.
C*  If the nonsorted ARRY contents were GT1BA2L0, the sorted ARRY
C*  contents would be ABGLT012.
C*
C                    SORTAARRY
```

Figure 87. SORTA Operation

SQRT (Square Root)

Code	Factor 1	Factor 2	Result Field	Indicators
SQRT (½)		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/400 exception/error handling routine receives control.

For further rules on the SQRT operation, see “Arithmetic Operations” on page 189.

See Figure 33 on page 191 for an example of the SQRT operation.

SUB (Subtract)

Code	Factor 1	Factor 2	Result Field	Indicators
SUB(½)	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 189.

See Figure 33 on page 191 for examples of the SUB operation.

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.

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 character string, or the base character string followed by ':', followed by the start location. The base string portion must be character, and 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.

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 56 and 57 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, and can contain one of the following: a field name, array element, data structure, or table name. The substring is left-justified, and its 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. No 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 (position 53) indicates that the result field should be padded on the right with blanks after the substring occurs.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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 RESULT.  Indicator 90 is not set on because no error
C* occurred.
C                   Z-ADD3         T       20
C                   MOVEL'ABCDEF'  STRING 10
C           2       SUBSTSTRING:T  RESULT      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-ADD4         T       10
C           5       SUBSTSTRING: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 in position 53, only the first 3 characters of RESULT
C* are changed.  RESULT contains '123XXXXX'.
C                   Z-ADD3         LENGTH  20
C                   Z-ADD5         T       20
C                   MOVE 'TEST123' STRING  8
C                   MOVE *ALL'X'   RESULT
C           LENGTH  SUBSTSTRING:T  RESULT  8    90
C*
C* This example is the same as the previous one except P is
C* specified in position 53, and the result is padded with blanks.
C* RESULT equals '123bbbbb'.
```

Figure 88 (Part 1 of 2). SUBST Operation

```
C                   Z-ADD3         LENGTH  20
C                   Z-ADD5         T       20
C                   MOVE 'TEST123' STRING  8
C                   MOVE *ALL'X'   RESULT
C           LENGTH  SUBSTSTRING:T  RESULT  8 P  90
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                   SUBSTCITY:C    TCNTRE
C*
C* Before the operations STRING='ЬЬЬJohnЬЬЬЬЬ'.
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* 'J' to the end of STRING.  Padding is used to ensure that
C* nothing is left from the previous contents of the result
C* field.
C* After the operations RESULT='JohnЬЬЬЬЬ'.
C*
C           ' '     CHECKSTRING    ST             10
C   10              SUBSTSTRING:ST RESULT    P
```

Figure 88 (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 17) 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/400 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 62 on page 272 for examples of the TAG operation.

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 54 through 59 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 affected. The field specified in factor 2 or 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 89 on page 362 illustrates uses of the TESTB operation.

Indicators assigned in positions 54 through 59 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 54 and 55:* 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 56 and 57:* 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, some of the specified bits are equal to 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 56 and 57 is not set on.
- *Positions 58 and 59:* 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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  The field bit settings are FLDF = 00000001, and FLDG = 11110001.
C*
C*  Indicator 16 is set on because bit 3 is off (0) in FLDF.
C*  Indicator 17 is set off.
C                   TESTB'3'      FLDF       16  17
C*
C*  Indicator 16 is set on because both bits 3 and 6 are off (0) in
C*  FLDF.  Indicators 17 and 18 are set off.
C                   TESTB'36'     FLDF       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'     FLDF       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'      FLDF       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                   TESTBFLDG     FLDF       161718
C*
C*  The hexadecimal literal X'88' (10001000) is used in factor 2.
C*  Indicator 17 is set on because bit 0 is on and bit 4 is off.
C*  Indicators 16 and 18 are set off.
C                   TESTBX'88'    FLDG       161718
```

Figure 89. 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 54 and 55:* Either the result field contains numeric characters, or it contains a 1-character field that consists of a letter from A to R.
- *Positions 56 and 57:* The result field contains both numeric characters and at least one leading blank. For example, the values ␣123 or ␣␣123 set this indicator on. However, the value ␣1␣23 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 56 and 57 when a field of length one is tested because the character field must contain at least one numeric character and one leading blank.
- *Positions 58 and 59:* 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.

To prevent undesirable results or an abnormal end of a program, the TESTN operation validates data in fields before arithmetic or editing operations are processed on the fields. Following validation, the field must be moved to a numeric field to process the arithmetic and editing operations.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  The field values are FLDA = 123, FLDB = 1X4, FLDC = 004,
C*  FLDD = ␣␣␣, FLDE = ␣1␣3, and FLDF = ␣12.
C*
C*  Indicator 21 is set on because FLDA contains all numeric
C*  characters.
C                     TESTN          FLDA       21
C*  Indicator 22 is set on because FLDA contains all numeric
C*  characters.  Indicators 23 and 24 remain off.
C                     TESTN          FLDA       222324
C*  All indicators are off because FLDB does not contain valid
C*  numeric data.
C                     TESTN          FLDB       252627
C*  Indicator 28 is set on because FLDC contains valid numeric data.
C*  Indicators 29 and 30 remain off.
C                     TESTN          FLDC       282930
C*  Indicator 33 is set on because FLDD contains all blanks.
C*  Indicators 31 and 32 remain off.
C                     TESTN          FLDD       313233
C*  Indicators 34, 35, and 36 remain off.  Indicator 35 remains off
C*  off because FLDE contains a blank after a digit.
C                     TESTN          FLDE       343536
C*  Indicator 38 is set on because FLDF contains leading blanks and
C*  low order digits.  Indicators 37 and 39 remain off.
C                     TESTN          FLDF       373839
```

Figure 90. 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 54 through 59. The characters &, A through I, and any character with the same zone as the character A set on the indicator in positions 54 and 55. The characters - (minus), J through R, and any character with the same zone as the character J set on the indicator in positions 56 and 57. Characters with any other zone set on the indicator in positions 58 and 59.

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...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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   60      TIME OF DAY
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 120      TIME STAMP
C                   MOVELTIMSTP   TIME    60
C                   MOVE TIMSTP   SYSDAT  60
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 140      TIME STAMP
C                   MOVELTIMSTP   TIME    60
C                   MOVE TIMSTP   SYSDAT  80
```

Figure 91. TIME Operation

UNLCK (Unlock a Data Area or Release a Record)

Code	Factor 1	Factor 2	Result Field	Indicators
UNLCK		Data area or file name		_ ER _

The UNLCK operation is used to unlock data areas and release record locks in a program. The data area must already be specified in the result field of an *NAMVAR DEFN statement. If the UNLCK operation is specified to an already unlocked data area, an error does not occur.

In addition, the UNLCK operation allows the most recently locked record to be unlocked for an update disk file.

Factor 2 must contain the name of the data area to be unlocked, the name of an update disk file, or the reserved word *NAMVAR. When *NAMVAR is specified in factor 2, all data areas in the program that are locked are unlocked. Factor 2 must not refer to the local data area or the Program Initialization Parameters (PIP) data area.

The file specified in factor 2 must be an UPDATE disk file.

You can specify a resulting indicator in positions 56 and 57 to be set on if an error occurs on the operation. Positions 54, 55, 58, and 59 must be blank.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilenameIPEAF....RlenLK1AIOvKlocEDevice+......KExit++Entry+A....U
F*
FUPDATA  UF  E                    DISK
F*
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.  If *IN50 is on, the record is
C*  updated; otherwise the record is unlocked using the UNLCK
C*  operation.  Note that factor 2 of the UNLCK operation is the
C*  file name, UPDATA, not the record name, VENDOR.
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C                   READ VENDOR                   12
C                     :
C*          *IN50     IFEQ *ON
C*                    UPDATVENDOR                     Update record
C                     ELSE
C                     UNLCKUPDATA                99   Release record
C                     ENDIF
C*
```

Figure 92. UNLCK Operation

UPDAT (Modify Existing Record)

Code	Factor 1	Factor 2	Result Field	Indicators
UPDAT		File name	Data structure	_ ER _

The UPDAT operation modifies the last 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 UPDAT 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 56 and 57 to be set on if the UPDAT operation is not completed successfully.

Remember the following when using the UPDAT 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 UPDAT operation.
- Before UPDAT is issued to a file or record, a valid input operation with lock (READ, READC, READE, READP, REDPE, 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 UPDAT cannot be issued. The record must be read again with the default of blank in position 53 to specify a lock request.
- Consecutive UPDAT operations to the same file or record are not valid. Intervening successful read operations must be issued.
- Beware of using the UPDAT operation on primary or secondary files during total calculations. At this stage in the RPG/400 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 UPDAT 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 field specified in the ID entry of the file specifications continuation line. (If there is no such entry, the device for the last successful input operation is used.) This device must be the same one you specified for the input operation that must precede the UPDAT operation. You must not process input or output operations to other devices in between the input and UPDAT operations. If you do, your UPDAT 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 UPDAT operations to another subfile in the same display file. If you do, the UPDAT operation will fail.

WHxx (When True Then Select)

Code	Factor 1	Factor 2	Result Field	Indicators
WHxx	Comparand	Comparand		

The WHxx operations of a select group determine where control passes after the "SELEC (Begin a Select Group)" operation is processed.

The WHxx 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 WHxx are processed until the next WHxx, OTHER, ENDSL, or END operation.

When performing the WHxx operation remember:

- After the operation group is processed, control passes to the statement following the ENDSL operation.
- You can code complex WHxx conditions using ANDxx and ORxx. Calculations are processed when the condition specified by the combined WHxx, ANDxx, and ORxx operations is true.
- The WHxx 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 17) are not allowed.

Refer to "Compare Operations" on page 193 for valid values for xx.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C* The following example shows nested SELEC 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                     SELEC
C           EMPTYP    WHEQ 'C'
C           EMPTYP    OREQ 'T'
C                     Z-ADD0         DAYS
C           EMPTYP    WHEQ '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-ADD14        DAYS
C*
C* Nested select group.
C                     SELEC
C           YEARS     WHLT 2
C           YEARS     WHLE 5
C                     ADD  5         DAYS
C           YEARS     WHLE 10
C                     ADD  10        DAYS
C                     OTHER
C                     ADD  20        DAYS
C                     ENDSL
C* End of nested select group.
C*
C           EMPTYP    WHEQ 'S'
C                     Z-ADD5         DAYS
C                     ENDSL
```

Figure 93 (Part 1 of 2). WHxx Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C* Example of a SELEC group with complex WHxx 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 A (add) was entered and the record does not
C*     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 ACREC code is A; mark
C*     the record as deleted.
C*   * When action code is C (change), the record exists, and the
C*     ACREC code is A; update the record.
C*   * Otherwise, do error processing.
C*
C           RSCDE     CHAINFILE                 50
C                     SELEC
C           *INKC     WHEQ *ON
C                     EXSR QUIT
C           ACODE     WHEQ 'A'
C           *IN50     ANDEQ*ON
C                     WRITEREC
C           ACODE     WHEQ '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                     UPDATREC
C           ACODE     WHEQ 'C'
C           *IN50     ANDEQ*OFF
C           ACREC     ANDEQ'A'
C                     UPDATREC
C                     OTHER
C                     EXSR ERROR
C                     ENDSL
```

Figure 93 (Part 2 of 2). WHxx 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 56 and 57 can contain an indicator to be set on if the WRITE operation is not completed successfully. The indicator in positions 56 and 57 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 58 and 59; 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 RECNO (relative record number) field 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 66 of the file description specifications. (See “Position 66 (File Addition)” on page 100.)
- Device dependent functions are not available. For example, if a WRITE is issued to a PRINTER device, there is no spacing or skipping (normally specified in columns 17 through 22 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 specified in the ID entry on the file specifications continuation line. If there is no such entry, data is written to the program device for which the last successful input operation was processed.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C*  The WRITE operation writes the fields in the data structure
C*  DS1 to the file, FILE1.
C*
C                   WRITEFILE1     DS1                ADD RECORD
```

Figure 94. WRITE Operation

XFOOT (Summing the Elements of an Array)

Code	Factor 1	Factor 2	Result Field	Indicators
XFOOT (½)		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 (position 53) 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 189.

See Figure 33 on page 191 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. 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, and can contain one of the following: a field name, array element, named constant, data structure name, data structure subfield, 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 character field, character array element, data structure or a character 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 17.

If factor 2 is shorter than the result field, a P specified in the operation extender position (position 53) indicates that the result field should be padded on the right with blanks after the translation.

Columns 54 and 55 must be blank. An indicator in positions 56-57 turns on if an error occurs on the operation. Columns 58-59 must be blank.

Both factor 2 and the result field must be character or both must be DBCS.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
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           ' ':'-'   XLATENUMBER    RESULT  8
```

Figure 95. XLATE Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
I..............Namedconstant+++++++++C.........Fldnme.............
I*
I* In the following example, all values in STRING are translated to
I* uppercase.  As a result, RESULT='RPG DEP'.
I*
I             'ABCDEFGHIJKLMNOPQRS-  C         UP
I             'TUVWXYZ'
I             'abcdefghijklmnopqrs-  C         LO
I             'tuvwxyz'
C*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C                   MOVE 'RpG Dep' STRING  7
C           LO:UP     XLATESTRING    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     XLATESTRING    RESULT       90
```

Figure 96. XLATE Operation With Named Constants

Z-ADD (Zero and Add)

Code	Factor 1	Factor 2	Result Field	Indicators
Z-ADD (½)		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 (position 53) can be specified.

For the rules for the Z-ADD operation, see “Arithmetic Operations” on page 189.

See Figure 33 on page 191 for an example of the Z-ADD operation.

Z-SUB (Zero and Subtract)

Code	Factor 1	Factor 2	Result Field	Indicators
Z-SUB (½)		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 (position 53) can be specified.

For the rules for the Z-SUB operation, see “Arithmetic Operations” on page 189.

See Figure 33 on page 191 for an example of the Z-SUB operation.

Chapter 12. RPG/400 Words with Special Functions

The following RPG/400 reserved words have special functions within a program:

- UDATE, *DATE, UMONTH, *MONTH, UYEAR, *YEAR, UDAY, and *DAY allow you to access the system date, or a portion of it, to be used in the program.
- PAGE, PAGE1-PAGE7 can be used for numbering the pages of a report, for record sequence numbering, or to sequentially number output fields.
- Figurative constants (*BLANK/*BLANKS, *ZERO/*ZEROS, *HIVAL, *LOVAL, *ON/*OFF, *ALLX'x1..', and *ALL'X..') are implied literals that allow specifications without referring to length.
- *IN and *INxx allow indicators to be referred to as data.
- The following reserved words define symbolic locations within the file information data structure (INFDS) and the program status data structure.

  *FILE
  *OPCODE
  *PARMS
  *PROGRAM
  *RECORD
  *ROUTINE
  *STATUS

- The following reserved words provide symbolic labels for the ENDSR operation for the file and program exception/error subroutines or for the INFDS.

*CANCL	Cancel the program
*DETC	Detail calculations
*DETL	Detail lines
*GETIN	Get input record
*INIT	Program initialization
*OFL	Overflow lines
*TERM	Program ending
*TOTC	Total calculations
*TOTL	Total lines

- Special words used with operation codes.

  *DEFN
  *ENTRY
  *INZSR
  *LDA
  *LIKE
  *LOCK
  *NAMVAR
  *OFF
  *ON
  *PDA
  *PSSR

- Special words used with translation.

  *FILE
  *EQUATE

- *PLACE allows repetitive placement of fields in an output record. (See "*PLACE" on page 177 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)

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, and are written in the format specified by the control specification. (For a description of the job date, see the *Work Management Guide*.)

Rules for User Date

Remember the following rules when using the user date:

- UDATE, when specified in positions 32 through 37 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. Both special words can produce three different date formats:

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

  Use positions 19 and 20 of the control specification to specify the date format and the editing to be done. If positions 19 and 20 are blank, the date format is determined by the contents of position 21.

- For an interactive or batch program, the user date special words are set to the value of the Job Date when the program starts running in the system. The value of the user date special words are not updated during program execution even if the program runs over midnight or if the job is changed. 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 32 through 37 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 38 of the output specifications. The control specification entry in position 20 determines 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 38 of the output specifications.
- 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.

Note: The operation codes CLEAR and RESET, the array index for factor 2 of LOKUP, and factor 1 of PARM are exceptions to this rule. The user date special words cannot be used with them.

- The user date special words cannot appear as the result field in a calculation or as an input field.
- Blank-after (position 39 of the output specifications) cannot be used with user date special words.
- *YEAR is a 4-digit numeric field containing the year portion of the job date.
- *MONTH and *DAY function precisely as UMONTH and UDAY, respectively.

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. PAGE1 through PAGE7 are used to serially number several output files.

The eight possible PAGE entries (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 and PAGE1 through PAGE7 can be specified in positions 32 through 37 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, enter that page number in a field of an input record and name that field PAGE in positions 53 through 58, or use a calculation operation such as Z-ADD. The number entered in the PAGE field should be 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 97 on page 384). Any entry in the PAGE field should be right-adjusted, such as 0023.
- 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 39 of the output specifications).
  - Specify the PAGE field as the result field of an operation in the calculation specifications.
  - Specify output indicators in the output specifications (see Figure 98). 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 97.

- 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 44) has been specified. Editing and the data format override the suppression of leading zeros.
- PAGE can be specified in input or calculation specifications, and can be of any length. 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...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+L1M1FrPlMnZr...
IINPUT   PG  50   1 CP
I                                        2   50PAGE
```

Figure 97. Page Record Description

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
OName++++DFBASbSaN01N02N03Field+YBEnd+PConstant/editword+++++++++...
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   01   L1
O                15      PAGE  1   75
```

Figure 98. Resetting the PAGE Fields to Zero

Figurative Constants

The figurative constants *BLANK/*BLANKS, *ZERO/*ZEROS, *HIVAL, *LOVAL, *ALL'x..', *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 385.)

Figurative constants can be specified in positions 18 through 27 (factor 1) and in positions 33 through 42 (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 fields.
*ZERO/*ZEROS	Character/numeric fields: All zeros.
*HIVAL	Character fields: The highest collating character for the system (hexadecimal FFs). Numeric fields: All nines with a positive sign.
*LOVAL	Character fields: The lowest collating character for the system (hexadecimal zeros). Numeric fields: All nines with a negative sign.

*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.
*ALLX'x1..'	Character fields: The hexadecimal literal X'x1..' is cyclically repeated to a length equal to the associated field.
*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:

- 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..' and *ALLX'x1..'. The character 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 character string.
- Figurative constants are valid in compare operations such as COMP, CAB, DOU, DOW, and IF; when the associated field in the compare operations is the field with which the figurative constant is to be compared.
- Figurative constants are not allowed in factor 1 of a DEBUG or DSPLY operation or in factor 2 of MHLZO, MLHZO, MHHZO, MLLZO, BITON, BITOF, TESTB, or SQRT operations.
- The figurative constants, *BLANK/*BLANKS, are moved as zeros to a numeric field in a MOVE operation.
- *BLANK/*BLANKS do not cause any storage allocation if used in factor 2. Otherwise, storage equivalent to the implied length of blanks is used. The performance of *BLANK/*BLANKS is equal to using spaces when the implied length of blanks is less than or equal to 140.

Chapter 13. Using Arrays and Tables

An array is a systematic program-internal arrangement of data fields (**array elements**) with the same field length, data type (character or numeric), and number of decimal positions (if numeric). You can search an array sequentially for an identifiable array element, or you can refer to an array element by its position within the array. With some operations, you can refer to all of the array elements by using only the array name.

A table is also a systematic program-internal arrangement of data fields (table elements) with the same field length, data type (character or numeric), and number of decimal positions (if numeric). You search a table sequentially, using the LOKUP operation, to find a uniquely identifiable table element and any associated data. You cannot refer to table elements by their position within the table. Except in the LOKUP operation, the table name refers to the last table element found in a LOKUP operation. Unlike an array name, the table name does not refer to the entire set of table elements.

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 the 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. It 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 comma, followed by (3) an index (for example: AR,IND). The index indicates the position of the element within the array 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 array name with comma and index can be up to 6 characters long. If the array is only specified in factor 1 or factor 2 of calculation specifications, the array name with comma and index can be up to 10 characters long.
- The index is a numeric field with zero decimal positions, or a numeric constant.
- 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.

Here are some examples of valid and invalid specifications of an array name and index:

- *Valid Array Names and Indexes:*
  - `AR,1` This is the first element of array `AR`.
  - `X,YY2` This is an element of array X. The index indicates which element of the array this is. `YY2` is the name of a field containing the index value.
- *Invalid Array Names and Indexes:*
  - `AR,+1` The array name has an invalid signed index.
  - `AR,0` The index value must be between 1 and the number of elements in the array, inclusive.
- *Array Names and Indexes Valid in Some Situations:*
  - `BAL,XX1` The name including the comma has more than 6 characters. This name is only valid for factor 1 and factor 2 of calculation specifications. It is not a valid name for the result field because the result field is only six positions long.

The Essential Array Specifications

You define an array on an extension specifications line. Here are the essential specifications for all arrays:

- Specify the array name in positions 27 through 32.
- Specify the number of entries in the array, right justified, in positions 36 through 39.
- Specify the length of an entry, right justified, in positions 40 through 42.
- If the array elements are numeric, specify the number of decimal positions in position 44.

Figure 99 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 positions 33 through 35 must be blank for a run-time array.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments
E                    ARC        12  3
```

Figure 99. The Essential Array Specifications to Define a Run-Time Array

Loading a Run-Time Array

You assign initial values for a run-time array through input or calculation specifications. You may also put data into other types of arrays this way.

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 storing this data in the records of a file.

Array Information in One 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 100 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...
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments
E                    INPARR      6 12
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+L1M1FrPlMnZr...*
IARRFILE AA  01
I                                        1  72 INPARR
```

Figure 100. Defining 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 101 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...
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments
E                    ARRX        6 12
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+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 101. Defining a Run-Time Array with Scattered Elements

When an array is located in a data structure, all of the elements of the array are consecutive in the data structure.

Array Information in More Than One Record

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. Figure 103 on page 391 shows the array that results when array information is loaded from more than one input record. Each record identified by a 1 or 3 in position 1 contains 12 elements of array information. Records identified by a 2 in position 1 do not contain array information, although they appear in the same input file. The RPG/400 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 infor-

mation 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 extension specifications (A or D in position 45) defining the array. If the sequence is not specified, the array is sorted into ascending sequence. When the high (positions 54 and 55 of the calculation specifications) or low (positions 56 and 57 of the calculation specifications) indicators are used in the LOKUP operation, the array sequence must be specified.

Coding a Compile-Time Array

For a *compile-time array*, you must specify in positions 33-35 of the extension specification how many array entries are in an array input record. This is in addition to the required array specifications on the extension specification. See the specifications in Figure 102.

Loading a Compile-Time Array

For a *compile-time array*, enter array input data into records in the program source member, following the source records for this program, and following the alternate collating sequence records and file translation records, if any. 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.

Rules for Array Input Records

The rules for array input records are:

- The first array entry for each input record must begin in position 1.
- An entire record need not be filled with entries. If it is not, blanks or comments can be included after the entries (see Figure 102). The unused entries in numeric arrays are filled with zeros; the unused entries in character arrays are filled with blanks.

```
*....+....1....+....2....+....3....+....4....+....5....+....6....+....*
     E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments
     E                    ARC    12  12  3
**
48K16343J64044H12648A47349K34650B125
```

48K	163	43J	640	44H	126	48A	473	49K	346	50B	125

```
                This is the compile-time array, ARC.
```

Figure 102. Array Input Record with Comments

- Each input record except the last must contain the same number of entries. A record can contain one entry or as many entries as the record can hold with the exception of the last record. In the last record, unused entries must be blank and comments can be included after the unused entries. Comments in

the last record must begin in the same position as comments in preceding records.

- Each entry must be contained entirely on one input record. An entry cannot be split between two records; therefore, the length of a single entry is limited to the maximum length of 80 characters (size of source record). If arrays are used and are described in alternating format, corresponding elements must be on the same input record; together they cannot exceed 80 characters.
- Arrays can be described separately or in alternating format. Alternating format means that the elements of one array are intermixed on the input record with elements of another array.

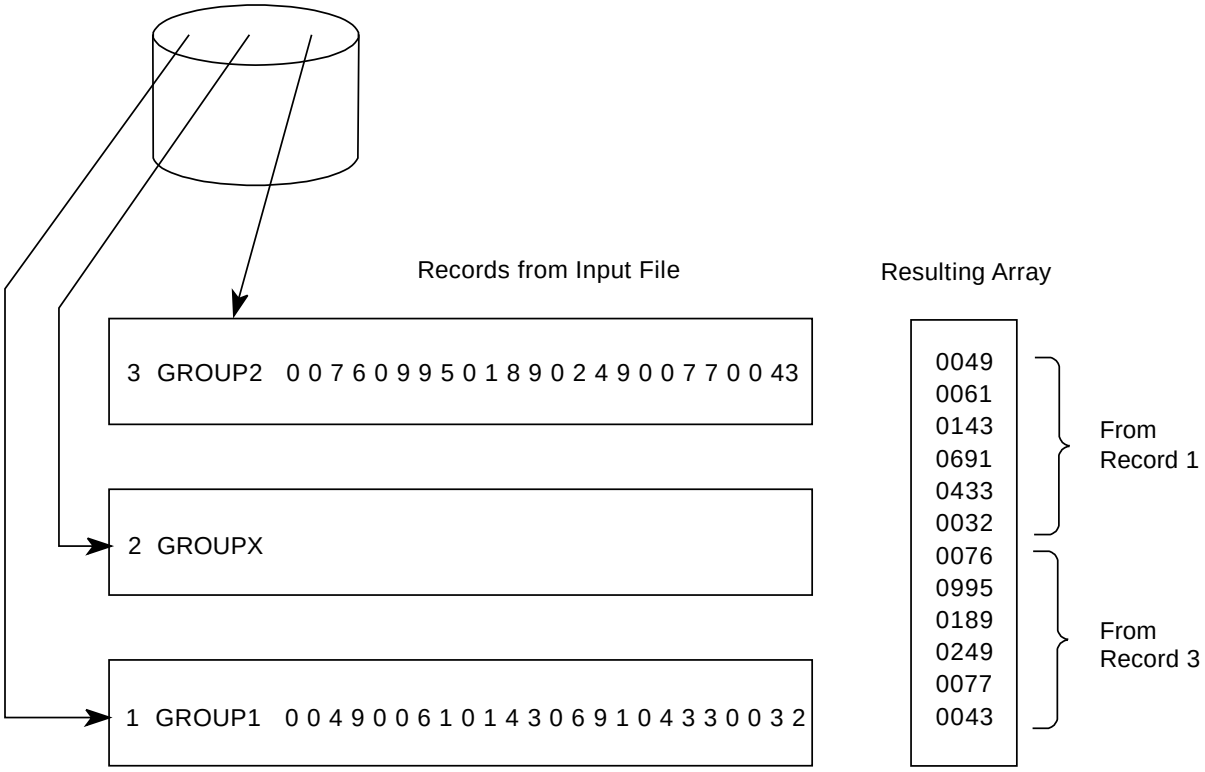


Figure 103. Loading an Array from Input Records

- All elements of an array must have the same characteristics (length, format and number of decimal positions, if numeric).
- A record with **b (b = blank) in positions 1 through 3 must precede the first input record of each array.
- Arrays are loaded in the order in which they are described in the extension specifications.
- Character array elements can contain a maximum of 80 positions. Numeric array elements can contain a maximum of 30 positions.
- Arrays can be in ascending, descending, or no sequence (A, D, or blank in positions 45 and 57 of the extension specifications).
- If L or R is specified in positions 43 and 55 of the extension specifications, the length specified for each element must include the sign (+ or -).

Coding a Prerun-Time Array

On the extension specifications, in addition to the essential array specifications, specify the name of the file with the array input data, in positions 11 through 18. In positions 19 through 26, you may optionally 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 15 of the file description specifications), the names in 11 through 18 and 19 through 26 must be the same.

In position 43, 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. Otherwise, leave position 43 blank.

Specify a T in position 16 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 104.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments
E* Compile-time arrays in alternating format.  Both arrays have
E* eight elements (three elements per record).  In both arrays, the
E* length of each element is 12, with four decimal positions.
E                    ARC     3   8 12 4 ARD    12 4
E*
E* Prerun-time array.  ARE, which is to be read from file DISKIN,
E* has 250 character elements (12 elements per record).  Each element
E* is five positions long.  The  elements are arranged in
E* ascending sequence.
E    DISKIN          ARE    12 250  5  A
E*
E* Run-time array.  ARI has 10 numeric elements, each 10 positions
E* with zero decimal positions.
E                    ARI        10 10 0
E*
E* Prerun-time array specified as a combined file.  ARH is written
E* back to the same file from which it is read.  ARH has 250
E* character elements (12 elements per record).  Each elements is five
E* positions long.  The elements are arranged in ascending sequence.
E    DISKOUT DISKOUT ARH    12 250  5  A
```

Figure 104. Extension Specifications for Four Types of Arrays

Figure 104 shows the extension specifications required for four types of arrays.

For compile-time arrays, positions 19 through 26 can also contain the name of a file to which the array is to be written at end of job. For prerun-time arrays, positions 46 through 51 can also describe an array that is entered in alternating format with the array described in positions 27 through 32.

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.

Data Structure Initialization with Arrays

Run-Time Arrays

To initialize each element in a run-time array to the same value, specify the array as a data structure subfield, place an I in position 8, and specify the initialization value in positions 21 to 42 of the input specifications. In the case where a subfield initialization overlaps a run-time array, 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

Subfield initialization values (positions 21 to 42 of the input specifications) cannot be specified for a compile-time or prerun-time array, because they are initialized by definition. If a compile-time or prerun-time array appears in a globally initialized data structure (I in position 8 of the input specifications), 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.

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 More than one Array

There are three ways that you can define more than one array, which are:

- Defining two run-time arrays
- Mixing compile-time and prerun-time arrays
- Loading two compile-time or two prerun-time arrays in alternating format.

Two Run-Time Arrays

You can specify two run-time arrays on one extension specifications line by entering a second array name, length of entry, and decimal position in positions 46 through 57 of the first array's extension specifications.

Mixing Compile-Time and Prerun-Time Arrays

The specifications for compile-time and prerun-time arrays and tables can be intermixed in the extension specifications. The sequence in which arrays are specified in the extension specifications determines the order in which they are loaded at the start of the program.

Arrays in Alternating Format

You can load two compile-time arrays or two prerun-time arrays in *alternating format* by specifying a second array name, length of entry, and decimal position in positions 46 through 57 of the first array's extension specifications. Arrays in this format are referred to as **alternating arrays**. (Tables are called **alternating tables**.) 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 specification for the number of entries per record in positions 33 through 35 of the extension specifications indicates the number of corresponding pairs per record, each pair of elements counting as a single entry.

Figure 105 shows two arrays, ARRA and ARRB, in alternating format.

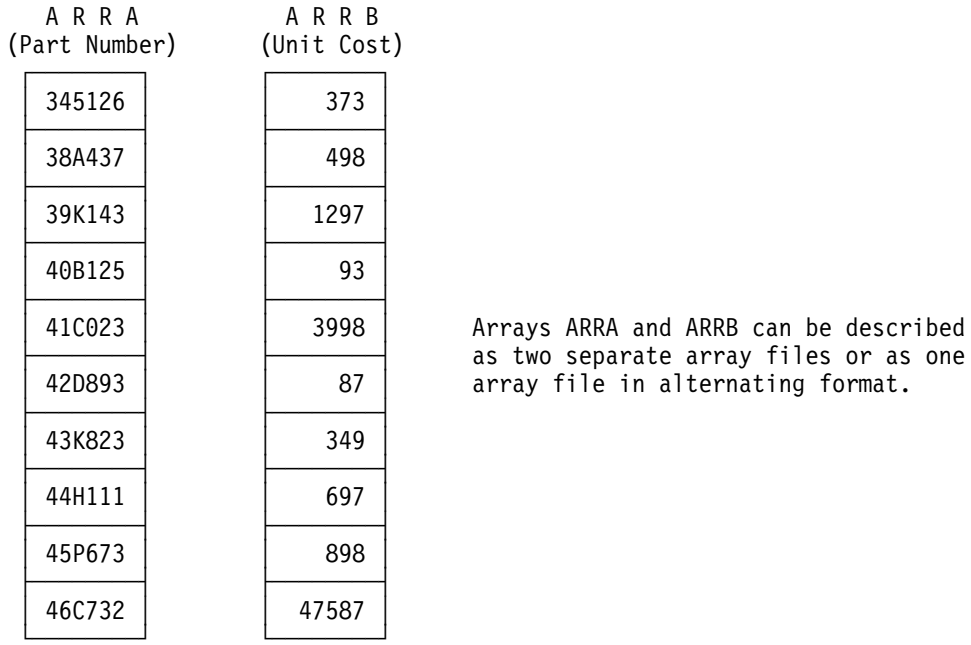


Figure 105. Arrays in Alternating and Nonalternating Format

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 106. 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 107. 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.....	7.....	12....	18....	23....	29....	34....	40....	45....	51....

Figure 108. Arrays Records for One Array File in Alternating Format

Searching Arrays

The LOKUP operation can be used to search arrays. See “LOKUP (Look Up)” on page 286 for a description of the LOKUP 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 LOKUP 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). Factor 1, the search argument, can be:

- A character or numeric literal
- A field name
- A data structure name
- An array element
- A table name.

Specify the operation code LOKUP in positions 28 through 32. 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 LOKUP operation. The resulting indicators must *not* be specified in high or low if the array is not in sequence (A or D in position 45 and/or position 57 of the extension specifications). Conditioning indicators (specified in positions 7 through 17) 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 109 shows an example of a LOKUP on an array without an index.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilenameIPEAF....RlenLK1AIOvKlocEDevice+......KExit++Entry+A....U1.
FARRFILE IT  F       5           EDISK
F*
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments
E    ARRFILE         DPTNOS  1  50  5 0             DEPT NUMBERS
E*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C* The LOKUP 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    LOKUPDPTNOS                   20
C*
C* Otherwise, indicator 20 is not set on and the GOTO statement
C* conditioned by N20 causes a branch to the BADNUM TAG.
C  N20                GOTO BADNUM
C                       :
C                       :           Calculations
C                       :
C           BADNUM     TAG
```

Figure 109. LOKUP 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 15) with an array file designation (T in position 16). The file is program described (F in position 19), and each record is 5 positions in length (5 in position 27). The E in position 39 indicates that the file is further described in the extension specifications.

In the extension specifications, ARRFILE is defined as containing the array DPTNOS. The array contains 50 entries (positions 38 and 39). Each entry is 5 positions in length (position 42) with zero decimal positions (position 44). One department number can be contained in each record (1 in position 35). However, each record does not have to contain an entry. Any record that does not contain an entry is filled with zeros.

Searching an Array with an Index

To find out which element satisfies a LOKUP 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 positions 33 through 42, enter the name of the array to be searched, followed by a comma and a numeric constant or the name of a numeric field (with zero decimal positions) containing the number of the element to be used. The numeric constant or numeric field provides 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. All other rules that apply to an array without an index apply to an array with an index.

Figure 110 shows a LOKUP on an array with an index.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilenameIPEAF....RlenLK1AIOvKlocEDevice+......KExit++Entry+A....U1.
FARRFILE IT  F       5           EDISK
F*
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments+++++++++
E    ARRFILE         DPTNOS  1  50  5 0 DPTDSC 20     DEPT NO/DESCRPTN
E*
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C* The Z-ADD operation begins the LOKUP at the first element in DPTNOS.
C                    Z-ADD1         X      30       SET X FOR LOKUP
C* At the end of a successful LOKUP, 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    LOKUPDPTNOS,X                 20
C* If an element is not found that is equal to the search argument,
C* the GOTO operation conditioned by N20 causes a branch to BADNUM.
C  N20               GOTO BADNUM
C                    MOVE DPTDSC,X   DPTNAM 20
C           BADNUM    TAG
```

Figure 110. LOKUP Operation on an Array with an Index

This example shows the same array of department numbers, DPTNOS, as Figure 109 on page 396. However, an alternating array of department descriptions, DPTDSC, is also defined in positions 46 through 51 of the extension specifications. Each element in DPTDSC is 20 positions in length (positions 53 and 54). Any record in DPTNOS that does not contain an entry is filled with zeros. Any record in DPTDSC that does not contain an entry is filled with blanks.

Specifying Arrays

Arrays can be used in input, output, or calculation specifications after they are specified on the extension 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. Remember, if an array element is to be used as a result field, the array name with the comma and index cannot exceed 6 characters.

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, MOVE, MOVEL, MOVEA, MLLZO, MLHZO, MHLZO, MHHZO, DEBUG, XFOOT, LOKUP, SORTA, PARM, DEFN, CLEAR, RESET, CHECK, CHEKR, 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, BITOF, COMP, CABxx, TESTZ, TESTN, TESTB, MVR, DOUxx, DOWxx, IFxx, SUBST, and CAT.

When specified with an array name without an index, certain operations are repeated for each element in the array. These are ADD, Z-ADD, SUB, Z-SUB, 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 do 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.
- Resulting indicators (positions 54 through 59) cannot be used because of the number of operations being processed.

Modifying Contents of Arrays

Arrays can be temporarily changed while the program is running when the array name is used as a result field in an arithmetic or move operation. The appropriate entry in the array is modified for the duration of the program. The next time the program runs, however, the array contains the original entries. Temporary changes can be made permanent if the input records are changed, or if the changed array is written at end of program.

Figure 111 on page 399 shows the specifications for changing the contents of arrays ARFL and ARLI.

The initialization operations CLEAR and RESET can also change the contents of an array during program run-time. CLEAR sets all elements of an array to zero, blank, or '0', depending on the array type (numeric, character or indicator respectively); RESET sets them to their values as assigned at the end of the program initialization step.

Adding Entries to Arrays

Entries can be added to arrays before or during run of the program. The simplest way to add entries to an array is to write additional entries on the input records before the program runs. However, entries that are created by calculation operations or read from an input record can also be added during the running of a program.

Figure 112 shows how entries are added to numeric arrays with the LOKUP and MOVE operations. Such entries are temporary unless they are written in array input records. If these entries are to become a permanent part of the array, they must be written in records and included with the other array file records.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C           25        LOKUPARFL,X                    10   FOUND
C   10                MOVE 500       ARLI,X
C   10                MOVE 30        ARFL,X
```

Figure 111. Changing Array Data with MOVE Operations

The element in ARFL,X that contains 25 to be changed to 30. The corresponding element in ARLI,X is to be changed to 500. The search argument is the numeric literal 25. The search starts at the array element specified in the index X. If the search is successful, the number of the array element is placed in the index field (X), and indicator 10 is set on. The new value in X can then be used to move 500 into the appropriate element in ARLI and to move 30 into the appropriate element in ARFL.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C   01      000       LOKUPARRA,X                    35   FOUND
C   35 01             MOVE NEWA      ARRA,X
C   35 01             MOVE NEWB      ARRB,X
```

Figure 112. Adding Entries to Arrays

The LOKUP operation is conditioned by indicator 01. Indicator 01 is set on when a record containing information in the fields NEWA and NEWB is read. These fields are to be added to the arrays ARRA and ARRB, respectively. To get the entry in the correct place in the array, a search is made to find the first empty array element. Unfilled entries in arrays are filled with zeros. Therefore, the search argument used is 000. When the first 000 entry is found, indicator 35 is set on, and the NEWA and NEWB fields are moved into the array elements ARRA,X and ARRB,X. These new elements become part of arrays ARRA and ARRB.

Array Output

Entire arrays can be written out under RPG/400 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 to be used in positions 19 through 26 of the extension 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 32 through 37 of the output specifications must contain the array name used in the extension specifications.
- Positions 40 through 43 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 23 through 31) can be specified. Zero suppress (position 38), blank-after (position 39), and P = packed/B = binary and L = left/R = right (position 44) entries pertain to every element in the array.

Arrays that are loaded with data by means of input or calculation specifications cannot be written automatically at end of program.

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 38), 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.
Searching	The LOKUP operation can be used to search tables.

LOKUP 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 17) can also be used.

Whenever a table element is found that satisfies the type of search being made (equal, high, low), a copy of that table element is placed in an area in the system. Every time a search is successful, the newly found table element is placed in this area, replacing what was previously there. If the search is not successful, the contents of the area remain the same as they were before the unsuccessful search.

Before a first successful LOKUP, the area in the system reserved for the table element contains the first element of the table.

Resulting indicators reflect the result of the search. If the indicator is on, reflecting a successful search, a copy of the element searched for is in the area.

LOKUP with Two Tables

When two tables are used in a search, only one is actually searched (see Figure 113). When the search condition (high, low, equal) is satisfied, the corresponding elements from both tables are placed in their respective areas in the system and 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. Conditioning indicators can be specified in positions 7 through 17, 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 LOKUP before a successful LOKUP occurs, undesirable results can occur because the contents of the area referenced by the table name contain a previous value.

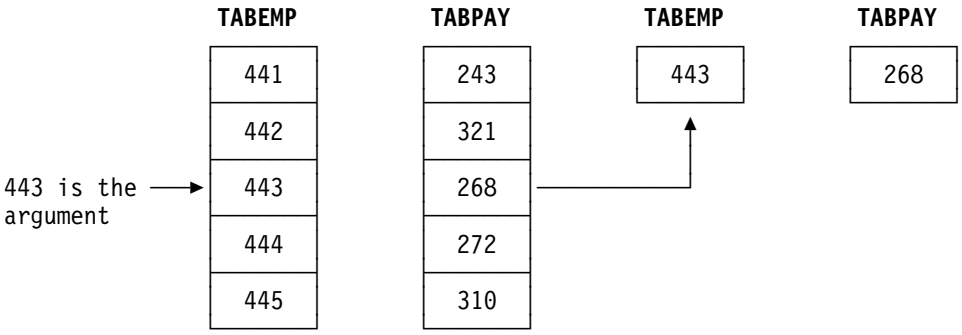


Figure 113. LOKUP Operation for Related Tables

Storage Areas: Tables TABEMP and TABPAY are read into storage. Assume an input record is read with 443 in the EMPNUM field. With 443 as the search argument, the table TABEMP can be searched for an equal entry. When the correct entry is found, the table item 443 is moved into the storage area for TABEMP. At the same

time, the corresponding item 268 is moved into the storage area for TABPAY. The contents of the areas can now to be used in subsequent calculation operations by specifying the appropriate table name. The coding needed to process the LOKUP operation also shows how to refer to the contents of the storage area after a successful LOKUP operation.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C* The LOKUP 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 moved into their storage areas.
C           EMPNUM    LOKUPTABEMP    TABPAY         09
C                         :
C                         :
C                         :
C* If indicator 09 is set on, the contents of the field named
C* HRSWKD are multiplied by the contents of the storage area for
C* TABPAY.  The storage area for TABPAY contains the element found
C* during the last successful LOKUP operation involving TABPAY.
C   09      HRSWKD    MULT TABPAY     AMT    62H
```

Figure 114. Searching for an Equal Entry

Specifying the Table Element Found in a LOKUP Operation

Whenever a table name is used in an operation other than LOKUP, 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 LOKUP operation, the contents of the area in the system are 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 LOKUP operation. In this case the contents of the area in the system are changed by the calculation specification. The corresponding table element in the table in main storage is also changed. In this way the contents of the table can be modified by calculation operations (see Figure 115).

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++++++
C           ARGMNT    LOKUPTABLEA                    20 SEARCH FOR =
C* If element is found multiply by 1.5
C   20      TABLEA    MULT 1.5       TABLEA
```

Figure 115. Specifying the Table Element Found in LOKUP Operations

Chapter 14. 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.

This chapter also contains information about using edit codes to edit externally described files.

Edit Codes

Edit codes are easier to use than edit words. Each of the edit codes edits in a different way according to a predefined pattern.

Edit codes 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 38 of the output specifications for the field to be edited.

Simple Edit Codes

You can use simple edit codes to edit numeric fields without adding 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. Editing for date fields is described in Table 36 on page 404. Positions 20 (date edit) and 21 (decimal notation) of the control specification can be used to alter edit formats.
- The Y edit code is not valid for *YEAR.
- 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.

Table 36. Edited Date Fields

		Control Specification				
UDATE/*DATE	**Edit Code**	**Contents of Position 19**	**Contents of Position 20**	**Contents of Position 21**		
				Blank	**D**	**I/J**
January 30, 1998	Y	Blank	Blank	1/30/98 1/30/1998	30/01/98 30/01/1998	30.01.98 30.01.1998
			- (dash)	1-30-98 1-30-1998	30-01-98 30-01-1998	30-01-98 30-01-1998
		M	Blank	1/30/98 1/30/1998	1/30/98 1/30/1998	1.30.98 1.30.1998
			- (dash)	1-30-98 1-30-1998	1-30-98 1-30-1998	1-30-98 1-30-1998
		D	Blank	30/01/98 30/01/1998	30/01/98 30/01/1998	30.01.98 30.01.1998
			- (dash)	30-01-98 30-01-1998	30-01-98 30-01-1998	30-01-98 30-01-1998
		Y	Blank	98/01/30 1998/01/30	98/01/30 1998/01/30	98.01.30 1998.01.30
			- (dash)	98-01-30 1998-01-30	98-01-30 1998-01-30	98-01-30 1998-01-30

Any character may be specified in position 20 as the separator character. In this example, a dash is used.

Combination Edit Codes

The combination edit codes (1 through 4, A through D, J through Q) punctuate a numeric field.

All of the edit codes suppress leading zeros to the left of the decimal notation except when J is specified in position 21 (decimal notation) of the control specification. (See the end of this section for further information on the J edit code.) 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 notation 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 notation 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 45 through 47 of the output specifications, along with an edit code in position 38 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 in position 18 of the control specification.)

An asterisk constant coded in positions 45 through 47 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 the following coding:

1. Place an edit code in position 38.
2. Place a currency symbol constant one space before the beginning of the edited field.
3. Place '*' in positions 45 through 47 of the line containing the edit code.

When an edit code is used to print an entire array, two blanks precede each element of the array (except the first element).

One of the decimal notation formats for output is a J entry in position 21 of the control specification. If you specify J, the use of commas and periods is reversed; that is, a period is used as the grouping separator character and a comma is used as the decimal notation. The zero-suppression character is in the second position (rather than the first) to the left of the decimal notation. This writes all zero balances and balances with zero values to the left of the comma with one leading zero (0,00 or 0,04). The J entry also overrides any edit codes that might suppress the leading zero. Remember that the decimal positions of the source field determine whether and where a decimal notation is printed.

Another decimal notation format is an I in position 21 of the control specification. If you specify I, a period is used as the grouping separator character and a comma is used as the decimal notation.

Table 37 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 37 (Page 1 of 2). Combination Edit Codes

	Negative Balance Indicator			
Format of Edited Data	**No Sign**	**CR**	**-**	**Floating Minus**
Prints with grouping separator characters, prints zero balance	1	A	J	N

Table 37 (Page 2 of 2). Combination Edit Codes				
	Negative Balance Indicator			
Format of Edited Data	**No Sign**	**CR**	**-**	**Floating Minus**
Prints with grouping separator characters, zero balance suppressed	2	B	K	0
Prints without grouping separator characters, prints zero balance	3	C	L	P
Prints without grouping separator characters, zero balance suppressed	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 *CL 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 38 on page 407 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 39 on page 408 shows how fields look after they are edited.

Table 40 on page 409 shows the effect that the different edit codes have on the same field with a specified end position for output.

Table 38. Edit Codes

				Entry in Column 21 of Control Specification			
Edit Code	**Commas**	**Decimal Point**	**Sign for Negative Balance**	**D or Blank**	**I**	**J**	**Zero Suppress**
1	Yes	Yes	No Sign	.00 or 0	,00 or 0	0,00 or 0	Yes
2	Yes	Yes	No Sign	Blanks	Blanks	Blanks	Yes
3		Yes	No Sign	.00 or 0	,00 or 0	0,00 or 0	Yes
4		Yes	No Sign	Blanks	Blanks	Blanks	Yes
5-9[1]							
A	Yes	Yes	CR	.00 or 0	,00 or 0	0,00 or 0	Yes
B	Yes	Yes	CR	Blanks	Blanks	Blanks	Yes
C		Yes	CR	.00 or 0	,00 or 0	0,00 or 0	Yes
D		Yes	CR	Blanks	Blanks	Blanks	Yes
J	Yes	Yes	- (minus)	.00 or 0	,00 or 0	0,00 or 0	Yes
K	Yes	Yes	- (minus)	Blanks	Blanks	Blanks	Yes
L		Yes	- (minus)	.00 or 0	,00 or 0	0,00 or 0	Yes
M		Yes	- (minus)	Blanks	Blanks	Blanks	Yes
N	Yes	Yes	- (floating minus)	.00 or 0	,00 or 0	0,00 or 0	Yes
O	Yes	Yes	- (floating minus)	Blanks	Blanks	Blanks	Yes
P		Yes	- (floating minus)	.00 or 0	,00 or 0	0,00 or 0	Yes
Q		Yes	- (floating minus)	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  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.

Table 39. 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	00012b[5]	00012b[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
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	00012b[5]	00012b[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 40. 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	
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 or named constant. An edit word 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.

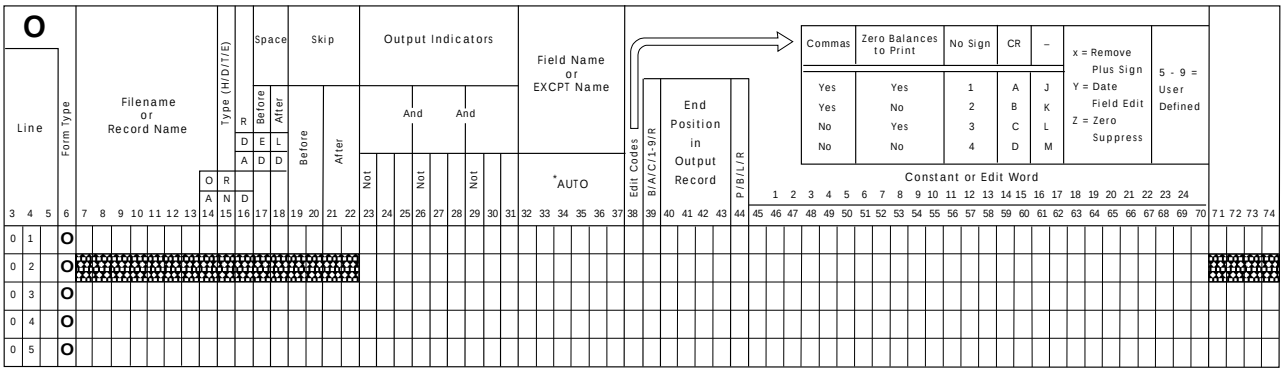
The edit word is used as a template, which the system applies to the source data to produce the output.

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. See “Named Constant Specifications” on page 155 for more information on named constant rules.

How to Code an Edit Word

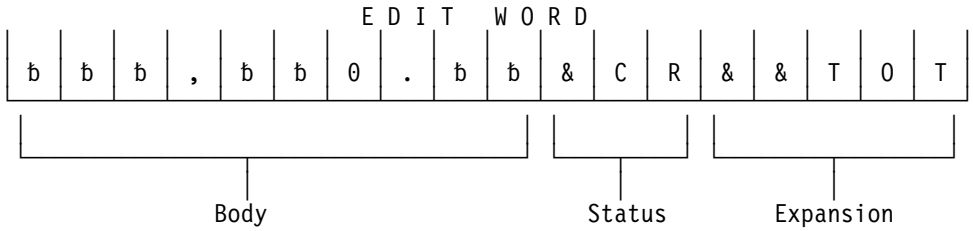
To use an edit word, code the unshaded portion of the output specifications as shown below:



Position	Entry
23-31	Can contain conditioning indicators.
32-37	Contains the name of the numeric field from which the data that is to be edited is taken.
38	*Edit code*. Must be blank, if you are using an edit word to edit the source data.
39	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.
40-43	Identifies the end (rightmost) position of the field in the output record.
45-70	*Edit word*. Can be up to 24 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 45. The edit word, unless a named constant, must begin in column 46.

Parts of an Edit Word

An edit word (coded into positions 45 to 70 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 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: Is replaced with the character from the corresponding position of the source data field specified by the field name in positions 32 through 37 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:
'bbbbbbb'	0000072	bbbbb72
'bbbbbbb.bb'	000000012	bbbbbbbb12
'bbbbbbb.bb'	000000123	bbbbbb1.23

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:
'␣␣␣0␣␣␣␣␣␣'	00000004	␣␣␣␣000004
'␣␣␣0␣␣␣␣␣␣'	012345	␣␣␣␣012345
'␣␣␣0␣␣␣␣␣␣'	012345678	␣␣12345678

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:
'0␣␣␣'	0156	␣156
'0␣␣␣␣'	0156	␣0156

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:
'␣␣␣␣␣␣0.␣␣'	000000001	␣␣␣␣␣␣.01
'␣␣␣␣␣␣0.␣␣'	000000000	␣␣␣␣␣␣.00
'␣␣␣,␣0␣.␣␣'	00000012	␣␣␣␣␣0.12
'␣␣␣,␣0␣.␣␣'	00000123	␣␣␣␣␣1.23
'␣0␣,␣␣␣.␣␣'	00000123	␣␣0,001.23

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:
'*␣␣␣␣␣.␣␣'	000000123	*␣␣␣␣1.23
'␣␣␣␣*␣.␣␣'	000000000	******0.00

Edit Word	Source Data	Appears in Output Record as:
'␣␣␣␣␣*␣.␣␣**'	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:
'␣␣,␣␣␣,␣$0.␣␣'	000000012	␣␣␣␣␣␣␣␣$.12
'␣␣,␣␣␣,␣$0.␣␣'	000123456	␣␣␣␣$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:
'$␣,␣␣␣,␣␣0.␣␣'	000123456	$␣␣␣␣1,234.56
'$␣␣,␣␣␣,Ob0.␣␣'	000000000	$␣␣␣␣␣␣␣␣00.00
'$␣,␣␣␣,*␣␣.␣␣'	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&␣␣␣&NO.&␣␣␣-␣␣␣␣'	4165551212	␣AREA␣416␣NO.␣555-1212

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:
'$␣␣␣␣␣**DOLLARS&␣␣&CTS'	000012345	$****123*DOLLARS␣45␣CTS

Edit Word	Source Data	Appears in Output Record as:
'$bbbbbb**DOLLARS&bb&CTS'	000000006	$*******DOLLARSb06bCTS
'$bbbbbbb&DOLLARS&bb&CTS'	000000006	$bbbbbbbbbbbbbbbbb6bCTS

A date could be printed by using either edit word:

Edit Word	Source Data	Appears in Output Record as:
'bb/bb/bb'	010388	b1/03/88
'Obb/bb/bb'	010389	b01/03/89

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:
'bb0.bb000'	01234	b12.34000
'bb*.bb000'	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:
'bbbbbbb.bb-'	000000123-	bbbbbb1.23-
'bbbbbbb.bb-'	000000123	bbbbbb1.23b

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:
'b,bbb,bb0.bb&30&DAY&CR'	000000123-	bbbbbbbbb1.23b30bDAYbCR
'b,bbb,bb0.bb&30&DAY&CR'	000000123	bbbbbbbbb1.23bbbbbbbbbb

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:
'b,bb0.bb&CR&NET'	000123-	bbbb1.23bCRbNET
'b,bb0.bb&CR&NET'	000123	bbbb1.23bbbbNET

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:
'bb0.bb&SECRET'	12345-	123.45bSECRET
'bb0.bb&SECRET'	12345	123.45bbbbbET
'bb0.bb&CR&&SECRET'	12345	123.45bbbbbSECRET

Summary of Coding Rules for Edit Words

The following rules apply to edit words:

- Position 38 (edit codes) must be blank.
- Positions 32 through 37 (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 45. The edit word itself must begin in position 46.
- 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.

Formatting Edit Words

If you need to show a negative number, include a sign in the edit word. Use either the letters CR or the minus sign (-). These print only for a negative number; however, you must consider the character positions they require when you enter the end position of the field on the output specifications (see Figure 116).

To use a minus sign to indicate a negative number, leave a space between the number and the negative sign, and place an ampersand (&) in the edit word before the minus sign. PERCPL then prints as 25␣-.

If you want to print a currency symbol, you also indicate this in your edit word. Figure 117 shows examples of printing a currency symbol.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
O                        INUMB      7 '     0'
O                        ICOST     18 '$  0.  '
O                        SPRICE    27 '$  0.  '
O                        PERCPL    37 '  CR'
```

Figure 116. Examples of Edit Words on Output Specifications

Editing Externally Described Files

Edit codes must be specified in data description specifications (DDS), instead of the RPG/400 language, to edit output for externally described files. See the *DDS Reference* 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.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
                         SPRICE    27 '$  0.  '     1
                         SPRICE    27 '$  *.  '     2
                         SPRICE    27 '  $0.  '     3
```

Figure 117. Different Edit Words Used on the Same Field

1 To print a currency symbol at the left of the field, put the currency symbol next to the first apostrophe, and then put in the necessary blanks and punctuation. A currency symbol in this position is called a fixed currency symbol. The SPRICE field can look like any of the following (N stands for any number):

```
$NNN.NN
$ NN.NN
$  N.NN
$   .NN
```

2 To avoid blanks between the currency symbol and the first digit when zero suppression occurs, indicate asterisk (*) fill. Instead of using 0 to indicate zero suppression, use the asterisk (*) to indicate that all extra spaces should be filled with asterisks. The `SPRICE` field can look like any of the following (N stands for any number):

```
$NNN.NN
$*NN.NN
$**N.NN
$***.NN
```

3 To have the currency symbol always print next to the leftmost digit, place the $ sign next to the zero-suppress 0 in the edit word. A currency symbol that changes positions depending upon the number of positions that are zero-suppressed is known as a floating currency symbol. The `SPRICE` field can look like any of the following:

```
$NNN.NN
 $NN.NN
  $N.NN
   $.NN
```

Chapter 15. 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/400 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 proc-

essing 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 5 on page 20 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 61 and 62, 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/400 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 118 on page 421.
- All match fields with the same match field values (M1 through M9) should be the same length and type (character or numeric). 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 118 on page 421). The fields are combined according to descending sequence (M9 to M1) of matching field values.
- Match fields values cannot be repeated in a record.
- Match fields can be either character or numeric. However, 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.
- 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 speci-

fied by M1 and M2 fields will not set the MR indicator on (see Figure 118 on page 421).

- 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 43 of the input specifications) cannot be assigned a match field value. However, a field specified as packed (P in position 43 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 119 on page 423).
- 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 121 on page 424 is an example of how match fields are specified.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilenameIPEAF....RlenLK1AIOvKlocEDevice+......KExit++Entry+A....U
FMASTER  IP  E           K        DISK
FWEEKLY  IS  E           K        DISK
```

The files in this example are externally described (E in position 19) and are to be processed by keys (K in position 31).

Figure 118 (Part 1 of 2). Match Fields in Which All Values Match

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
I*                           MASTER FILE
IRcdname+....In..............................Field+L1M1...........
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 118 (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...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+L1M1FrPlMnZr...*
IDISK    AB  01   1 C1
I       OR   02   1 C2
I       OR   03   1 C3
I                                          1  100EMPNO   M1
I                                         11  150DUMMY   M2
I                                         11  150DEPT    M202
I                                         16  200DEPT    M203
```

Figure 119. Match Fields with Dummy Match Field

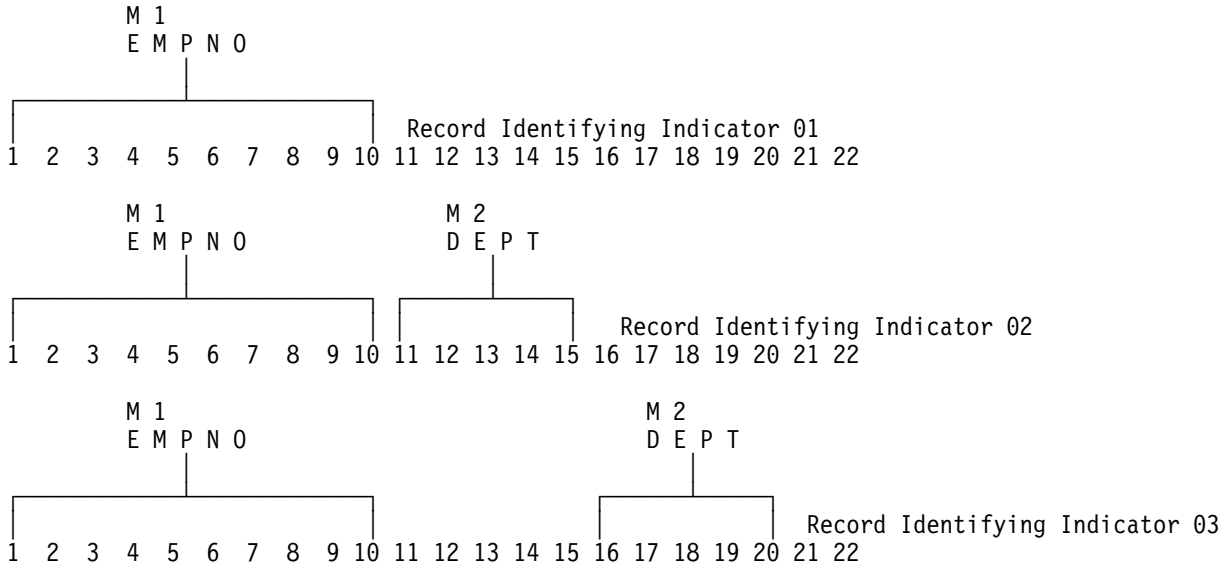


Figure 120. Match Fields with a Dummy M2 Field

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 an entry in positions 63 and 64. 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...
FFilenameIPEAF....RlenLK1AIOvKlocEDevice+......KExit++Entry+A....U1.
FPRIMARY IPEAF      64           DISK
FFIRSTSECIS AF      64           DISK
FSECSEC  IS AF      64           DISK

*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+L1M1..........
IPRIMARY AA  01   1 CP   2NC
I                                    2   3 MATCH   M1
I*
I        BB  02   1 CP   2 C
I                                    2   3 NOM
I*
IFIRSTSECAB  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
```

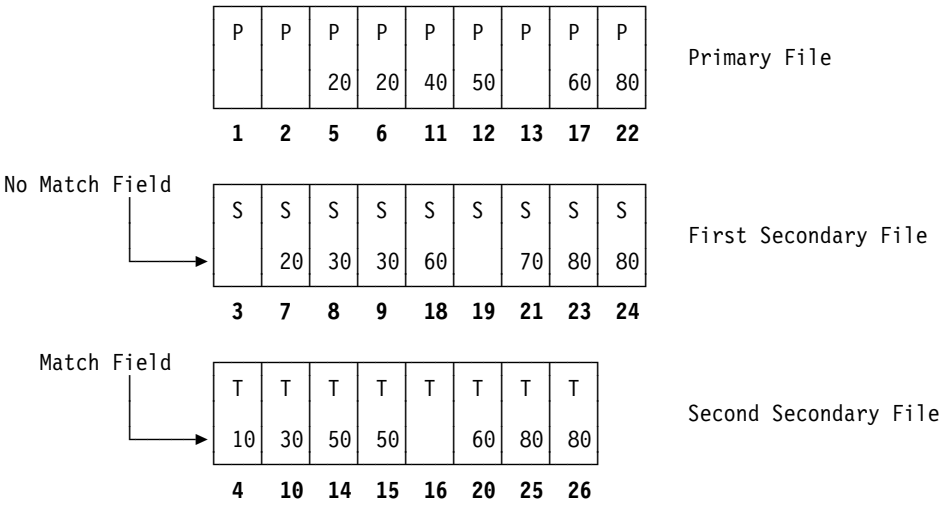
Figure 121. Match Field Specifications for Three Disk Files

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 122 through Figure 123 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 122. Normal Record Selection from Three Disk Files

Table 41 (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

Table 41 (Page 2 of 2). Normal Record Selection from Three Disk Files			
Cycle	**File Processed**	**Indicators On**	**Reason for Setting Indicator**
21	FIRSTSEC	03	First secondary low; no primary match
22	PRIMARY	01, MR	Primary matches both secondary files
23	FIRSTSEC	03, MR	First secondary matches primary
24	FIRSTSEC	03, MR	First secondary matches primary
25	SECSEC	05, MR	Second secondary matches primary
26	SECSEC	05, MR	Second secondary matches primary

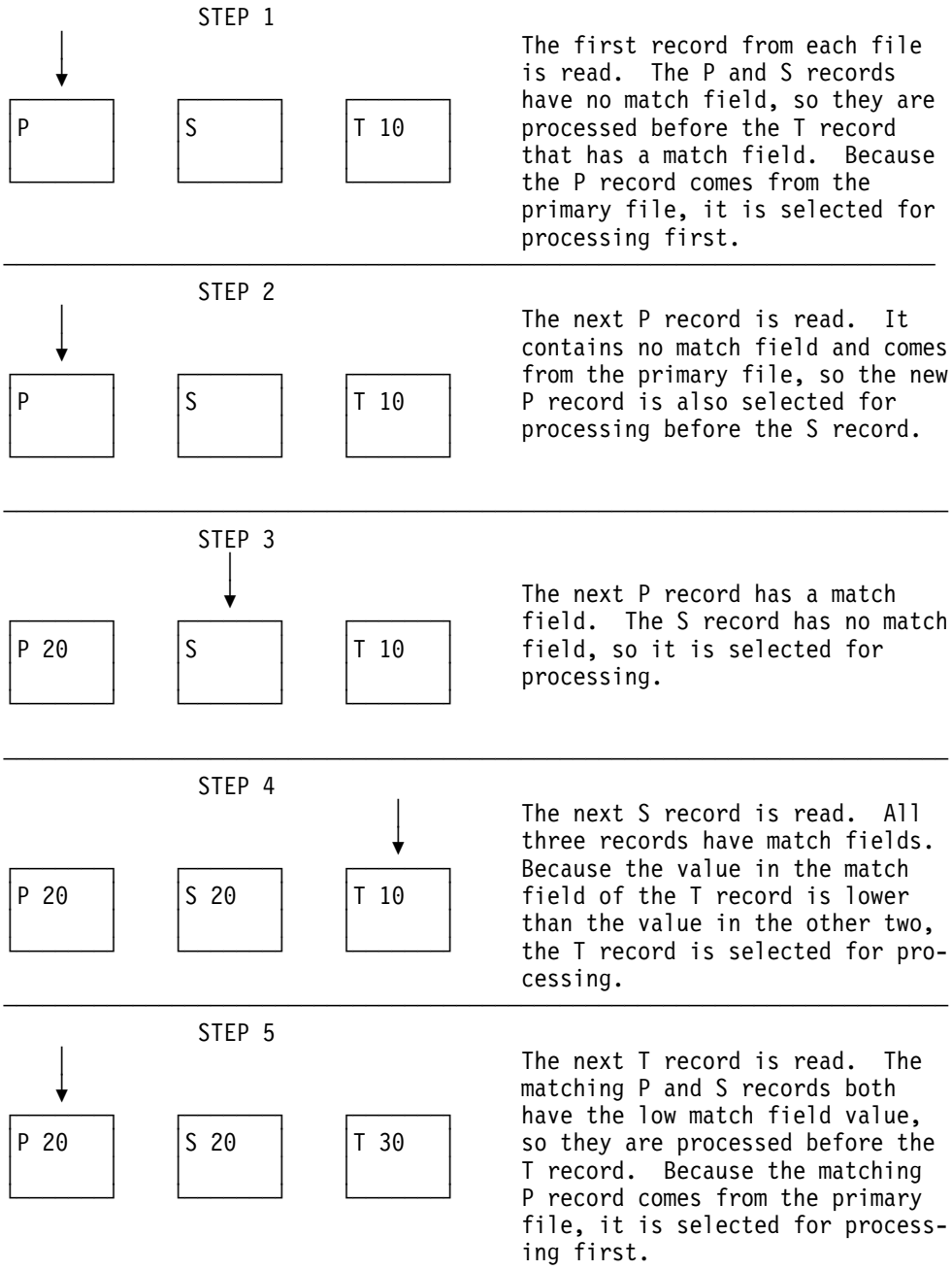


Figure 123 (Part 1 of 2). Normal Record Selection from Three Disk Files

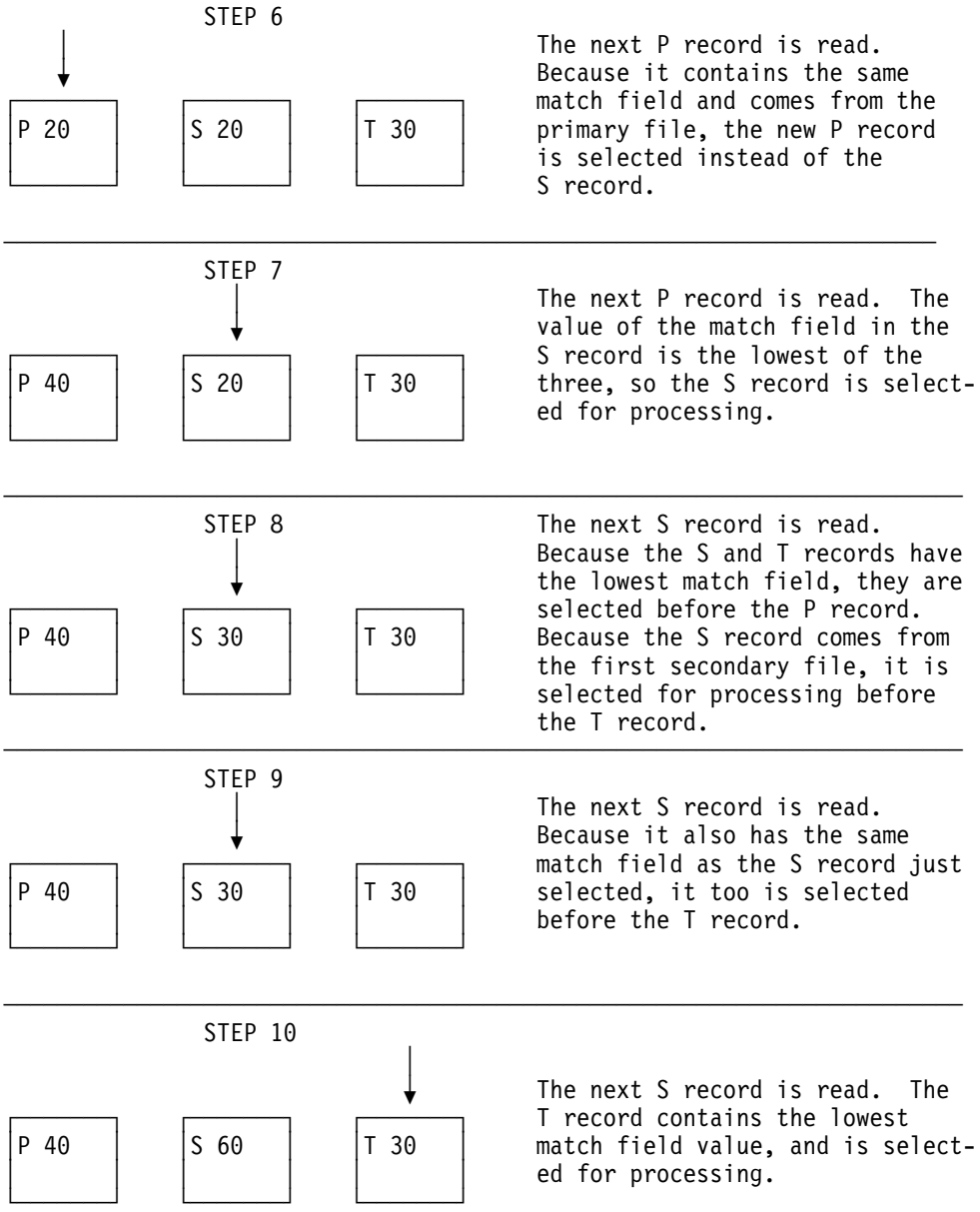


Figure 123 (Part 2 of 2). Normal Record Selection from Three Disk Files

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. The calculation operations affected by the alternate collating sequence are ANDxx, COMP, CABxx, CASxx, DOUxx, DOWxx, IFxx, ORxx, and WHxx. 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 LOKUP and SORTA operations 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.

Specifying an Alternate Collating Sequence

To specify that an alternate collating sequence is to be used, enter an S in position 26 in the control specification. Transcribe the sequence changes into the correct record format for entry into the system. These records, called the alternate collating sequence table records, must be entered after the RPG/400 specifications and, if used, after the file translation table records. However, the alternate collating sequence table records must be entered before arrays and tables loaded at compile time.

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 441 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.)
7-8	Leave these positions blank.
9-10	Enter the hexadecimal value for the character whose normal sequence is being changed.
11-12	Enter the hexadecimal value of the character replacing the character whose normal sequence is being changed.
13-16 17-20 21-24 ... 77-80	All groups of four beginning with position 13 are used in the same manner as positions 9 through 12. 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 **b (b = 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 or output data occurs *after* any editing or field selection has taken place.

If file translation is used to translate data in an update file, each record must be written before the next record is read.

Remember the following when specifying file translation:

- File translation can be specified for data in array or table files (T in position 16 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`, `REDPE`, `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.

Specifying File Translation

To specify file translation, enter an F in position 43 of 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 **b (b = blank) in positions 1 through 3. 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 *FILES␣␣ (␣ represents a blank) to indicate that all files are to be translated. Complete the file translation table record beginning with positions 9 and 10. If *FILES␣␣ 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 9 and 10. The *FILES␣␣ entry is *not* made in positions 1 through 8 when a specific file is to be translated.
9-10	Enter the hexadecimal value of the character to be translated from on input or to be translated to on output.
11-12	Enter the hexadecimal equivalent of the internal character the RPG/400 language works with. It will replace the character in positions 9 and 10 on input and be replaced by the character in positions 9 and 10 on output.
13-16 17-20 21-24 ... 77-80	All groups of four beginning with position 13 are used in the same manner as positions 9 through 12. 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 16 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.

Specifying the Files

File translation table records must be formatted as follows:

Record Position	Entry
1-7	*EQUATE
8	Leave this position blank.
9-80	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
8	Leave this position blank.
9-10	Enter the hexadecimal value of the character to be translated from on input or to be translated to on output.
11-12	Enter the hexadecimal equivalent of the internal character the RPG/400 language works with. It will replace the character in positions 9 and 10 on input and be replaced by the character in positions 9 and 10 on output.
13-16 17-20 21-24 ... 77-80	All groups of four beginning with position 13 are used the same way as positions 9 through 12. 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 8 on the next record and continue as before with the translation pairs in positions 9 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 **b (b = blank) in positions 1 through 3. The remaining positions in this record can be used for comments.

Special File

SPECIAL in positions 40 through 46 of the file description specifications allows you to specify an input and/or output device that is not directly supported by the RPG/400 language. The input and output operations for the file are controlled by a user-written routine. Positions 54 through 59 of the file description specifications line that contains SPECIAL in positions 40 through 46 must contain the name of the user-written routine. This user-written routine is called to open the file, read and write the records, and close the file. A parameter list is created for use by the user-written routine. The parameter list contains an option code parameter (option), a return status parameter (status), an error-found parameter (error), and a record area parameter (area). This parameter list is accessed by the RPG/400 compiler and by the user-written routine; it cannot be accessed by the RPG/400 program that contains the SPECIAL file.

The following describes the parameters in this RPG/400-created parameter list:

Option The option parameter is a 1-position character field that indicates the action the user-written routine is to process. Depending on the operation being processed on the SPECIAL file (OPEN, CLOSE, READ, WRITE,

DELET, UPDAT), one of the following values is passed to the user-written routine:

Value Passed	Description
O	Open the file.
C	Close the file.
R	Read a record and place it in the area defined by the area parameter.
W	A record has been placed in the area defined by the area parameter; it is to be written out.
D	Delete the record.
U	The record is an update of the last record read.

Status The status parameter is a 1-position character field that indicates the status of the user-written routine when control is returned to the RPG/400program. Status must contain one of the following return values when the user-written routine returns control to the RPG/400 program:

Return Value	Description
0	Normal return. The requested action was processed.
1	The input file is at end of file, and no record has been returned. If the file is an output file, this return value is an error.
2	The requested action was not processed; error condition exists.

Error The error parameter is a 5-digit zoned numeric field with zero decimal positions. If the user-written routine detects an error, the error parameter contains an indication or value representing the type of error. The value is placed in the first five positions of location *RECORD in the file information data structure (INFDS) when the status parameter contains 2.

Area The area parameter is a character field whose length is equal to the record length associated with the SPECIAL file. This field is used to pass the record to or receive the record from the RPG/400 program.

You can add additional parameters to the RPG/400-created parameter list. To do this, specify PLIST in positions 54 through 59 and the name of the PLIST in positions 60 through 65 of a file description specifications continuation line for the SPECIAL file (see Figure 124 on page 433). Then use the PLIST operation in the calculation specifications to define the additional parameters.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilenameIPEAF....RlenLK1AIOvKlocEDevice+......KExit++Entry+A....U
F*
F* The file EXCPTN is assigned to the device SPECIAL.  The I/O
F* operations from the SPECIAL device are controlled by the user
F* written routine USERIO.
FEXCPTN  I   F                    SPECIAL      USERIO
F                                             KPLIST SPCL
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C*
C* The parameters specified for the programmer-defined PLIST SPCL
C* are added to the end of the RPG/400-created parameter list for the
C* SPECIAL device.  The programmer-specified parameters can be
C* accessed by the user RPG/400 program and the user-written routine;
C* whereas the RPG/400-created parameter list can be accessed only by
C* internal RPG/400 logic and the user-written routine.
C           SPCL      PLIST
C                     PARM           FLD1
C                     PARM           FLD2
C                     PARM           FLD3
```

Figure 124. SPECIAL Device

The user-written routine, the name of which is specified in positions 54 through 59 of the file description specifications for the SPECIAL file, must contain an entry parameter list that includes both the RPG/400 compiler-created parameters and the user-specified parameters.

If the SPECIAL file is specified as a primary file, the user-specified parameters may need to be initialized before the first primary read. This can be done with a factor 2 entry on the PARM statements or by the specification of a compile-time array or an array element as a parameter.

Table 42. Valid File Operations for a SPECIAL File

File Description Specifications Positions		Calculation Specifications Positions
15	**16**	**28-32**
I	P/S	CLOSE, FEOD
C	P/S	WRITE, CLOSE, FEOD
U	P/S	UPDAT, DELET, CLOSE, FEOD
O		WRITE, OPEN, CLOSE, FEOD
I	F	READ, WRITE, OPEN, CLOSE, FEOD
C	F	READ, WRITE, OPEN, CLOSE, FEOD
U	F	READ, UPDAT, DELET, OPEN, CLOSE, FEOD

Chapter 16. Using Double-Byte Character Set (DBCS) Data in RPG/400 Programs

This chapter describes enhancements to the RPG/400 programming language which are useful for handling DBCS data. It describes where to use DBCS data in an RPG/400 program, and considerations for working with DBCS data in the RPG/400 language.

Where You Can Use DBCS Data in RPG/400 Programs

In an RPG/400 program, you can use DBCS data:

- In any comment statement (a statement with an asterisk in position 7)
- In the comment field of the extension specification (positions 58-74)
- In the comment field of the calculation specification (positions 60-74)
- In literals and constants (including named constants)
- As data in any character field, subfield, array, or table.

You may mix DBCS data with alphanumeric data. The RPG/400 language treats DBCS data in the same way as Single Byte Character Set (SBCS) data.

The RPG/400 language uses the value 10 in the *0UT field in the file information data structure (INFDS) to indicate that the national language output capability of the device makes it possible for you to use DBCS data with it.

How to Work with DBCS Literals in RPG/400 Programs

Transparent Literals and Constants

A constant or literal is transparent if the constant or literal:

- Begins with an apostrophe followed by a shift-out character.
- Ends with a shift-in character followed by an apostrophe.
- Contains no embedded shift control characters.
- Contains only DBCS data.
- If you are specifying a transparent constant using hexadecimal literals, you must specify the shift-out and shift-in characters in hexadecimal.

The following is an example of a DBCS named constant. The constant is continued by placing a hyphen instead of an apostrophe at the end of each continued line. When concatenated, the internal SO/SI characters will be dropped and only the starting SO and ending SI will remain. For more information on named constants see “Named Constant Specifications” on page 155.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
I..............Namedconstant+++++++++C.........Fldnme.............
I              'oK1K2K3i-             C         DOUBLE
I              'oK4K5i'
```

Using DBCS data in a constant or literal that is not transparent may produce unwanted results. The DBCS data may include a character representing an apos-

trophe. This character ends your constant or literal where you did not expect it to end. With a transparent constant or literal, the RPG/400 language ensures this result does not occur.

If you specify 1 in position 57 of the control specification, transparent literals and constants are scanned. Your literal or constant is checked to be transparent, when an apostrophe followed by a shift-out character is found. If it is not, a warning message is issued on the compiler listing. The literal or constant is then treated as a literal or constant that is not transparent.

Note: If you specify the shift-out and shift-in characters in a hexadecimal literal, it will not be considered a transparent literal, and will not be checked if you specify a 1 in column 57 of the control specification.

All RPG/400 restrictions on the length of constants or literals apply to transparent constants and literals, including the apostrophes and control characters.

You may use transparent constants and literals in any of the places you use constants or literals:

- In factor 1 and factor 2 of the calculation specifications. However, it may not be meaningful to use DBCS data in all of these places. For example, a move zone operation with a transparent literal in factor 2 will move only the zone of the shift control character. See the description of each operation code in Chapter 11, “Operation Codes” on page 185 to decide if it is appropriate to use a transparent literal.
- As a constant in the constant or edit word field (positions 45-70) of the output specifications.

Additional Considerations for Using DBCS Data

Fields with DBCS data from externally defined files are defined by the RPG/400 language as character fields. These fields are treated in the same manner as other character fields. However, on the cross-reference listing, the DBCS fields are indicated by `IGC` in the key field section, or by `G` in the field section.

Example of Coding DBCS Data in an RPG/400 Program

Here are some RPG/400 specifications that include DBCS data.

In this figure, `'Kn'` represents a DBCS character, `'o'` represents shift-out character, and `'i'` represents a shift-in character.

This is an EBCDIC example.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
H........1..CDYI....S..............1.F.............................
H                                                  1
```

You may check transparent literals and constants by specifying a 1 in column 57 of the control specification.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
E*
E*oK1K2K3K4iABoK1K2K3K4iCDEFGHoK1K2K3K4K5K6K7K8K9i


*...1....+....2....+....3....+....4....+....5....+....6....+....7...
E....FromfileTofile++Name++N/rN/tbLenPDSArrnamLenPDSComments
E                    ARR         4  5 0             oK1K2i ARRoK1i
E*
```

DBCS data may be mixed with alphanumeric data in comments.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments++++++
C           'oK1K2K3i'COMP FLDA                 161718
C                     MOVE 'oK1K2K3i'FLDB
C           LABELA    TAG                          'oK1K2K3K4i'
C*
```

Transparent constants and literals are DBCS data enclosed in apostrophes.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
OName++++DFBASbSaN01N02N03Field+YBEnd+PConstant/editword+++++++++...
O                                    7 'oK1K2i'
O                                   48 'oK1K2K3K4K5K6K7i'
O                         FLDC      57 '$  0. oK1K2K3i'
```

You may use DBCS constants in your output specifications.

Error status code 450 will be returned if shift-out and shift-in characters are not properly used.

Here are some sample entries from a compile listing. FLDK1 is a DBCS field in an externally described file.

```
                        PHYSICAL      LOGICAL
             FILE/RCD   FIELD         FIELD      ATTRIBUTES


                        FLDK1                    IGC      6
                        AFLD                     CHAR     3
```

Figure 125. Sample Key Field Information

```
           FIELD       ATTR       REFERENCE (M=MODIFIED D=DEFINED)


           FLDK1       G(6)        1000010D
           AFLD        A(2)        1000004D
```

Figure 126. Sample Cross-Reference Listing

Appendix A. RPG/400 Restrictions

Table 43. RPG/400 Restrictions

Function	Restriction
AN/OR lines (positions 7 and 8 of calculation specifications)	Maximum of 7 per operation.
Arrays and tables	Maximum of 200 per program.
Array/table input record length for compile time	Maximum length is 80.
Character field length	Maximum length is 256.
Control fields (position 59 and 60 of input specifications) length	Maximum length is 256.
Data structure length	Maximum of 9999.
Data structure occurrences (number of)	Maximum of 9999 per data structure.
Edit Word	Maximum length of 24 for literals or 115 for named constants.
Elements in an array/table (positions 36 through 39 of extension specifications)	Maximum of 9999 per array/table.
File	Maximum of 50 per program.
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 256 for character named constant, 512 for hexadecimal named constant, and 30 digits with 9 decimal positions for numeric named constant.
Overflow indicator	Only 1 unique overflow indicator can be specified per printer file.
Parameters	Maximum of 255
Primary file (P in position 16 of file description specifications)	Maximum of 1 per program.
Printer file (PRINTER in positions 40 through 46 of file description specifications)	Maximum of 8 per program.
Printing lines per page	Minimum of 2; maximum of 112.
Program status data structure	Only 1 allowed per program.
Record address file (R in position 16 of file description specifications)	Only 1 allowed per program.
Record length for program described file (positions 24 through 27 of file description specifications)	Maximum length is 9999. [1]
Structured groups (see levels of nesting)	
Subroutines	Maximum of 254 per program.
Tables (see arrays)	

[1] Any device record size restraints override this value.

Appendix B. EBCDIC Collating Sequence

EBCDIC Collating Sequence

Table 44 (Page 1 of 4). EBCDIC Collating Sequence

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
65	␣	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 44 (Page 2 of 4). EBCDIC Collating Sequence

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
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 44 (Page 3 of 4). EBCDIC Collating Sequence

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
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 44 (Page 4 of 4). EBCDIC Collating Sequence

Ordinal Number	Symbol	Meaning	Decimal Representation	Hex Representation
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

- *Publications Guide, GC41-9678*, GC41-9678
  Short Name: *Publications Guide*
- *Data Management Guide, SC41-9658*, SC41-9658
  Short Name: *Data Management Guide*
- *Data Description Specifications Reference, SC41-9620*, SC41-9620
  Short Name: *DDS Reference*
- *Distributed Data Management Guide, SC41-9600*, SC41-9600
  Short Name: *DDM Guide*
- *Database Guide, SC41-9659*, SC41-9659
  Short Name: *Database Guide*
- *Communications: Intersystem Communications Function Programmer's Guide, SC41-9590*, SC41-9590
  Short Name: *ICF Programmer's Guide*
- *Programming: GDDM Programming Reference, SC41-0537*, SC41-0537
  Short Name: *GDDM Programming Reference*
- *Programming: GDDM Programming Guide, SC41-0536*, SC41-0536
  Short Name: *Graphical Data Display Manager (GDDM)*
- *Programming: System/38 Environment Programmer's Guide and Reference, SC41-9755*, SC41-9755
  Short Name: *System/38 Environment Programmer's Guide/Reference*
- *System Operation*, SC41-3203
  Short Name: *System Operation*
- *Systems Application Architecture* Structured Query Language/400 Reference, SC41-9608*, SC41-9608
  Short Name: *Programming: Structured Query Language Reference*
- *Languages: RPG Reference Summary, SX09-1164*
  Short Name: *RPG Reference Summary*
- *RPG Debugging Template*, GX21-9129
  Short Name: *RPG Debugging Template*
- *RPG/400 User's Guide*, SC09-1816
  Short Name: *RPG/400 User's Guide*

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T