z/OS
2.5

DFSORT Application Programming Guide

SC23-6878-50

Note

Before using this information and the product it supports, read the information in “Notices” on page 853.

This edition applies to Version 2 Release 5 of z/OS® (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2022-03-28

Contents

Figures

Tables

About this document

This document is intended to help you to sort, merge, and copy data sets using DFSORT. This document is not designed to teach you how to use DFSORT, but is for programmers who already have a basic understanding of DFSORT, and need a task-oriented guide and reference to its functions and options. If you are a new user, then you should read *z/OS DFSORT: Getting Started* first.

z/OS DFSORT: Getting Started is a self-study guide that tells you what you need to know to begin using DFSORT quickly, with step-by-step examples and illustrations.

How to use this document

The various sections of this document present related information grouped according to tasks you want to do. The first three chapters of the document explain what you need to know to invoke and use DFSORT's primary record-processing functions. The remaining chapters explain more specialized features. The appendixes provide specific information about various topics.

- Chapter 1, “Introducing DFSORT,” on page 1, presents an overview of DFSORT, explaining what you can do with DFSORT and how you invoke DFSORT processing. It describes how DFSORT works, discusses data set formats and limitations, and explains how installation defaults can affect your DFSORT applications.
- Chapter 2, “Invoking DFSORT with Job Control Language,” on page 27, explains how to use job control language (JCL) to run your DFSORT jobs. It explains how to code JOB, EXEC, and DD statements, and how you can use cataloged procedures and EXEC PARM options to simplify your JCL and override installation defaults.
- Chapter 3, “Using DFSORT program control statements,” on page 77, presents the DFSORT control statements you use to sort, merge, and copy data. It explains how to filter your data so you work only with the records you need, how to edit data by reformatting and summing records, and how to produce multiple output data sets and reports. It explains how to write statements that direct DFSORT to use your own routines during processing.
- Chapter 4, “Using a JOINKEYS application for joining two files,” on page 439 explains how you can perform various types of "join" applications on two files (F1 and F2) by one or more keys with DFSORT. It explains the JCL and control statements you can use for JOINKEYS applications.
- Chapter 5, “Using your own user exit routines,” on page 467, describes how to use DFSORT's program exits to call your own routines during program processing. You can write routines to delete, insert, alter, and summarize records, and you can incorporate your own error-recovery routines.
- Chapter 6, “Invoking DFSORT from a program,” on page 517, describes how you use a system macro instruction to initiate DFSORT processing from your own assembler program. It also lists specific restrictions on invoking DFSORT from PL/I and COBOL.
- Chapter 7, “Using ICETOOL,” on page 537, describes how to use the multi-purpose DFSORT utility ICETOOL. It explains the JCL and operators you can use to perform a variety of tasks with ICETOOL.
- Chapter 8, “Using symbols for fields and constants,” on page 685, explains how to define symbols and use them in DFSORT control statements and ICETOOL operators.
- Chapter 9, “Using extended function support,” on page 717, explains how to use the Extended Function Support (EFS) interface to tailor control statements, to handle user-defined data types and collating sequences, and to have DFSORT issue customized informational messages during processing.
- Chapter 10, “Improving efficiency,” on page 747, recommends ways with which you can maximize DFSORT processing efficiency. This chapter covers a wide spectrum of improvements you can make, from designing individual applications for efficient processing at your site to using DFSORT features such as Hipersorting, dataspace sorting, and ICEGENER.
- Chapter 11, “Examples of DFSORT job streams,” on page 765, contains annotated example job streams for sorting, merging, and copying records.

- Appendix A, “Using work space,” on page 797, explains main storage considerations and how to estimate the amount of intermediate storage you might require when sorting data.
- Appendix B, “Specification/override of DFSORT options,” on page 805, contains a series of tables you can use to find the order of override for similar options that are specified in different sources.
- Appendix C, “Data format descriptions,” on page 823, gives examples of the assembled data formats.
- Appendix D, “EBCDIC and ASCII collating sequences,” on page 839, lists the collating sequences from low to high order for EBCDIC and ASCII characters.
- Appendix E, “DFSORT abend processing,” on page 847, describes the ESTAE recovery routine for processing abends, and the Checkpoint/Restart facility.

Required product knowledge

To use this document effectively, you should be familiar with the following information:

- Job control language (JCL)
- Data management
- Tape and disk hardware

You should also be familiar with the information presented in the following related documents:

Table 1. Related documentation
Related Documentation
z/OS DFSORT Messages, Codes and Diagnosis Guide
z/OS MVS JCL Reference
z/OS MVS JCL User's Guide
z/OS DFSMS Using Data Sets
z/OS DFSMS Using Magnetic Tapes

Referenced documents

This document refers to the following documents:

Table 2. Reference documentation
Reference documentation
z/OS DFSMSdfp Checkpoint/Restart
z/OS DFSMS Macro Instructions for Data Sets
z/OS DFSMS Using Data Sets
z/OS MVS JCL Reference
z/OS MVS JCL User's Guide
z/OS MVS Programming: Assembler Services Reference IAR-XCT
z/OS MVS Programming: Assembler Services Guide
z/OS Install/Migration page (www.ibm.com/systems/z/os/zos/installation)
z/OS UNIX System Services User's Guide

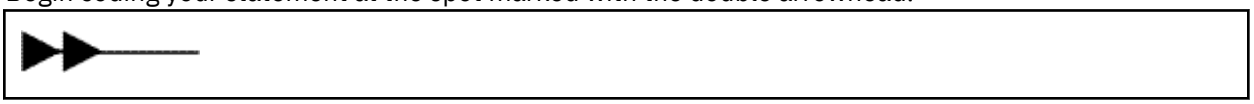
The *z/OS DFSORT Application Programming Guide* is a part of a more extensive DFSORT library. These documents can help you work with DFSORT more effectively.

Table 3. Documents to help you work more effectively with DFSORT.	
Task	**Documentation**
Planning for and Customizing DFSORT	*z/OS DFSORT Installation and Customization*
Learning to use DFSORT	*z/OS DFSORT: Getting Started*
Diagnosing DFSORT Failures and Interpreting DFSORT Messages	*z/OS DFSORT Messages, Codes and Diagnosis Guide*
Tuning DFSORT	*z/OS DFSORT Tuning Guide*

Notational conventions

The syntax diagrams in this document are designed to make coding DFSORT program control statements simple and unambiguous. The lines and arrows represent a path or flowchart that connects operators, parameters, and delimiters in the order and syntax in which they must appear in your completed statement. Construct a statement by tracing a path through the appropriate diagram that includes all the parameters you need, and code them in the order that the diagram requires you to follow. Any path through the diagram gives you a correctly coded statement, if you observe these conventions:

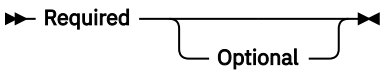
- Read the syntax diagrams from left to right and from top to bottom.
- Begin coding your statement at the spot marked with the double arrowhead.



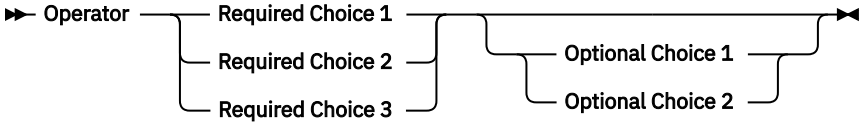
- A single arrowhead at the end of a line indicates that the diagram continues on the next line or at an indicated spot.

  A continuation line begins with a single arrowhead.

- Strings in upper-case letters, and punctuation (parentheses, apostrophes, and so on), must be coded exactly as shown.
  - Semicolons are interchangeable with commas in program control statements and the EXEC PARM string. For clarity, only commas are shown in this document.
- Strings in all lowercase letters represent information that you supply.
- Required parameters appear on the same horizontal line (the main path) as the operator, while optional parameters appear in a branch below the main path.

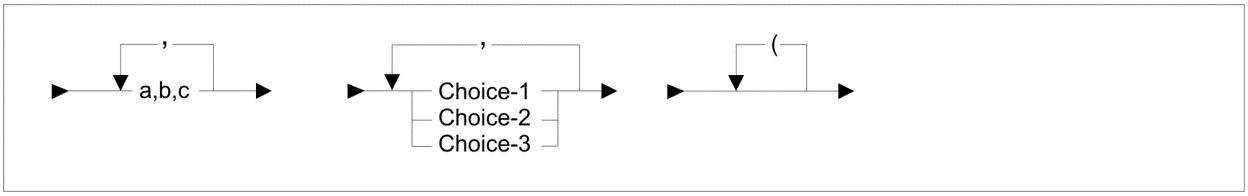


- Where you can make one choice between two or more parameters, the alternatives are stacked vertically.



  If one choice within the stack lies on the main path (as in this example, left), you *must* specify one of the alternatives. If the stack is placed below the main path (as in this example, right), then selections are optional, and you can choose either one or none of them.

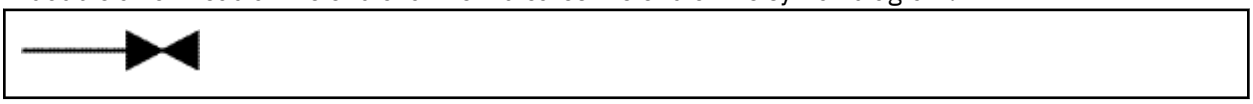
- The repeat symbol shows where you can return to an earlier position in the syntax diagram to specify a parameter more than once (see the first example later in this section), to specify more than one choice at a time from the same stack (see the second example later in this section), or to nest parentheses (see the third example later in this section).



  Do not interpret a repeat symbol to mean that you can specify incompatible parameters. For instance, do not specify both **ABEND** and **NOABEND** in the same **EXEC** statement, or attempt to nest parentheses incorrectly.

  Use any punctuation or delimiters that appear within the repeat symbol to separate repeated items.

- A double arrowhead at the end of a line indicates the end of the syntax diagram.



How to send your comments to IBM

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead "If you have a technical problem" on page xxv.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function
If your comment or question is about z/OS itself, submit a request through the IBM RFE Community (www.ibm.com/developerworks/rfe/).

Feedback on IBM® Documentation function
If your comment or question is about the IBM Documentation functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Documentation Support at ibmdocs@us.ibm.com.

Feedback on the z/OS product documentation and content
If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrcfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS DFSORT Application Programming Guide, SC23-6878-50
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive authority to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

Summary of changes

z/OS Version 2 Release 5 summary of changes

This document contains information that was previously presented in *z/OS DFSORT Application Programming Guide*, SC23-6878-40, which supported z/OS Version 2 Release 4.

New information

March 2022 refresh

- None.

Prior to March 2022 refresh

- None.

Changed information

March refresh

- Added note for JOINKEYS to “Regular expressions application notes” on page 109 (APAR PH38775).
- Added note to the last example to “Regular expressions application examples” on page 109 (APAR PH38775).

Prior to March 2022 refresh

- None.

z/OS Version 2 Release 4 summary of changes

This document contains information that was previously presented in *z/OS Application Programming Guide*, SC23-6878-30, which supported z/OS Version 2 Release 3.

The following sections summarize the changes to that information.

The messages ICE027A, ICE114A and ICE161A will be issued for the following additional error situation:

Invalid use of Regular expressions.

Any automated actions based on the presence of this message should be evaluated.

New information

This edition includes the following new enhancements:

Integrated Accelerator for Z Sort (September 2020)

DFSORT and DFSORT's ICETOOL are enhanced to exploit the IBM Integrated Accelerator for Z Sort feature termed as "ZSORT" to reduce the CPU costs and improve the elapsed time for eligible workloads.

Regular expressions

DFSORT now supports Regular expressions in the following DFSORT comparison operands: COND, INCLUDE, OMIT, BEGIN, END, WHEN and TRLID. This support allows DFSORT users to use Regular expressions in their batch jobs for additional filtering capabilities. Regular expressions contain a series of characters that define a pattern of text to be matched, which allows for more robust filtering capabilities.

Unicode comparisons

DFSORT now supports the usage of Unicode data formats (UTF-8, UTF-16 and UTF-32) in the following comparison operands: COND, INCLUDE and OMIT.

ASCII free format numeric

DFSORT now supports two new ASCII free format numeric data formats (AUF and ASF) in SORT and MERGE operands. The new ASCII free format numeric data formats (AUF and ASF) are also now supported in the following comparison operands: COND, INCLUDE and OMIT.

Encryption support

DFSORT now supports reading and writing to sequential extended format data sets, VSAM extended format data sets (KSDS, ESDS, RRDS, VRRDS) and PDSE data sets.

Message changes

The following changes are made to z/OS Version 2 Release 4 (V2R4).

The following messages are new, changed, or no longer issued for z/OS DFSORT Installation and Customization in z/OS V2R4.

The text for existing DFSORT messages has been changed. Any automated actions based on the presence of these messages should be evaluated.

ICE000I
ICE039A
ICE201I

New

The following messages are new.

ICE167A
ICE267I
ICE268I
ICE269A
ICE399I

New Reserved Words for Symbols

The following are new DFSORT/ICETOOL reserved words which are no longer allowed as symbols: AUF and ASF. If you used any of these words as a symbol previously, you must change them. For example, if you used ASF, you can change it to asf.

Summary of changes for z/OS Version 2 Release 3

This document contains information that was previously presented in *z/OS DFSORT Application Programming Guide*, SC23-6878-01, which supported z/OS Version 2 Release 2.

The following sections summarize the changes to that information.

The message ICE290A is will be issued for the following additional situation:

- The E15 or E35 exit was specified with block support, but the required address of the user block list parameter area was not present in the 64-bit invocation parameter list or the block list type is invalid in the user block list parameter area.

Exploit zHPF for DFSORT work data sets (November 2018):

- Enable DFSORT to create channel programs that exploit System z High Performance FICON (zHPF) with DFSORT work data sets to provide I/O performance improvements without the need for application changes.

Any automated actions based on the presence of this message should be evaluated.

New information

This edition includes the following new enhancements:

Sort/Merge of Unicode data:

DFSORT now supports the Sorting and Merging of Unicode data. This support allows users to:

- SORT/MERGE Unicode Data with control field length of 1 to 450 Unicode characters for UTF-8 format data.
- SORT/MERGE Unicode Data with control field length of 1 to 450 Unicode characters for UTF-16 format data.
- SORT/MERGE Unicode Data with control field length of 1 to 450 Unicode characters for UTF-32 format data.

E15/E35 Block support

E15/E35 exits have been enhanced to support the transfer of block of records between DFSORT and the E15/E35 exits.

Performance Improvements

DFSORT is enhanced to provide performance improvements for several DFSORT functions that generate code at run time. The enhancements will improve performance in both CPU and elapsed times. The design of performance improvements are based on the concept of storing data into the "256 byte area for the Instruction Cache". DFSORT's design has the ability to write data directly into the instruction cache stream. Modern processors can also access a portion of this cache and may invalidate the instructions stored by DFSORT. In the modern processors like z13, the instruction cache stream is considered to be any location within the same 256-byte block of storage where an instruction resides, regardless of whether that location contains an actual instruction. This means that storing data into the same 256 byte block of an instruction can cause an impact on performance.

The following functions will see improvements in both CPU and elapsed times.

- INCLUDE COND Processing
- OMIT COND processing
- SORT statement with more than 5 keys

Message changes

The following changes are made to z/OS Version 2 Release 3 (V2R3).

The following messages are new, changed, or no longer issued for z/OS DFSORT Application Programming Guide in z/OS V2R3.

New

The following messages are new.

ICE292A
ICE500A
ICE501A
ICE502A
ICE503A

ICE504A
ICE505A
ICE506I

New Reserved Words for Symbols

The following are new DFSORT/ICETOOL reserved words which are no longer allowed as symbols: UTF8, UTF16, UTF32, and COLLKEY. If you used any of these words as a symbol previously, you must change them. For example, if you used UTF8, you can change it to utf8.

Chapter 1. Introducing DFSORT

DFSORT overview

This chapter introduces IBM z/OS DFSORT Licensed Program 5650-ZOS.

DFSORT is intended to run in problem state and in a user key (that is, key 8 or higher).

DFSORT is a program you use to sort, merge, and copy information.

- When you *sort* records, you arrange them in a particular sequence, choosing an order more useful to you than the original one.
- When you *merge* records, you combine the contents of two or more previously sorted data sets into one.
- When you *copy* records, you make an exact duplicate of each record in your data set.

Merging records first requires that the input data sets are identically sorted for the information you will use to merge them and that they are in the same order required for output. You can merge up to 100 different data sets at a time.

In addition to the three basic functions, you can perform other processing simultaneously:

You can control which records to keep in the final output data set of a DFSORT run by using INCLUDE and OMIT statements in your application. These statements work like filters, testing each record against criteria that you supply and retaining only the ones you want for the output data set. For example, you might choose to work only with records that have a value of “Kuala Lumpur” in the field reserved for office location. Or perhaps you want to leave out any record dated after 2001 if it also contains a value greater than 20 for the number of employees.

You can parse, edit, and reformat your records before or after other processing by using INREC and OUTREC statements. INREC and OUTREC statements support a wide variety of reformatting tasks including:

- The use of fixed position/length fields or variable position/length fields. For fixed fields, you specify the starting position and length of the field directly. For variable fields, such as delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings (and many other types), you define rules that allow DFSORT to extract the relevant data into fixed parsed fields, and then use the parsed fields as you would use fixed fields.
- Insertion of blanks, zeros, strings, current date, future date, past date, current time, sequence numbers, decimal constants, and the results of arithmetic expressions before, between, and after the input fields in the reformatted records.
- Sophisticated conversion capabilities, such as find and replace, hexadecimal display, bit display, translation of EBCDIC letters from lowercase to uppercase or uppercase to lowercase, translation of characters from EBCDIC to ASCII and from ASCII to EBCDIC, translation of characters using the ALTSEQ translation table, conversion of numeric values from one format to another, left-justify or left-squeeze (remove leading blanks or all blanks and shift left), and right-justify or right-squeeze (remove trailing blanks or all blanks and shift right).
- Sophisticated editing capabilities, such as control of the way numeric fields are presented with respect to length, leading or suppressed zeros, thousands separators, decimal points, leading and trailing positive and negative signs, and so on.

  Twenty-seven pre-defined editing masks are available for commonly used numeric editing patterns, encompassing many of the numeric notations used throughout the world. In addition, a virtually unlimited number of numeric editing patterns are available via user-defined editing masks.
- Transformation of SMF, TOD, and ETOD date and time values to more usable forms.
- Conversion of input date fields of one type (CH, ZD, PD, 2-digit year, 4-digit year, Julian, Gregorian) to corresponding output date fields of another type or to a corresponding day of the week.

- Various types of arithmetic operations for input date fields.
- Selection of a character constant, hexadecimal constant, or input field from a lookup table for output, based on a character, hexadecimal, or bit string as input (that is, lookup and change).

You can create the reformatted INREC or OUTREC records in one of the following ways:

- By building the entire record one item at a time.
- By only overlaying specific columns.
- By performing find and replace operations.
- By using sophisticated conditional logic or group operations to choose how different records are reformatted.

You can sum numeric information from many records into one record with the SUM statement. For example, if you want to know the total amount of a yearly payroll, you can add the values for a field containing salaries from the records of all your employees.

You can create one or more output data sets for a sort, copy, or merge application from a single pass over one or more input data sets by using OUTFIL control statements. You can use multiple OUTFIL statements, with each statement specifying the OUTFIL processing to be performed for one or more output data sets. OUTFIL processing begins after all other processing ends (that is, after processing for exits, options, and other control statements). OUTFIL statements support a wide variety of output data set tasks, including:

- Creation of multiple output data sets containing unedited or edited records from a single pass over one or more input data sets.
- Creation of multiple output data sets containing different ranges or subsets of records from a single pass over one or more input data sets. In addition, records that are not selected for any subset can be saved in a separate output data set.
- Conversion of variable-length record data sets to fixed-length record data sets.
- Conversion of fixed-length record data sets to variable-length record data sets.
- A wide variety of parsing, editing and reformatting tasks including:
  - The use of fixed position/length fields or variable position/length fields. For fixed fields, you specify the starting position and length of the field directly. For variable fields, such as delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings (and many other types), you define rules that allow DFSORT to extract the relevant data into fixed parsed fields, and then use the parsed fields as you would use fixed fields.
  - Insertion of blanks, zeros, strings, current date, future date, past date, current time, sequence numbers, decimal constants, and the results of arithmetic expressions before, between, and after the input fields in the reformatted records.
  - Sophisticated conversion capabilities, such as find and replace, hexadecimal display, bit display, translation of EBCDIC letters from lowercase to uppercase or uppercase to lowercase, translation of characters from EBCDIC to ASCII or from ASCII to EBCDIC, translation of characters using the ALTSEQ translation table, conversion of numeric values from one format to another, left-justify or left-squeeze (remove leading blanks or all blanks and shift left), and right-justify or right-squeeze (remove trailing blanks or all blanks and shift right).
  - Sophisticated editing capabilities, such as control of the way numeric fields are presented with respect to length, leading or suppressed zeros, thousands separators, decimal points, leading and trailing positive and negative signs, and so on.

    Twenty-seven pre-defined editing masks are available for commonly used numeric editing patterns, encompassing many of the numeric notations used throughout the world. In addition, a virtually unlimited number of numeric editing patterns are available via user-defined editing masks.
  - Transformation of SMF, TOD, and ETOD date and time values to more usable forms.
  - Conversion of input date fields of one type (CH, ZD, PD, 2-digit year, 4-digit year, Julian, Gregorian) to corresponding output date fields of another type or to a corresponding day of the week.

  - Various types of arithmetic operations for input date fields.
  - Selection of a character constant, hexadecimal constant, or input field from a lookup table for output, based on a character, hexadecimal, or bit string as input (that is, lookup and change).
- Creation of the reformatted records in one of the following ways:
  - By building the entire record one item at a time.
  - By only overlaying specific columns.
  - By performing find and replace operations.
  - By using sophisticated conditional logic or group operations to choose how different records are reformatted.
- Highly detailed three-level (report, page, and section) reports containing a variety of report elements you can specify (for example, current date, current time, edited or converted page numbers, character strings, and blank lines) or derive from the input records (for example, character fields; unedited, edited, or converted numeric input fields; edited or converted record counts; and edited or converted totals, maximums, minimums, and averages for numeric input fields).
- Creation of multiple output records from each input record, with or without intervening blank output records.
- Repetition and sampling of data records.
- Splitting of data records in rotation among a set of output data sets.

You can perform various "join" operations on two files by one or more keys. A JOINKEYS application allows you to create joined records in a variety of ways including inner join, full outer join, left outer join, right outer join and unpaired combinations. The two input files can be of different types (fixed, variable, VSAM, and so on) and have keys in different locations. The records from the two input files can be processed in a variety of ways before and after they are joined.

You can control DFSORT functions with other control statements by specifying alternate collating sequences, invoking user exit routines, overriding installation defaults, and so on.

You can direct DFSORT to pass control during run time to routines you design and write yourself. For example, you can write user exit routines to summarize, insert, delete, shorten, or otherwise alter records during processing. However, keep in mind that the extensive editing capabilities provided by the INCLUDE, OMIT, INREC, OUTREC, SUM, and OUTFIL statements can eliminate the need to write user exit routines. You can write your own routines to correct I/O errors that DFSORT does not handle, or to perform any necessary abnormal end-of-task operation before DFSORT terminates.

You can write an EFS (Extended Function Support) program to intercept DFSORT control statements and PARM options for modification prior to use by DFSORT or to provide alternate sequence support for user-defined data.

You can define and use a symbol for any field, constant, or output column that is recognized in a DFSORT control statement or ICETOOL operator. This makes it easy to create and reuse collections of symbols (that is, mappings) representing information associated with various record layouts. You can use system symbols (for example, &JOBNAME.) in your symbol constants. You can use SET and PROC symbols in your symbol constants. See Chapter 8, “Using symbols for fields and constants,” on page 685.

DFSORT on the Web

For articles, online documents, news, tips, techniques, examples, and more, visit the DFSORT Home Page (www.ibm.com/storage/dfsort).

DFSORT FTP site

You can obtain DFSORT articles and examples via anonymous FTP to:

```
ftp://ftp.software.ibm.com/storage/dfsort/mvs/ (ftp://ftp.software.ibm.com/storage/dfsort/mvs/)
```

Invoking DFSORT

You can invoke DFSORT processing in the following ways:

- With an EXEC job control statement in the input stream using the name of the program (for example, PGM=ICEMAN or PGM=SORT) or the name of a cataloged procedure (for example, SORTD). See Chapter 2, “Invoking DFSORT with Job Control Language,” on page 27.

  TSO users can allocate the needed ddnames (for example, SYSOUT, SORTIN, SORTOUT and SYSIN), and invoke DFSORT using a calling method equivalent to PGM=ICEMAN. For example:

  ```
  call *(iceman)
  ```

  Restriction: TSO users cannot invoke DFSORT using:

  ```
  iceman
  ```

  or any other alias for DFSORT (for example, SORT) in this form.

  See Chapter 11, “Examples of DFSORT job streams,” on page 765 for examples of invoking DFSORT from REXX and CLISTs.
- With a program written in basic assembler language using a system macro instruction. See Chapter 6, “Invoking DFSORT from a program,” on page 517.
- With programs written in either COBOL or PL/I with a special facility of the language. See the programmer's guide describing the compiler version available at your location.
- With the ICETOOL utility. See Chapter 7, “Using ICETOOL,” on page 537.

In this document, the term *directly invoked* means that DFSORT is not initiated from another program. The term *program invoked* means that DFSORT is initiated from another program.

How DFSORT works

This section contains a list of the operating systems supported by DFSORT and an explanation of how DFSORT uses control fields and collating sequences to sort, merge, and copy the records of a data set.

The Blockset technique is DFSORT's most efficient technique for sorting, merging and copying data sets. DFSORT uses the Blockset technique whenever possible to take advantage of its highly optimized internal algorithms and efficient utilization of IBM hardware. If Blockset cannot be used, DFSORT uses another of its techniques — Peerage/Vale or Conventional.

Operating systems

DFSORT runs under control of your z/OS operating system and must be initiated according to the appropriate conventions.

Additionally, DFSORT runs under z/OS when it is running as a guest under z/VM®.

DFSORT is compatible with all of the IBM processors supported by z/OS. In addition to any device supported by z/OS for program residence, DFSORT also operates with any device QSAM or VSAM uses for input or output.

Control fields and collating sequences

You define *control fields* to identify the information you want DFSORT to sort or merge. When thinking of the contents of your data sets, you probably think of names, dates, account numbers, or similar pieces of useful information. For example, when sorting your data sets, you might choose to arrange your records in alphabetical order, by family name. By using the *byte position* and *length* (in bytes) of the portion of each record containing a family name, you can define it as a control field to manipulate with DFSORT.

DFSORT uses the control fields you define as keys in processing. A *key* is a concept, such as family name, that you have in mind when you design a record processing strategy for a particular application. A control field, on the other hand, is a discrete portion of a record that contains the text or symbols corresponding to that information in a form that can be used by DFSORT to identify and sort, or merge the records. For all practical purposes, you can think of keys as equivalent to the control fields DFSORT uses in processing.

To arrange your records in a specific order, identify one or more control fields of your records to use as keys. The sequence in which you list the control fields becomes the order of priority DFSORT uses to arrange your records. The first control field you specify is called the *major control field*. Subsequent control fields are called *minor control fields*, as in first, second, third minor control fields, and so on.

If two or more records have identical values for the first control field, they are arranged according to the values in the second. Records with identical values for the first and second are arranged according to the third, and so on, until a difference is found or no more control fields are available.

Records with identical values for all the control fields specified retain their original input order or are arranged randomly, depending upon which of the two options, EQUALS or NOEQUALS, is in effect. You can direct DFSORT to retain the original input order for records with identical values for all control fields by specifying EQUALS.

Control fields may overlap, or be contained within other control fields (such as a three-digit area code, within a 10-digit telephone number). They do not need to be contiguous but must be located within the first 32752 bytes of the record (see Figure 1 on page 5).

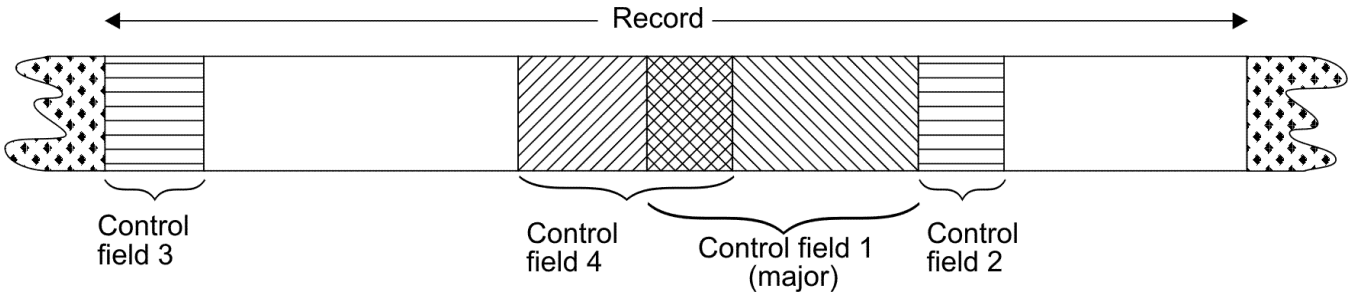


Figure 1. Control Fields

DFSORT offers several standard *collating sequences*. You can choose to arrange your records according to these standard collating sequences or according to a collating sequence defined in the active locale. Conceptually, a collating sequence is a specific arrangement of character priority used to determine which of two values in the same control field of two different records should come first. DFSORT uses EBCDIC, the standard IBM collating sequence, or the ASCII collating sequence when sorting or merging records. If locale processing is in effect, DFSORT will use the collating sequence defined in the active locale.

The collating sequence for character data and binary data is absolute; character and binary fields are not interpreted as having signs. For packed decimal, zoned decimal, fixed-point, normalized floating-point, and the signed numeric data formats, collating is algebraic; each quantity is interpreted as having an algebraic sign.

You can modify the standard EBCDIC sequence to collate differently if, for example, you want to allow alphabetic collating of national characters. An alternate collating sequence can be defined with the ALTSEQ installation option, or you can define it yourself at run-time with the ALTSEQ program control statement. You can also specify a modified collating sequence with an E61 user exit or with an EFS program.

You can specify the LOCALE installation or run-time option to use an active locale's collating rules.

Cultural environment considerations

DFSORT's collating behavior can be modified according to your cultural environment. Your cultural environment is defined to DFSORT using the X/Open** locale model. A locale is a collection of data grouped into categories that describes the information about your cultural environment.

The collate category of a locale is a collection of sequence declarations that defines the relative order between collating elements (single character and multi-character collating elements). The sequence declarations define the collating rules.

The cultural environment is established by selecting the active locale. The active locale affects the behavior of locale-sensitive functions. In particular, the active locale's collating rules affect DFSORT's SORT, MERGE, INCLUDE, and OMIT processing as follows:

- Sort and Merge

  DFSORT produces sorted or merged records for output according to the collating rules defined in the active locale. This provides sorting and merging for single- or multi-byte character data based on defined collating rules that retain the cultural and local characteristics of a language.

- Include and Omit

  DFSORT includes or omits records for output according to the collating rules defined in the active locale. This provides inclusion or omission for single- or multi-byte character data based on defined collating rules that retain the cultural and local characteristics of a language.

  Note: Locale processing is not used for IFTRAIL TRLID or IFTHEN WHEN, BEGIN or END constants or compare fields.

The DFSORT option LOCALE specifies whether locale processing is to be used and, if so, designates the active locale. Only one locale can be active at a time for any DFSORT application.

Unicode

The Unicode Standard provides a single character set that covers the languages of the world, and a small number of machine-friendly encoding forms and schemes to fit the needs of existing applications and protocols. It is designed for best interoperability with both ASCII and ISO-8859-1, the most widely used character sets, to make it easier for Unicode to be used in applications and protocols.

A special number is assigned to every single character. You can refer to The Unicode Consortium (www.unicode.org) to find out more about Unicode. You can find the code charts, which show you which number is assigned to which character within the Unicode standard on the Unicode Character Code Charts (www.unicode.org/charts). If you prefer to have a list of all available characters on your workstation so that you do not have to be online every time you want to find a specific character in the Unicode code page, you can download the Unibook Character Browser (www.unicode.org/unibook).

Currently, the following three forms of Unicode encoding are supported:

UTF8:

Unicode **T**ransformation **F**ormat in 8 bits that uses 1 to 4 bytes, depending on the character.

UTF16:

Unicode **T**ransformation **F**ormat in 16 bits that uses either 2 or 4 bytes to represent a character.

UTF32:

Unicode **T**ransformation **F**ormat in 32 bits.

Unicode Environment Considerations

DFSORT's Unicode Collation behavior can be modified according to your cultural environment. A new COLLKEY parameter allows specifying the Unicode standard character suite for national collation sequences. Your Unicode environment is defined to DFSORT using the COLLKEY=value

Collation supports customization, which means that collation service might behave according to some specific collation rules. Collation rules can be specified using a Locale or User Collation Rules (UCR).

The active collation rules affects the behavior of locale-sensitive functions. In particular, the active locale's collating rules affect DFSORT's SORT and MERGE processing as follows:

Sort and Merge

DFSORT produces sorted or merged records for output according to the collating rules defined in the active locale. This provides sorting and merging for single-or multi-byte character data based on defined collating rules that retain the cultural and local characteristics of a language.

For supported Locales settings, see Appendix E. Locales for collation in *z/OS Unicode Services User's Guide and Reference*

IBM Integrated Accelerator for Z Sort

The IBM z15 provides a new sort accelerator known as the IBM Integrated Accelerator for Z Sort. A new hardware instruction SORTL instruction is introduced to allow use of a hardware accelerated approach to sorting. To take advantage of the IBM Integrated Accelerator for Z Sort, a z15 or later processor is required.

The IBM Integrated Accelerator for Z Sort feature termed as "ZSORT" helps to reduce the CPU costs and improve the elapsed time for eligible workloads. One of the primary requirements for ZSORT is providing enough virtual, real and auxiliary storage. Sort jobs which run in memory constrained environments, where the amount of memory available to be used by DFSORT jobs is restricted, may not achieve optimal performance results or may not be able to exploit ZSORT.

The 64-bit memory objects (above-the-bar-storage) are used to exploit the ZSORT accelerator for sort workloads for optimal results. Since ZSORT is part of the CPU and memory latency is much lower than disk latency, sorting in memory is more efficient than sorting with both memory and disk workspace.

By allowing ZSORT to process the input completely in memory, it can achieve the best results in both elapsed time and CPU time. It may usually require a memory object size to be approximately 2x the input size (actual size may vary depending on record format, record length (LRECL), and key length) to process the sort completely in memory. For simplicity the installation default MOSIZE=MAX is recommended, which allows obtaining the optimal size of memory object. At a minimum, the memory object size must be at least 75% of the input file size to be eligible for ZSORT.

When a sort workload is not able to be processed completely in memory, disk workspace is used in combination with memory object sorting. When using ZSORT enhanced I/O technologies are used and if the sort workload cannot be contained entirely in memory, then it may require work datasets that support zHPF (z High Performance FICON). This is needed to provide the benefit of reducing elapsed and CPU time.

When using ZSORT with disk workspace, make sure enough disk workspace to be allocated and available. The amount of disk workspace required is at least the size of the input file. For jobs that define the sort disk workspace using SORTWKxx DD statements, you must specify enough primary and secondary space in the JCL space parameters.

If enough 64-bit storage for ZSORT cannot be allocated, or enough disk space when using sort works, DFSORT will transition to traditional sort processing. Message ICE267I indicates whether the ZSORT accelerator was used with a reason code to explain the condition the sort job met.

The exploitation of ZSORT is shipped as disabled by default. Enabling the new function via DFSORT installation option allows you to quickly use the new function without having to modify JCL and programs. Consider first enabling ZSORT for individual jobs. This allows you to evaluate ZSORT in your overall batch framework. These options can be activated using one of the following methods in:

- ICEPRMxx members of concatenated PARMLIB
- An ICEMAC macro option
- OPTION control statement

For the first two methods See *z/OS DFSORT Installation and Customization*. For the third method See "OPTION control statement" on page 173.

IBM's zBNA tool (V2.2) provides modeling support for identifying potential ZSORT-eligible candidate jobs and estimates the benefits of ZSORT. The tool uses information in the SMF type 16 records.

Please note that not all sorts would be eligible to use ZSORT. Below is a list of restrictions that will disable ZSORT and will revert to using traditional sorting technique:

- SORTL facility is not enabled/unavailable on the processor
- ZSORT is not enabled
- OPTION COPY or SORT FIELDS=COPY is specified
- Usage of INREC
- Usage of JOINKEYS
- Usage of MERGE FIELDS
- Usage of MODS(EXITS) statements
- Usage of OUTREC
- Usage of OUTFIL
- Usage of SUM FIELDS
- Program invoked sorts
- Memory objects cannot be created
- Insufficient memory object storage available (required more than currently available)
- Unsupported sort fields specified (examples Unicode, Locale, ALTSEQ ...)
- Unknown file size or file size=0.
- SIZE/FILSZ=Uxxxxxx is specified
- SORTIN/SORTOUT is a VSAM Cluster
- Sort control field positions is beyond 4092 and VLSHRT is specified
- Usage of EXCP access method was requested
- Insufficient storage (example above the line or below the line)
- Sorting key greater than 4088 bytes or greater than 4080 bytes if EQUALS is specified
- For variable records, the record length (LRECL) must be greater than 24
- ZHPF is unavailable for a sort that could not be performed entirely in memory
- Insufficient amount of sort workspace.

Note:

1. Since ZSORT's goal is to reduce CPU time and elapsed time, it can require more storage than a DFSORT application that does not use ZSORT. It is recommended to have REGION=0M and MEMLIMIT=NOLIMIT for ZSORT jobs.
2. For below-the-bar memory requirements, DFSORT must be able to obtain enough memory for its various control blocks that reside both below and above the 16MB line. Specifying a SIZE/MAINSIZE=MAX is recommended to achieve the best performance.

   For sorts under 100GB, the shipped default DSA value of 128 is enough. For sorts larger than 100GB, consider increasing DSA to 256. Note that this is only a limit. DFSORT's dynamic storage adjustment calculates the optimal virtual storage allocation for each sort. Increasing the DSA limit impacts only very large sorts that DFSORT determines would benefit from increased virtual storage.

DFSORT processing

You must prepare job control language (JCL) statements and DFSORT program control statements to invoke DFSORT processing. JCL statements (see Chapter 6, "Invoking DFSORT from a program," on page 517) are processed by your operating system. They describe your data sets to the operating system and

initiate DFSORT processing. DFSORT program control statements (see Chapter 3, “Using DFSORT program control statements,” on page 77) are processed by the DFSORT program. They describe the functions you want to perform and invoke the processing you request.

A sort application usually requires intermediate storage as working space during the program run. This storage can be one of the following:

1. Hiperspace, using DFSORT's Hipersorting feature.
2. Work data sets—either allocated dynamically by DFSORT's DYNALLOC facility or specified by the user, using JCL DD statements. If specified by the user, the intermediate storage devices and the amount of work space must be indicated. Methods for determining the amount of work space to allocate are explained in Appendix A, “Using work space,” on page 797.
3. A combination of Hiperspace and work data sets.

Merge and copy applications do not require intermediate storage.

Figure 2 on page 10 illustrates the processing order for record handling, exits, statements, and options. Use this diagram with the text following it to understand the order DFSORT uses to run your job.

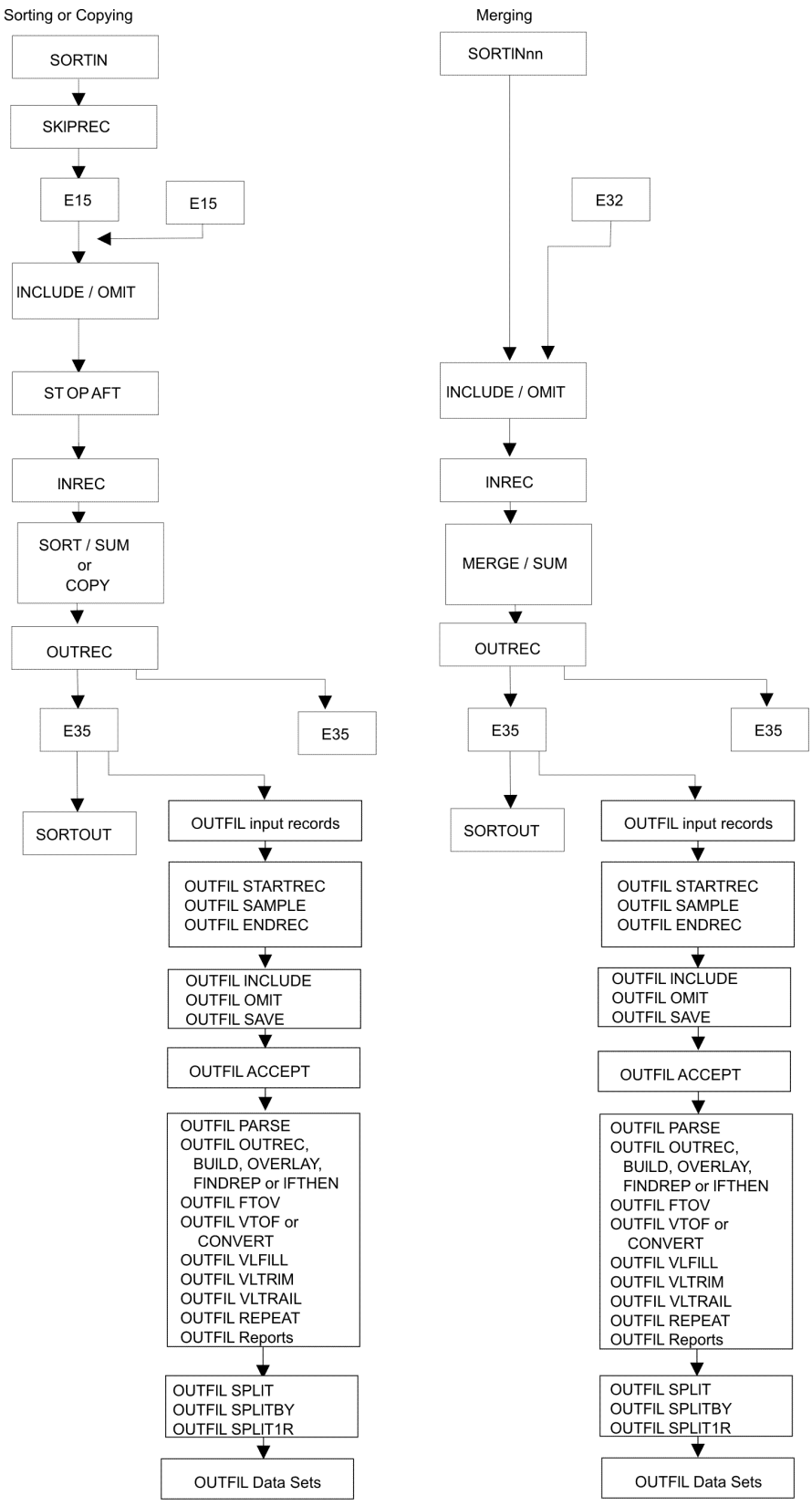


Figure 2. Record Processing Order

As shown in Figure 2 on page 10, DFSORT processing follows this order:

1. DFSORT first checks whether you supplied a SORTIN data set for SORT and COPY jobs or SORTINnn data sets for MERGE jobs. If so, DFSORT reads the input records from them.
   - If no SORTIN data set is present for a SORT or COPY job, you must use an E15 user exit to insert all the records. (This is also true if you invoke DFSORT from a program with the address of an E15 user exit in the parameter list, because SORTIN will be ignored.) DFSORT can use a COBOL E15 routine if you specified the E15 user exit in the MODS statement.
   - If no SORTINnn data sets are present for a MERGE job, you must use an E32 user exit to insert all the records.
2. If input records for SORT or COPY jobs are read from a SORTIN data set, DFSORT performs processing specified with the SKIPREC option. DFSORT deletes records until the SKIPREC count is satisfied. Eliminating records before a SORT or COPY gives better performance.
3. If the input records for a SORT or COPY job are read from a SORTIN data set, DFSORT checks whether you specified an E15 user exit. If so, DFSORT transfers control to the user exit routine. You can use a COBOL E15 routine if the E15 user exit is specified in the MODS statement. The E15 routine can insert, delete, or reformat records.
4. DFSORT performs processing specified on an INCLUDE or OMIT statement. If you used an E15 user exit routine to reformat the records, the INCLUDE/OMIT fields you specify must apply to the current format rather than to the original format. If you use the INCLUDE or OMIT statements to delete unnecessary records before SORT, MERGE, or COPY processing, your jobs run more efficiently.
5. For SORT or COPY jobs, DFSORT performs processing specified with the STOPAFT option. Record input stops after the maximum number of records (n) you specify have been accepted. DFSORT accepts records for processing if they are:
   - Read from SORTIN or inserted by E15
   - Not deleted by SKIPREC
   - Not deleted by E15
   - Not deleted by an INCLUDE or OMIT statement.
6. DFSORT performs processing specified in an INREC statement. Data records are parsed, edited and reformatted according to the options specified. If you reformatted the records before this step, the INREC fields you specify must apply to the current format rather than to the original format.
7. DFSORT performs processing specified in the SORT, MERGE, or OPTION COPY statement.
   - For SORT, all input records are processed before any output record is processed.
   - For COPY or MERGE, an output record is processed after an input record is processed.
   - For SORT or MERGE, if a SUM statement is present, DFSORT processes it during the SORT or MERGE processing. DFSORT summarizes the records and deletes duplicates. If you reformatted the records before this step, the SORT or MERGE and SUM fields you specify must apply to the current format rather than to the original format.
8. DFSORT performs processing specified in an OUTREC statement. Data records are parsed, edited and reformatted according to the options specified. If you reformatted the records before this step, the OUTREC fields you specify must apply to the current format rather than to the original format.
9. If an E35 user exit is present, DFSORT transfers control to your user exit routine after all statement processing (except OUTFIL) is completed. If you reformatted the records, the E35 user exit receives the records in the current format rather than in the original format. You can use a COBOL E35 routine if you specify the E35 user exit in the MODS statement. You can use the E35 exit routine to add, delete, or reformat records.

   If SORTOUT and OUTFIL data sets are not present, the E35 exit must dispose of all the records because DFSORT treats these records as deleted. (This is also true if you do not specify OUTFIL data sets and DFSORT is invoked with the address of an E35 user exit in the parameter list, because SORTOUT will be ignored.)
10. DFSORT writes your records to the SORTOUT data set, if present.
11. DFSORT performs processing specified in one or more OUTFIL statements, if present:

- DFSORT performs processing specified with the STARTREC, SAMPLE, and ENDREC options. Record input for the OUTFIL data sets starts with the record indicated by STARTREC, ends with the record indicated by ENDREC, and is sampled according to the records indicated by SAMPLE.
- DFSORT performs processing specified with the INCLUDE, OMIT, or SAVE option. Records are included or omitted from the OUTFIL data sets according to the criteria specified.
- DFSORT performs processing specified with the ACCEPT option. Record processing ends when the ACCEPT limit is reached.
- DFSORT performs processing specified with the PARSE, OUTREC (or BUILD), OVERLAY, FINDREP, IFTHEN, FTOV, VTOF (or CONVERT), VLFILL, VLTRIM, VLTRAIL and REPEAT options. Data records are parsed, edited, reformatted, converted and repeated according to the options specified.
- DFSORT performs processing specified with the LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS, NODETAIL, BLKCCH1, BLKCCH2, BLKCCT1 and REMOVECC options. Data records are reformatted and report records are generated for the OUTFIL data sets.
- DFSORT performs SPLIT, SPLITBY, or SPLIT1R processing. Records are distributed among the OUTFIL data sets as evenly as possible.
- DFSORT writes your OUTFIL records to the appropriate OUTFIL data sets.

Input data sets—SORTIN and SORTINnn

DFSORT processes two types of input data sets, referred to as the SORTIN data set (or just SORTIN) and the SORTINnn data sets (or just SORTINnn).

The SORTIN DD statement specifies the input data set (or concatenated input data sets) for a sort or copy application. If a SORTIN DD statement is present, it will be used by default for a sort or copy application unless you invoke DFSORT from a program with the address of an E15 user exit in the parameter list.

The SORTINnn DD statements (where nn can be 00 to 99) specify the data sets for a merge application. If a SORTINnn DD statement is present, it will be used by default for a merge application unless you invoke DFSORT from a program with the address of an E32 user exit in the parameter list.

“Data set considerations ” on page 13 contains general information about input data sets. For specific information about the SORTIN data set, see “SORTIN DD statement” on page 66. For specific information about the SORTINnn data sets, see “SORTINnn DD statement” on page 68.

Output data sets—SORTOUT and OUTFIL

DFSORT processes two types of output data sets, referred to as the SORTOUT data set (or just SORTOUT) and the OUTFIL data sets.

The SORTOUT DD statement specifies the single non-OUTFIL output data set for a sort, copy, or merge application. OUTFIL processing does not apply to SORTOUT. If a SORTOUT DD statement is present, it will be used by default for a sort, copy, or merge application unless you invoke DFSORT from a program with the address of an E35 user exit in the parameter list.

The FNAMES and FILES parameters of one or more OUTFIL statements specify the ddnames of the OUTFIL data sets for a sort, copy, or merge application. The parameters specified for each OUTFIL statement define the OUTFIL processing to be performed for the OUTFIL data sets associated with that statement. Each ddname specified must have a corresponding DD statement.

Although the ddname SORTOUT can actually be used for an OUTFIL data set, the term "SORTOUT" will be used to denote the single non-OUTFIL output data set.

“Data set considerations ” on page 13 contains general information about output data sets. For specific information about the SORTOUT data set, see “SORTOUT and OUTFIL DD statements” on page 71. For specific information about the OUTFIL data sets, see “SORTOUT and OUTFIL DD statements” on page 71 and “OUTFIL control statements” on page 221.

Data set considerations

You must define any data sets you provide for DFSORT according to the conventions your operating system requires. You can use the label checking facilities of the operating system during DFSORT processing. See "General considerations" on page 13 for general considerations.

You must describe all data sets (except those allocated with the DYNALLOC parameter) in DD statements. You must place the DD statements in the operating system input stream with the job step that allocates data sets for DFSORT processing.

Sorting or copying records

Input to a sort or copy application can be a blocked or unblocked QSAM or VSAM data set containing fixed- or variable-length records. QSAM input data sets can be concatenated even if they are on dissimilar devices. See "SORTIN DD statement" on page 66 for the restrictions that apply.

Output from a sort or copy application can be blocked or unblocked QSAM or VSAM data sets, regardless of whether the input is QSAM or VSAM. Unless OUTFIL is used to convert variable input to fixed output, or fixed input to variable output, an output data set must be the same type (fixed or variable) as the input data set.

Files in a z/OS file system are supported as input and output for sort and copy applications.

Merging records

Input to a merge application can be up to 100 blocked or unblocked QSAM or VSAM data sets containing fixed- or variable-length records. The input data sets can be either QSAM or VSAM, but not both. The records in all input data sets must already be sorted in the same order as that required for output.

Output from a merge application can be blocked or unblocked QSAM or VSAM data sets, regardless of whether the input is QSAM or VSAM. Unless OUTFIL is used to convert variable input to fixed output, or fixed output to variable output, an output data set must be the same type (fixed or variable) as the input data set.

Files in a z/OS file system are supported as input and output for merge applications.

Data set notes and limitations

There are some considerations and limitations that you need to be aware of. These are described in the following sections.

For more information about specific DFSORT data sets, see "Using DD statements" on page 60.

General considerations

Variable-length records are processed with a record descriptor word (RDW) in positions 1-4, so the data starts in position 5. Fixed-length records are processed without an RDW, so the data starts in position 1. Control statement positions should be specified accordingly.

Your records can be EBCDIC, ASCII, Japanese, and data types you define yourself. To process Japanese data types with DFSORT, you can use the IBM Double Byte Character Set Ordering Support Program (DBCS Ordering), Licensed Program 5665-360, Release 2.0, or you can use locale processing with the appropriate locale.

Input and output data sets must be on devices that can be used with QSAM or VSAM.

Standard system data management rules apply to all data set processing. In particular:

- Be aware that when using fixed standard record format for input data sets, the first short block is treated like an End of Volume. See *z/OS DFSMS Using Data Sets* for more details.

- Be aware that, in some cases, if a DD statement specifies a data set for output that is extended to a second or subsequent volume, and another DD statement within the same step requests the same data set, only the records on the first volume will be read, and incorrect output will result.

  Specifically, when a new output data set is allocated with a unit count and volume count greater than one, but specifies no volume serial numbers, one volume is allocated. If a second or succeeding DD statement within the same step requests the same data set, the same volume is allocated to it. If this job step extends the output data set to more volumes, this new volume information is not available to the second or succeeding DD statement.

  Thus, you should not use different DDs for a data set to be used for output and then input in the same step, unless the data set cannot be extended to a second or subsequent volume, or is allocated with the guaranteed space attribute in the storage class. See *z/OS MVS JCL Reference* and *z/OS DFSMS Using Data Sets* for more details.

The maximum record length DFSORT can handle is subject to the following limitations:

- Record length can never exceed the maximum record length you specify.
- Variable-length records are limited to 32756 bytes.
- VSAM variable-length records are limited to 32752 bytes.
- Fixed-length records are limited to 32760 bytes.
- Variable block-spanned records are limited to 32767 bytes.
- For a tape work data set sort, the maximum record length is limited to 32752 bytes with NOEQUALS in effect and to 32748 bytes with EQUALS in effect.

Note: If AQ format is specified, or CH format is specified and the CHALT option is in effect, the maximum record length for variable-length records is 32767 bytes, less the length of the control fields.

The number of records that can be sorted using a given amount of storage is reduced by:

- Processing control fields of different formats
- Large numbers of control fields
- Large numbers of intermediate data sets.

Providing an Extended Function Support program with an EFS01 routine can limit the record length that can be used when processing variable-length records.

The minimum block length for tape work data sets is 18 bytes; the minimum record length is 14 bytes.

Padding and truncation

You can control the action that DFSORT takes when the SORTOUT LRECL is smaller than the SORTIN/SORTINnn LRECL with the TRUNC option as described in “OPTION control statement” on page 173.

DFSORT truncates fixed-length records on the right when the SORTOUT LRECL is smaller than the SORTIN/SORTINnn LRECL provided that:

- The application is not a conventional merge or tape work data set sort.
- TRUNC=RC16 is not in effect.

You can control the action that DFSORT takes when a variable-length output record is longer than the LRECL of the SORTOUT or OUTFIL data set to which it is to be written by using the VLLONG or NOVLLONG option as described in “OPTION control statement” on page 173.

You can control the action that DFSORT takes when the SORTOUT LRECL is larger than the SORTIN/SORTINnn LRECL with the PAD option as described in “OPTION control statement” on page 173.

DFSORT pads fixed-length records with binary zeros on the right when the SORTOUT LRECL is larger than the SORTIN LRECL provided that:

- The Blockset technique is selected.

- The application is a sort or copy.
- PAD=RC16 is not in effect.

DFSORT does not pad or truncate records returned from an E15 or E35 user exit since it expects the exit to pad or truncate each record appropriately.

You can use INREC, OUTREC, and OUTFIL to pad, truncate, and reformat records. See "INREC control statement" on page 127 and "OUTREC control statement" on page 385 for details.

See "Use ICEGENER instead of IEBGENER" on page 760 for information about padding and truncating with ICEGENER.

For more information about Blockset and other DFSORT techniques, see "Specify efficient sort/merge techniques" on page 748.

QSAM considerations

- If you use DSN=NULLFILE on your DD statement for an input data set, a system restriction prevents DFSORT from using the EXCP access method.
- Empty input data sets can be used.
- If any of the input data sets are on tape without standard labels, DCB parameters must be specified on their DD statements.
- ISO/ANSI Version 1 tape files can only be used as input—never as output.
- DFSORT sets appropriate BUFNO values for the input and output data sets; specifying BUFNO in the DD statements for these data sets has no effect.

See "SORTIN DD statement" on page 66 for additional considerations.

VSAM considerations

- You can have DFSORT process VSAM records as fixed-length (F) or variable-length (V). When you use VSAM input, DFSORT selects fixed-length processing if you specify RECORD TYPE=F or variable-length processing if you specify RECORD TYPE=V. If you do not specify RECORD TYPE=x, DFSORT selects the record type to use according to the "rules" described in the discussion of the TYPE operand in "RECORD control statement" on page 418. The record type selected affects how the records are treated, and how control statement positions should be specified, as follows:
  - **Variable-length processing:** An RRDS, KSDS, ESDS or VRRDS can always be processed as variable-length. For VSAM input, DFSORT reads each record and prepends a record descriptor word (RDW) to it. For VSAM output, DFSORT removes the RDW before writing each record. Since DFSORT uses an RDW in positions 1-4 to process variable-length records, the data starts in position 5. Control statement positions should be specified accordingly.
  - **Fixed-length processing:** An RRDS can always be processed as fixed-length. A KSDS, ESDS or VRRDS used for input should only be processed as fixed-length if all of its records have a length equal to the maximum record size defined for the cluster. Otherwise, input records which are shorter than the maximum record size are padded with bytes that may or may not be zeros (that is, "garbage" bytes). DFSORT does not use an RDW to process fixed-length records, so the data starts in position 1. Control statement positions should be specified accordingly.
- If a data set is password protected, passwords can be entered at the console or (with some restrictions) through routines at user exits E18, E38, and E39.

  Note: Passwords cannot be handled in this way for OUTFIL data sets.

- If VSAMIO and RESET are in effect, a data set defined with REUSE can be used for both input and output for a sort; that is, the data set can be sorted in-place.
- A data set used for input or output must have been previously defined.
- If VSAMEMT is in effect, an empty input data set is processed as having zero records.

- VSAM data sets must not be concatenated (system restriction).
- VSAM and non-VSAM input data sets must not be specified together for a sort, merge or copy application.
- If output is a VSAM key-sequenced data set (KSDS), the key must be the first control field (or the key fields must be in the same order as the first control field). VSAM does not allow you to store records with duplicate primary keys.
- Any VSAM exit function available for input data sets can be used except EODAD. See the description of E18 use with VSAM in Chapter 5, “Using your own user exit routines,” on page 467.
- You must build the VSAM exit list with the VSAM EXLST macro instruction giving the addresses of your routines that handle VSAM exit functions.
-
- When processing variable-length records with VSAM input and non-VSAM output, the output LRECL must be at least 4 bytes greater than the maximum record size defined for the cluster. Non-VSAM variable-length records have a record descriptor word (RDW) field 4 bytes long at the beginning of each record, but VSAM records do not. The record size defined for the VSAM cluster is therefore 4 bytes less than the non-VSAM LRECL.
- An output data set defined without REUSE is processed as MOD.
- If RESET is in effect, an output data set defined with REUSE is processed as NEW. If NORESET is in effect, an output data set defined with REUSE is processed as MOD.
- DFSORT cannot access VSAM data sets in RLS mode, that is, RLS=CR and RLS=NRI are not supported for VSAM input and output data sets.

Data set encryption

With z/OS data set encryption, you can encrypt data without requiring application changes. z/OS data set encryption, through SAF controls and RACF® or equivalent function along with SMS policies, allows you to identify new data sets or groups of data sets to be encrypted. You can specify encryption key labels to identify encryption keys to be used to encrypt selected data sets. The specified key label and encryption key must exist in the ICSF key repository (CKDS). With data set encryption, you are able to protect data residing on disk from being viewed by unauthorized users in the clear. Authorization is based on access to the key label that is associated with the data set and used by the access methods to encrypt and decrypt the data.

DFSORT supports reading and writing to the following types of data sets:

- Sequential extended format data sets accessed through BSAM
- VSAM extended format data sets (KSDS, ESDS, RRDS, VRRDS), accessed through base VSAM
- PDSE data sets accessed through BSAM

Encrypted data sets must be SMS-managed extended format. They also can be compressed format.

For complete details on data set encryption See Chapter 5. Protecting Data sets in z/OS DFSMS Using Data Sets

XTIOT, uncaptured UCBs and DSAB above 16 megabytes

A program that invokes DFSORT, ICETOOL or ICEGENER can dynamically allocate input, output and work data sets using the options for XTIOT, uncaptured UCBs, and DSAB above 16 megabyte virtual. These data sets are supported to the extent that z/OS supports them.

Note: A program that invokes DFSORT, ICETOOL or ICEGENER should not allocate other data sets such as message, control, list, count, or symbol data sets using the options for XTIOT, uncaptured UCBs, and DSAB above 16 megabyte virtual. These data set options are not supported and the data set will not be recognized.

z/OS file system considerations

Files in a z/OS file system can be used for input and output, but are only supported by the Blockset technique. If Blockset is not selected for a DFSORT application that uses z/OS UNIX files, DFSORT will issue an error message and terminate.

You should be familiar with the information found in *z/OS UNIX System Services User's Guide* regarding z/OS UNIX files if you use them. DFSORT uses BSAM to access z/OS UNIX files and is thus subject to all of the capabilities and restrictions that entails, as described in *z/OS DFSMS Using Data Sets*.

Installation defaults

When your system programmers installed DFSORT, they selected separate sets of installation defaults for the following eight installation environments:

ICEAM1 (JCL)
is the batch direct invocation environment. This set of installation defaults is used at run time when DFSORT is invoked directly (that is, not through programs) by batch jobs, provided that an enabled time-of-day installation environment (ICETDx) is not activated.

ICEAM2 (INV)
is the batch program invocation environment. This set of installation defaults is used at run time when DFSORT is invoked through batch programs, provided that an enabled time-of-day installation environment (ICETDx) is not activated.

ICEAM3 (TSO)
is the TSO direct invocation environment. This set of installation defaults is used at run time when DFSORT is invoked directly (that is, not through programs) by foreground TSO users, provided that an enabled time-of-day installation environment (ICETDx) is not activated.

ICEAM4 (TSOINV)
is the TSO program invocation environment. This set of installation defaults is used at run time when DFSORT is invoked through programs by foreground TSO users, provided that an enabled time-of-day installation environment (ICETDx) is not activated.

ICETD1 (TD1)
is the first time-of-day installation environment. This set of installation defaults is used at run time when it is activated for the time-of-day of the run, provided it is enabled by the installation environment (ICEAMx) in effect.

ICETD2 (TD2)
is the second time-of-day installation environment. This set of installation defaults is used at run time when it is activated for the time-of-day of the run, provided it is enabled by the installation environment (ICEAMx) in effect.

ICETD3 (TD3)
is the third time-of-day installation environment. This set of installation defaults is used at run time when it is activated for the time-of-day of the run, provided it is enabled by the installation environment (ICEAMx) in effect.

ICETD4 (TD4)
is the fourth time-of-day installation environment. This set of installation defaults is used at run time when it is activated for the time-of-day of the run, provided it is enabled by the installation environment (ICEAMx) in effect.

The selected installation defaults can affect the way your applications run, and in many cases can be overridden by specifying the appropriate run-time parameters (see Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details). This document assumes that DFSORT is running with the installation defaults it was delivered with (that is, with the IBM-supplied installation defaults).

You can use an ICETOOL job similar to the following one to display a report of the installation defaults to be used at run-time.

```
//DFRUN JOB A402,PROGRAMMER
//LISTDEF EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=A
//DFSMSG DD SYSOUT=A
//SHOWDEF DD SYSOUT=A
//TOOLIN DD *
DEFAULTS LIST(SHOWDEF)
/*
```

Figure 3. Using ICETOOL to List Installation Defaults

See Chapter 7, “Using ICETOOL,” on page 537 and “DEFAULTS operator” on page 561 for more information on using ICETOOL and the DEFAULTS operator.

The functions of the available installation options are summarized later in this section. *z/OS DFSORT Installation and Customization* contains complete descriptions of the available installation options, as well as planning considerations and general information about installing DFSORT. Step-by-step installation procedures are listed in the z/OS Install/Migration page (www.ibm.com/systems/z/os/zos/installation).

Parameter
Function

INV|JCL|TSO|TSOINV|TD1|TD2|TD3|TD4
Specifies the invocation installation environment (ICEAMx) or time-of-day installation environment (ICETDx) for which this set of installation defaults is to be used.

ENABLE
Specifies whether ICETDx installation defaults are to be used if activated for this ICEAMx installation environment.

day
Specifies the time ranges for each day of the week when this ICETDx installation environment is to be activated.

ABCODE

Specifies the ABEND code used when DFSORT abends for a critical error.

ALTSEQ

Specifies changes to the ALTSEQ translation table.

ARESALL

Specifies the number of bytes reserved above 16MB virtual for system use.

ARESINV

Specifies the number of bytes reserved above 16MB virtual for the invoking program when DFSORT is program invoked.

CFW

Specifies whether DFSORT can use cache fast write when processing work data sets.

CHALT

Translates format CH as well as format AQ, or translates format AQ only.

CHECK

Specifies whether record count checking is suppressed for applications that use an E35 user exit routine without an output data set.

CINV

Specifies whether DFSORT can use control interval access for VSAM data sets.

COBEXIT

Specifies the library for COBOL E15 and E35 routines.

COLLKEY

Specifies a locale, where specific Collation Rules will modify any of the default Unicode Collation tables specified.

DIAGSIM

Specifies whether a SORTDIAG DD statement is to be simulated for DFSORT applications.

DSA

Specifies the maximum amount of storage available to DFSORT for dynamic storage adjustment of Blockset sort applications.

DSPSIZE

Specifies the maximum amount of data space to use for dataspace sorting.

DYNALOC

Specifies the default values for device name and number of work data sets to be dynamically allocated. These default values are used in conjunction with the DYNAUTO installation option and the DYNALLOC run-time option.

DYNAPCT

Specifies additional work data sets to be dynamically allocated and only used when needed.

DYNAUTO

Specifies whether work data sets are dynamically allocated automatically.

DYNSPC

Specifies the total default primary space allocation for all of the dynamically allocated work data sets when the file size is unknown.

EFS

Specifies the name of a user-written Extended Function Support program to be called by DFSORT.

EQUALS

Specifies whether the order of records that collate identically is preserved from input to output.

ERET

Specifies the action taken if DFSORT encounters a critical error.

ESTAE

Specifies whether DFSORT deletes its ESTAE recovery routine early or uses it for the entire run.

EXITCK

Specifies whether DFSORT terminates or continues when it receives certain invalid return codes from E15 or E35 user exit routines.

EXPMAX

Specifies the maximum total amount of available storage to be used at any one time by all Hipersorting, memory object sorting and dataspace sorting applications.

EXPOLD

Specifies the maximum total amount of old storage to be used at any one time by all Hipersorting, memory object sorting and dataspace sorting applications.

EXPRES

Specifies the minimum amount of available storage to be reserved for use by non-Hipersorting, non-memory object sorting and non-dataspace sorting applications.

FSZEST

Specifies whether DFSORT treats run-time options FILSZ=n and SIZE=n as exact or estimated file sizes.

GENER

Specifies the name that ICEGENER is to use to transfer control to the IEBGENER system utility. (ICEGENER is DFSORT's facility for IEBGENER jobs.)

GNPAD

Specifies the action to be taken by ICEGENER for LRECL padding.

GNTRUNC

Specifies the action to be taken by ICEGENER for LRECL truncation.

HIPRMAX

Specifies the maximum amount of Hiperspace to use for Hipersorting.

IDRCPCT

Specifies a percentage which represents the approximate amount of data compaction achieved by using the Improved Data Recording Capability feature of IBM tape devices that support compaction.

IEXIT

Specifies whether DFSORT passes control to your site's ICEIEXIT routine.

IGNCKPT

Specifies whether the checkpoint/restart facility is ignored if it is requested at run-time and the Blockset technique (which does not support the checkpoint/restart facility) can be used.

IOMAXBF

Specifies an upper limit to the amount of buffer space to be used for SORTIN, SORTINnn and SORTOUT data sets.

LIST

Specifies whether DFSORT prints control statements.

LISTX

Specifies whether DFSORT prints control statements returned by an Extended Function Support program.

LOCALE

Specifies whether locale processing is to be used and, if so, designates the active locale.

MAXLIM

Specifies an upper limit to the amount of main storage available to DFSORT below 16MB virtual.

MINLIM

Specifies a lower limit to the amount of main storage available to DFSORT.

MOSIZE

Specifies the maximum amount of memory object storage to use for memory object sorting.

MOWRK

Specifies whether DFSORT can use memory object storage as intermediate work space.

MSGCON

Specifies the class of program messages DFSORT writes to the master console.

MSGDDN

Specifies an alternate name for the message data set.

MSGPRT

Specifies the class of program messages DFSORT writes to the message data set.

NOMSGDD

Specifies whether DFSORT terminates or continues when the message data set is required but is not available.

NULLOFL

Specifies the action to be taken by DFSORT when there are no data records for an OUTFIL data set.

NULLOUT

Specifies the action to be taken by DFSORT when there are no records for the SORTOUT data set.

ODMAXBF

Specifies an upper limit to the amount of buffer space to be used for each OUTFIL data set.

OUTREL

Specifies whether unused temporary output data set space is released.

OUTSEC

Specifies whether DFSORT uses automatic secondary allocation for output data sets that are temporary or new.

OVERRGN

Specifies the amount of main storage above the REGION value available to Blockset.

OVFLO

Specifies the action to be taken by DFSORT when BI, FI, PD or ZD summary fields overflow.

PAD

Specifies the action to be taken by DFSORT for LRECL padding.

PARMDDN

Specifies an alternate ddname for the DFSORT DFSPARM data set.

RESALL

Reserves storage for system and application use when SIZE/MAINSIZE=MAX is in effect.

RESET

Specifies whether DFSORT processes a VSAM output data set defined with REUSE as a NEW or MOD data set.

RESINV

Reserves storage for programs invoking DFSORT when SIZE/MAINSIZE=MAX is in effect.

SDB

Specifies whether DFSORT should use the system-determined optimum block size for output data sets when the block size is zero.

SDBMSG

Specifies whether DFSORT and ICETOOL should use the system-determined optimum block size for message and list data sets when the block size is zero.

SIZE

Specifies the maximum amount of main storage available to DFSORT.

SMF

Specifies whether DFSORT produces SMF type-16 records.

SOLRF

Specifies whether DFSORT uses the reformatted record length for the SORTOUT LRECL.

SORTLIB

Specifies whether DFSORT searches a system or private library for the modules used with a tape work data set sort or Conventional merge.

SPANINC

Specifies the action to be taken by DFSORT when incomplete spanned records are detected.

SVC

Specifies a user SVC number for DFSORT.

SZERO

Specifies whether DFSORT treats numeric -0 and +0 values as signed (that is, different) or unsigned (that is, the same).

TEXIT

Specifies whether DFSORT passes control to your site's ICETEXIT routine.

TMAXLIM

Specifies an upper limit to the total amount of main storage above and below 16MB virtual available to DFSORT when SIZE/MAINSIZE=MAX is in effect.

TUNE

Specifies whether DFSORT should favor optimization of central storage or disk work space for sort applications.

TRUNC

Specifies the action to be taken by DFSORT for LRECL truncation.

VERIFY

Specifies whether the sequence of output records is verified.

VIO

Specifies whether virtual allocation of work data sets is accepted.

VLLONG

Specifies whether DFSORT truncates long variable-length output records.

VLSCMP

Specifies whether DFSORT pads short variable-length compare fields.

VLSHRT

Specifies whether DFSORT continues processing if a short variable-length control field, compare field or summary field is found.

VSAMBSP

Specifies the number of VSAM buffers DFSORT can use.

VSAMEMT

Specifies whether DFSORT accepts an empty VSAM input data set.

VSAMIO

Specifies whether DFSORT allows a VSAM data set defined with REUSE to be sorted in-place.

WRKREL

Specifies whether unused temporary work data set space is released.

WRKSEC

Specifies whether DFSORT uses automatic secondary allocation for temporary work data sets.

Y2PAST

Specifies the sliding or fixed century window.

ZDPRINT

Specifies whether DFSORT produces printable numbers from positive ZD fields that result from summarization.

Tables showing all the possible sources of specification and order of override for each option are shown in Appendix B, "Specification/override of DFSORT options," on page 805.

Migrating to DFSORT from other sort products

If you are migrating to DFSORT, you should run an ICETOOL DEFAULTS report to review the installation defaults set at your site. In particular, the options shown in the table that follows can make DFSORT operate more like other sort products, thus making migration easier. The installation options, described in *z/OS DFSORT Installation and Customization*, change the way DFSORT works globally by default. The run-time options, described in Chapter 2, "Invoking DFSORT with Job Control Language," on page 27 and Chapter 3, "Using DFSORT program control statements," on page 77, can be used to override the installation defaults for specific jobs.

Table 4. Options That Can Ease Migration

Options That Can Ease MigrationInstallation	**Run-Time**
ABCODE=MSG/n	
DYNALOC=(d,n)	DYNALLOC=(d,n)
DYNAPCT=x/OLD	DYNAPCT=x/OLD
DYNAUTO=YES/IGNWKDD/NO	DYNALLOC=(d,n)
DYNSPC=n	DYNSPC=n
EQUALS=YES/NO/VBLKSET	EQUALS/NOEQUALS
EXITCK=STRONG/WEAK	EXITCK=STRONG/WEAK
FSZEST=YES/NO	FILSZ=n/En/Un
NOMSGDD=QUIT/ALL/CRITICAL/NONE	
PARMDDN=ddname	
RESET=YES/NO	RESET/NORESET
SORTLIB=SYSTEM/PRIVATE	
SZERO=YES/NO	SZERO/NOSZERO
VLLONG=YES/NO	VLLONG/NOVLLONG
VLSCMP=YES/NO	VLSCMP/NOVLSCMP
VSAMEMT=YES/NO	VSAMEMT/NVSAMEMT
VSAMIO=YES/NO	VSAMIO/NOVSAMIO
ZDPRINT=YES/NO	ZDPRINT/NZDPRINT

DFSORT messages and return codes

You can determine, during installation or run-time, whether DFSORT writes messages to the message data set, to the master console, or to both. You can also direct an Extended Function Support program to write messages to the message data set.

Messages written to the message data set can be either critical error messages, informational error messages, or diagnostic messages, as determined during installation or run-time.

Messages written to the master console can be either critical error messages or informational error messages, as determined during installation.

See *z/OS DFSORT Messages, Codes and Diagnosis Guide* for complete information about DFSORT messages.

For successful completion, DFSORT passes back a return code of 0 or 4 to the operating system or the invoking program.

For unsuccessful completion due to an unsupported operating system, DFSORT passes back a return code of 24 to the operating system or the invoking program.

For unsuccessful completion with NOABEND in effect, DFSORT passes back a return code of 16, 20 or 28 to the operating system or the invoking program.

For unsuccessful completion with ABEND in effect, DFSORT issues a user abend with the appropriate code as specified by the ABCODE installation option (either the error message number or a number between 1 and 99).

The meanings of the return codes that DFSORT passes back (in register 15) are:

0

Successful completion. DFSORT completed successfully.

4

Successful completion. DFSORT completed successfully, and:

- OVFLO=RC4 was in effect and summary fields overflowed, or
- PAD=RC4 was in effect and the SORTOUT LRECL was larger than the SORTIN/SORTINnn LRECL (LRECL padding), or
- TRUNC=RC4 was in effect and the SORTOUT LRECL was smaller than the SORTIN/SORTINnn LRECL (LRECL truncation), or
- SPANINC=RC4 was in effect and one or more incomplete spanned records was detected, or
- NULLOUT=RC4 was in effect and there were no records for the SORTOUT data set, or
- NULLOFL=RC4 was in effect and there were no data records for an OUTFIL data set.

16

Unsuccessful completion. DFSORT detected an error that prevented it from completing successfully.

20

Message data set missing. Installation option NOMSGDD=QUIT was in effect and neither a message data set DD statement nor a SYSOUT DD statement was provided.

24

Unsupported operating system. This release of DFSORT does not support this operating system.

28

Wrong entry name. DFSORT detected one of the following errors related to the program entry name that prevented it from completing successfully:

- DFSORT was invoked directly with PGM=ICEMAN64 or PGM=SORT64 (instead of with PGM=ICEMAN or PGM=SORT).
- DFSORT was invoked from a program using entry name ICEMAN64 or SORT64, and an extended invocation parameter list, or a 24-bit invocation parameter list (instead of a 64-bit invocation parameter list).

- DFSORT was invoked from a program using an entry name other than ICEMAN64 or SORT64 (for example, ICEMAN or SORT) and a 64-bit invocation parameter list.

Use Blockset whenever possible

Blockset is DFSORT's most efficient technique. It supports many features not supported by DFSORT's less efficient Peerage/Vale and Conventional techniques (see ICE189A in *z/OS DFSORT Messages, Codes and Diagnosis Guide* for a list of these features).

DFSORT always selects Blockset for a copy application. DFSORT selects Blockset for a sort or merge application unless something prevents it from doing so (see ICE800I in *z/OS DFSORT Messages, Codes and Diagnosis Guide* for a list of reasons Blockset cannot be used).

Note: Blockset cannot be used to process BDAM data sets.

Message ICE143I indicates whether Blockset or a less efficient technique was selected for a particular run. If Blockset was not selected for a sort or merge application, check the reason code in message ICE800I, which indicates the reason Blockset could not be used. If you did not get message ICE800I, add the following DD statements to your application and rerun it:

```
//SORTDIAG DD DUMMY
//SYSOUT DD SYSOUT=*
```

If possible and appropriate, remove the obstacle that is causing Blockset not to be selected.

Chapter 2. Invoking DFSORT with Job Control Language

Using the JCL

Your operating system uses the job control language (JCL) you supply with your DFSORT program control statements to:

- Identify you as an authorized user
- Allocate the necessary resources to run your job
- Run your job
- Return information to you about the results
- Terminate your job.

You must supply JCL statements with every DFSORT job you submit.

Required JCL includes a JOB statement, an EXEC statement, and several DD statements. The statements you need and their exact form depend upon whether you:

- Use an EXEC statement in the input job stream or a system macro instruction within another program to invoke DFSORT
- Use EXEC statement cataloged procedures to invoke DFSORT
- Specify various DFSORT control statements or PARM options
- Want to use program exits to activate routines of your own
- Use dynamic binding or link-editing
- Want to see diagnostic messages.

The JCL statements and their functions are listed in the following. Details on coding the individual statements are presented in subsequent sections.

JCL Statement
Description

//JOBLIB DD

Defines your program link library if it is not already known to the system

//STEPLIB DD

Same as //JOBLIB DD

//SORTLIB DD

Defines the data set that contains special load modules if it is not already known to the system

//SYSOUT DD[1]

Defines the message data set

//SYMNAMES DD

Defines the SYMNAMES data set containing statements to be used for symbol processing

//SYMNOUT DD

Defines the data set in which SYMNAMES statements and the symbol table are to be listed

//SORTIN DD[1]

Defines the input data set for a sort or copy

//SORTINnn DD[1]

Defines the input data sets for a merge

//SORTOUT DD[1]

Defines the SORTOUT output data set for a sort, merge, or copy

//outfil DD

Defines an OUTFIL output data set for a sort, merge, or copy

//SORTWKdd DD[1]

Defines intermediate storage data sets for a sort

//DFSPARM DD[1]

Contains DFSORT PARM options and program control statements

//SYSIN DD

Contains DFSORT program control statements

//SORTCNTL DD[1]

Same as //SYSIN DD

//SORTDIAG DD

Specifies that all messages and program control statements be printed

//SORTCKPT DD

Defines the data set for checkpoint records

//SYSUDUMP DD

Defines the data set for output from a system ABEND dump routine

//SYSMDUMP DD

Same as //SYSUDUMP DD

//SYSABEND DD

Same as //SYSUDUMP DD

//SORTSNAP DD

Defines the snap dump data set dynamically allocated by DFSORT

//ddname

Defines the data set containing exit routines (as specified in the MODS program control statement).

The following DD statements are necessary only for dynamic binding or link-editing of exit routines

//SYSPRINT DD

Defines the message data set for the linkage editor

//SYSUT1 DD

Defines the intermediate storage data set for the linkage editor

//SYSLIN DD

Defines the data set for control information for the linkage editor

//SYSLMOD DD

Defines the data set for output from the linkage editor

//SORTMODS DD

Defines the temporary partitioned data set for user exit routines from SYSIN.

1

These are the default ddnames with which DFSORT was delivered. SYSOUT and DFSPARM may have been changed during DFSORT installation. You can change all of the indicated ddnames at run time. For override information, see Appendix B, “Specification/override of DFSORT options,” on page 805.

Using the JOB statement

The JOB statement is the first JCL statement of your job. It must contain a valid job name in the name field and the word JOB in the operation field. All parameters in the operand field are optional, although your site may have made information such as account number and the name of the programmer mandatory:

```
//jobname JOB accounting information, programmer's name, etc.
```

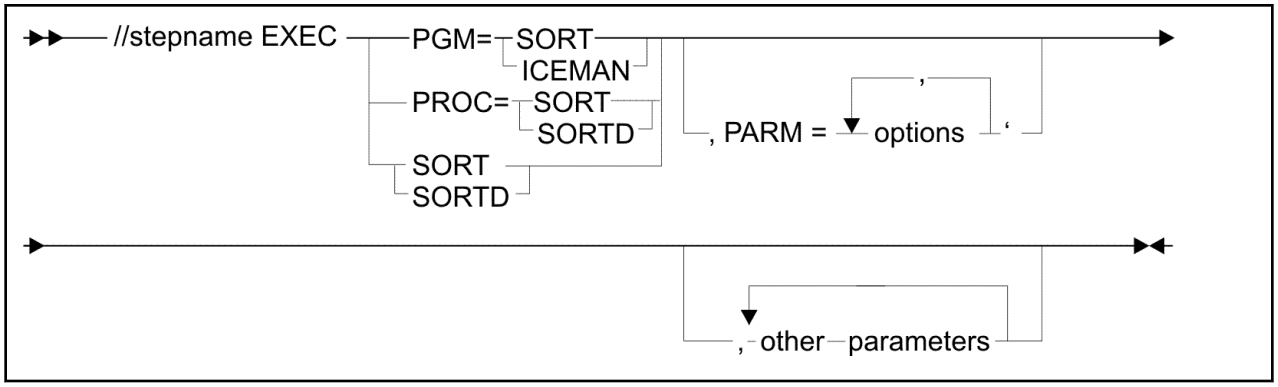
Using SET and PROC symbols in DFSORT control statements

For jobs that directly invoke DFSORT (PGM=SORT or PGM=ICEMAN), you can construct DFSORT symbols that incorporate JCL PROC or SET symbols as well as text and system symbols. You can then use those constructed DFSORT symbols in DFSORT control statements in the same way you use regular DFSORT symbols. For complete information, see Chapter 8, “Using symbols for fields and constants,” on page 685.

Using the EXEC statement

The EXEC statement is the first JCL statement of each job step or of each procedure step in a cataloged procedure. It identifies DFSORT to the operating system. You can also specify DFSORT options on the EXEC statement.

The format of the EXEC statement is:



If you use a cataloged procedure (discussed in detail later in this section), specify PROC=SORT or PROC=SORTD. You can omit PROC= and simply specify SORT or SORTD. However, PROC= can remind you that a cataloged procedure is being used.

If you do not use a cataloged procedure, use PGM= either with the actual name of the sort module (ICEMAN) or with one of its aliases: SORT, IERRCO00, or IGHRCO00. Be sure that the alias has not been changed at your site.

Specifying EXEC statement cataloged procedures

A cataloged procedure is a set of JCL statements, including DD statements, that has been assigned a name and placed in a partitioned data set called the procedure library. Two cataloged procedures are supplied with the program: SORT and SORTD. Specify them in the first parameter of the EXEC statement by PROC=SORT, PROC=SORTD, or simply SORT or SORTD.

SORT cataloged procedure

You can use the supplied SORT cataloged procedure when you include user routines that require binding or link-editing. Using this procedure without using bound or link-edited user routines is inefficient because the SORT cataloged procedure allocates linkage editor data sets whether or not you include user routines.

When you specify EXEC PROC=SORT or EXEC SORT, the following JCL statements are generated:

```
//SORT     EXEC  PGM=ICEMAN                                          00
//STEPLIB  DD    DSNAME=yyy,DISP=SHR                                 10
//SORTLIB  DD    DSNAME=xxx,DISP=SHR                                 20
//SYSOUT   DD    SYSOUT=A                                            30
//SYSPRINT DD    DUMMY                                               40
//SYSLMOD  DD    DSNAME=&GOSET,UNIT=SYSDA,SPACE=(3600,(20,20,1))     50
//SYSLIN   DD    DSNAME=&LOADSET,UNIT=SYSDA,SPACE=(80,(10,10))       60
//SYSUT1   DD    DSNAME=&SYSUT1,SPACE=(1024,(60,20)),                70
//               UNIT=(SYSDA,SEP=(SORTLIB,SYSLMOD,SYSLIN))          80
```

Line
Explanation

00
The stepname of the procedure is SORT. This EXEC statement initiates the program, which is named ICEMAN.

10
The STEPLIB DD statement defines the data set containing the DFSORT program modules. If DFSORT was installed as part of the normal system link libraries, the STEPLIB DD statement is unnecessary. It is needed only if DFSORT resides in a separate link library which is not part of the "link list." (Your installation's system programmers can give you this information.) The STEPLIB DD statement shown assumes that the data set name represented by yyy is cataloged.

20
The SORTLIB DD statement defines a private data set containing the modules needed for a sort using tape work files or a merge using the Conventional technique. The data set is cataloged, and the data set name represented by xxx was specified at installation time; it can be SYS1.SORTLIB.

If the modules were installed in a system library and installation option SORTLIB=SYSTEM is used, the SORTLIB DD statement is unnecessary and is ignored unless dynamic link of user exits is used.

30
Defines an output data set for system use (messages). It is directed to system output class A.

40
Defines SYSPRINT as a dummy data set because linkage editor diagnostic output is not required.

50
Defines a data set for linkage editor output. Any system disk device is acceptable for the output. Space for 20 records with an average length of 3600 bytes is requested; this is the primary allocation. Space for 20 more records is requested if the primary space allocation is not sufficient; this is the secondary allocation, which is requested each time primary space is exhausted. The last value is space for a directory, which is required because SYSLMOD is a new partitioned data set.

60
The SYSLIN data set is used by the program for linkage editor control statements. It is created on any system disk device, and it has space for 10 records with an average length of 80 bytes. If the primary space allocation is exhausted, additional space is requested in blocks large enough to contain 10 records. No directory space is necessary.

70/80
The SYSUT1 DD statement defines a work data set for the linkage editor.

SORTD cataloged procedure

You can use the supplied SORTD cataloged procedure when you do not include user routines or when you include user routines that do not require binding or link-editing.

When you specify EXEC PROC=SORTD or EXEC SORTD, the following JCL statements are generated:

```
//SORT  EXEC  PGM=ICEMAN                          00
//STEPLIB DD  DSNAME=yyy,DISP=SHR                 10
//SORTLIB DD  DSNAME=xxx,DISP=SHR                 20
//SYSOUT  DD  SYSOUT=A                            30
```

Line
Explanation

00
The stepname of the SORTD procedure is SORT

10
The STEPLIB DD statement defines the data set containing the DFSORT program modules. If DFSORT was installed as part of the normal system link libraries, the STEPLIB DD statement is unnecessary. It is needed only if DFSORT resides in a separate link library which is not part of the "link list." (Your installation's system programmers can give you this information.) The STEPLIB DD statement shown assumes that the data set name represented by yyy is cataloged.

20
The SORTLIB DD statement defines a private data set that contains the modules needed for a sort using tape work files or a merge that uses the Conventional technique. The data set name of the program subroutine library, represented by xxx, is specified at installation time; it can be SYS1.SORTLIB.

If the modules were installed in a system library and installation option SORTLIB=SYSTEM is used, then the SORTLIB DD statement is unnecessary and is ignored unless dynamic link edit of user exits is used.

30
Directs messages to system output class A

Specifying EXEC/DFSPARM PARM options

When you invoke DFSORT with JCL, you can specify some DFSORT options on the PARM parameter of the EXEC statement as illustrated on the following page. These options include EFS, LIST, NOLIST, LISTX, NOLISTX, MSGPRT, and MSGDDN, which are ignored if specified in an OPTION statement in SYSIN. Full override and applicability details are listed in Appendix B, “Specification/override of DFSORT options,” on page 805.

If you use the DFSPARM DD statement instead, you can specify both EXEC PARM options and DFSORT control statements in a single source data set that overrides all other sources. See “DFSPARM DD statement” on page 74.

Details of aliases for PARM options are given under the description of individual options. “Aliases for PARM options” on page 59 summarizes the available aliases.

DFSORT accepts but does not process the following EXEC/DFSPARM PARM options: BALANCE, BALN, BIAS=value, BMSG, CASCADE, CMP=value, CPU, CRCX, DEBUG, DIAG, ELAP, EXCPVR=value, IO, INCOR=value, INCORE=value, LRGSORT, L6=value, L7=value, NOCOMMAREA, NOINC, NOIOERR, NOSTIMER, OPT=value, OSCL, PEER, POLY, PRINT121 and STIMER.

Note: If DEBUG is specified as the first value in a DFSPARM statement, it will be interpreted as a DEBUG control statement rather than as a DFSPARM PARM option.

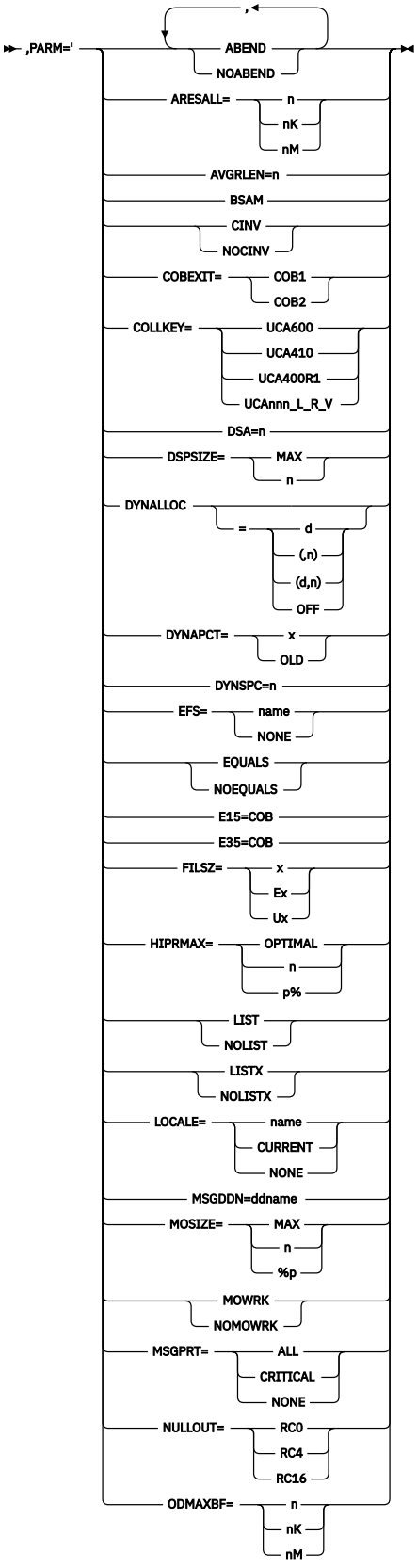
An option shown in the form parameter=value can also be specified in the equivalent form parameter(value) or parameter=(value). For example, the following are equivalent:

- DYNALLOC=SYSDA
- DYNALLOC(SYSDA)
- DYNALLOC=(SYSDA)

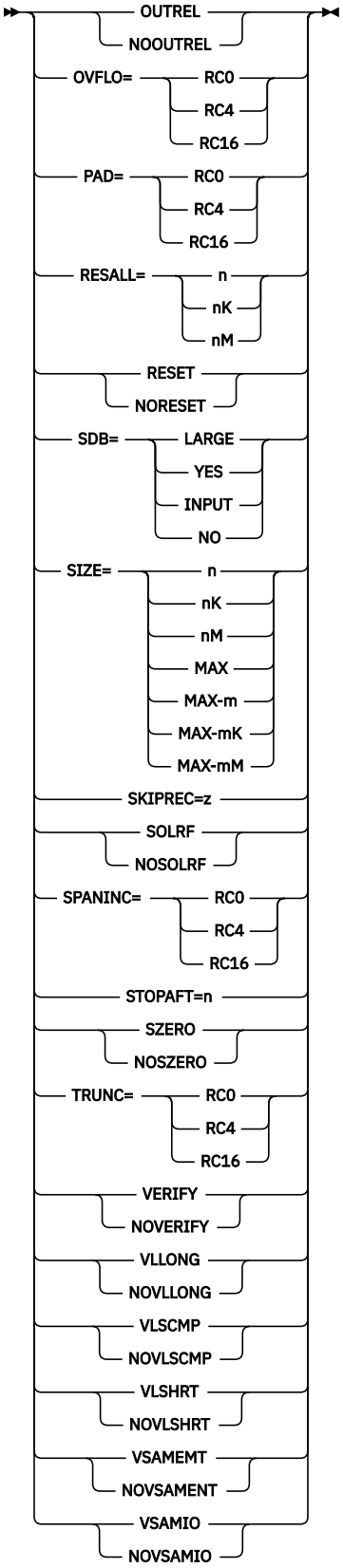
An option shown in the form parameter=(list) can also be specified in the equivalent form parameter(list). For example, the following are equivalent:

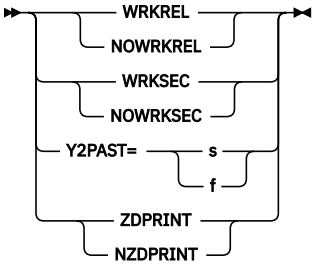
- DYNALLOC=(SYSDA,5)
- DYNALLOC(SYSDA,5)

Syntax Diagram for EXEC PARM

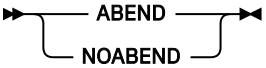


Additional


OUTREL
NOOUTREL
OVFLO=
RC0
RC4
RC16
PAD=
RC0
RC4
RC16
RESALL=
n
nK
nM
RESET
NORESET
SDB=
LARGE
YES
INPUT
NO
SIZE=
n
nK
nM
MAX
MAX-m
MAX-mK
MAX-mM
SKIPREC=z
SOLRF
NOSOLRF
SPANINC=
RC0
RC4
RC16
STOPAFT=n
SZERO
NOSZERO
TRUNC=
RC0
RC4
RC16
VERIFY
NOVERIFY
VLLONG
NOVLLONG
VLSCMP
NOVLSCMP
VLSHRT
NOVLSHRT
VSAMEMT
NOVSAMENT
VSAMIO
NOVSAMIO



ABEND or NOABEND



Temporarily overrides the ERET installation option, which specifies whether DFSORT abends or terminates with a return code of 16 if your sort, copy, or merge is unsuccessful.

ABEND

specifies that if your sort, copy, or merge is unsuccessful, DFSORT abends with a user completion code equal to the appropriate message number or with a user-defined number between 1 and 99, as set during installation with installation option ABCODE=n.

When DEBUG ABEND is in effect, a user abend code of zero may be issued when a tape work data set sort or Conventional merge is unsuccessful.

NOABEND

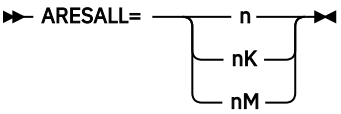
specifies that an unsuccessful sort, copy, or merge terminates with a return code of 16.

Note: RC16=ABE and NORC16 can be used instead of ABEND and NOABEND, respectively.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805

ARESALL



Temporarily overrides the ARESALL installation option, which specifies the number of bytes to be reserved above 16MB virtual for system use. For more information, see the discussion of the ARESALL option in “OPTION control statement” on page 173.

n
specifies that n bytes of storage are to be reserved.

Limit: 8 digits.

nK
specifies that n times 1024 bytes of storage are to be reserved.

Limit: 5 digits.

nM
specifies that n times 1048576 bytes of storage are to be reserved.

Limit: 2 digits.

Note: RESERVEX=value can be used instead of ARESALL=value.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

AVGRLEN

Specifies the average input record length in bytes for variable-length record sort applications. For more information, see the discussion of the AVGRLEN option in “OPTION control statement” on page 173.

n
specifies the average input record length. The value for n must be between 4 and 32767 and must include the 4 byte record descriptor word (RDW).

Note: L5=n can be used instead of AVGRLEN=n.

Default: If AVGRLEN=n is not specified, DFSORT will use one-half of the maximum record length as the average record length. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

BSAM

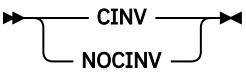
Temporarily bypasses the EXCP access method normally used for input and output data sets. BSAM is ignored for VSAM input and output data sets. Note that if Blockset is not selected and BSAM processing is used with concatenated SORTIN input and both null and non-null data sets are specified, all null data sets must precede all non-null data sets; otherwise, the results are unpredictable.

Attention: This option can degrade performance.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

CINV or NOCINV



Temporarily overrides the CINV installation option, which specifies whether DFSORT can use control interval access for VSAM data sets. For more information, see the explanation of the CINV option in “OPTION control statement” on page 173.

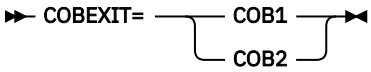
CINV
directs DFSORT to use control interval access when possible for VSAM data sets.

NOCINV
directs DFSORT not to use control interval access.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

COBEXIT



Temporarily overrides the COBEXIT installation option, which specifies the library for COBOL E15 and E35 routines.

COB1
specifies that COBOL E15 and E35 routines are run with the OS/VS COBOL run-time library or, in some cases, with no COBOL run-time library.

COBEXIT=COB1 is **obsolete**, but is still available for compatibility reasons.

Note that Language Environment® is the only run-time library for COBOL supported by IBM service.

COB2
specifies that COBOL E15 and E35 routines are run with either the VS COBOL II run-time library or the Language Environment run-time library.

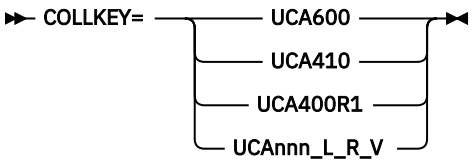
Note that Language Environment is the only run-time library for COBOL supported by IBM service.

Note: The DFSORT documents only discuss the Language Environment run-time library, and assume that COBEXIT=COB2 is in effect. Although it is possible to run with older run-time libraries, and with COBEXIT=COB1, these are not recommended or discussed, and are not supported by IBM service.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

COLLKEY



Temporarily overrides the COLLKEY installation option, which specifies the collation rules.

UCA600
This collation version supports the Unicode Standard character suite 6.0.0 and uses Normalization Service under 6.0.0 Unicode character suite.

UCA410
This collation version supports the Unicode Standard character suite 4.1.0 and uses Normalization Service under 4.1.0 Unicode character suite.

UCA400R1
This collation version supports the Unicode Standard character suite 4.0.0 and uses Normalization Service under 4.0.1 Unicode character suite.

UCAnnn_L_R_V
Specifies a collation version, where specific collation rules will modify any of the default Unicode Collation tables specified (UCA400R1, UCA410, or UCA600). Collation versions are set when you specify the following fields:

L
- Language (Specify a language for desired locale.)

R
- Region (Specify a region for desired locale.)

V
- Variant (Specify a variant for desired locale.)

Note:

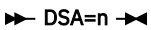
- For supported collation version settings (Language/Region/Variant), see Appendix E. Locales for collation in *z/OS Unicode Services User's Guide and Reference*
- If there is no collation information, installation default collation version will be set as default without any change.

Unicode Locales repository data set name SYS1.SCUNLOCL contains a set of locales documented in Locales for collation support. All of those locales contain a section for Collation rules.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805

DSA



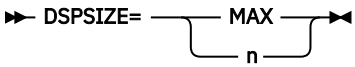
Temporarily overrides the DSA installation option, which specifies the maximum amount of storage available to DFSORT for dynamic storage adjustment of a Blockset sort application when SIZE/MAINSIZE=MAX is in effect. For more information, see the discussion of the DSA option in “OPTION control statement” on page 173".

n
specifies that DFSORT can dynamically adjust storage to improve performance, subject to a limit of **n** MB. n must be **n** value between 0 and 2000. If **n** is less than or equal to the TMAXLIM value in effect, **n** is set to 0 to indicate that storage will not be dynamically adjusted.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

DSPSIZE



Temporarily overrides the DSPSIZE installation option which specifies the maximum amount of data space to be used for dataspace sorting. For more information, see the discussion of the DSPSIZE option in “OPTION control statement” on page 173.

MAX
specifies that DFSORT dynamically determines the maximum amount of data space to be used for dataspace sorting. In this case, DFSORT bases its data space usage on the size of the file being sorted and the paging activity of the system.

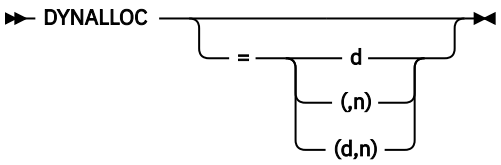
n
specifies the maximum amount, in megabytes, of data space to be used for dataspace sorting. n must be a value between 0 and 9999. The actual amount of data space used does not exceed n, but may be less depending on the size of the file being sorted and the paging activity of the system.

If n is zero, dataspace sorting is not used.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

DYNALLOC



Specifies that DFSORT dynamically allocates needed work space. You do not need to calculate and use JCL to specify the amount of work space needed by the program.

For more information, see the discussion of the DYNALLOC option in “OPTION control statement” on page 173 and Appendix A, “Using work space,” on page 797

d
specifies the device name. You can specify any IBM disk or tape device supported by your operating system in the same way you would specify it in the JCL UNIT parameter. You can also specify a group name, such as DISK or SYSDA.

n
specifies the maximum number of requested work data sets. If you specify more than 255, a maximum of 255 data sets is used. If you specify 1 and the Blockset technique is selected, a maximum of 2 data sets is used. If you specify more than 32 and the Blockset technique is not selected, a maximum of 32 data sets is used.

Note: For optimum allocation of resources such as virtual storage, avoid specifying a large number of work data sets unnecessarily.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

DYNALLOC=OFF

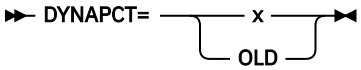
Directs DFSORT *not* to allocate intermediate workspace dynamically. It overrides installation option DYNAUTO=YES or the DYNALLOC parameter (without OFF) specified at run-time. For more information, see the discussion of the DYNALLOC option in “OPTION control statement” on page 173.

OFF
directs DFSORT not to allocate intermediate workspace dynamically.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

DYNAPCT



specifies additional work data sets to be dynamically allocated with zero space. DFSORT only extends these data sets when necessary to complete a sort application. The availability of additional work data sets can help avoid out of space ABENDs.

x
specifies the number of additional work data sets (y) as a percentage of the maximum number of dynamically allocated work data sets (DYNALLOC/DYNALOC n value) in effect. y will be set to n * x%. The total number of dynamically allocated work data sets will be n + y. For example, if DYNALLOC=(SYSDA,20) and DYNAPCT=20 are in effect, 4 additional work data sets will be allocated for a total of 24.

The value x must be between 0 and 254. The minimum value for y is 1 and the maximum value for y is 254. The maximum value for n + y is 255; if x results in a value for n + y greater than 255, y will be set to 255-n.

OLD
specifies additional work data sets should only be allocated when DFSORT cannot determine the file size. When DFSORT is able to determine the file size, additional work data sets will not be allocated (y=0), and the total number of work data sets will be n.

Note: When message ICE118I is issued indicating that DFSORT cannot determine the file size, y is set as follows:

- For DYNAPCT=OLD, y is set to n * 50%
- For DYNAPCT=x with x <= 50, y is set to n * 50%
- For DYNAPCT=x with x > 50, y is set to n * x%

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

DYNSPC

Temporarily overrides the DYNSPC installation option, which specifies the total default primary space allocation for all of the dynamically allocated work data sets when the input file size is unknown. That is, when DFSORT cannot determine the input file size for a sort application and the number of records is not supplied by a FILSZ or SIZE value. For more information, see the discussion of the DYNSPC option in “OPTION control statement” on page 173.

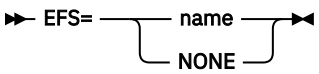
n
specifies the *total* default primary space, in megabytes, to be allocated for *all* dynamically allocated work data sets (n is *not* the primary space for each data set). n must be a value between 1 and 65535.

Do not specify a value which exceeds the available disk space, because this causes dynamic allocation to fail for sort applications that use this value.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

EFS



Temporarily overrides the EFS installation option, which specifies whether DFSORT passes control to an EFS program. See Chapter 9, “Using extended function support,” on page 717 for more information on EFS.

name
specifies the name of the EFS program that will be called to interface with DFSORT.

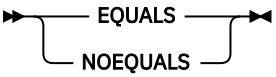
NONE
means no call will be made to the EFS program.

Note: If you use locale processing for SORT, MERGE, INCLUDE, or OMIT fields, you must not use an EFS program. DFSORT's locale processing may eliminate the need for an EFS program. See “OPTION control statement” on page 173 for information related to locale processing.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

EQUALS or NOEQUALS



Temporarily overrides the EQUALS installation option, which specifies whether the original sequence of records that collate identically for a sort or a merge should be preserved from input to output. For more information, see the discussion of the EQUALS and NOEQUALS options in “OPTION control statement” on page 173.

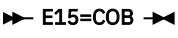
EQUALS
specifies that the original sequence must be preserved.

NOEQUALS
specifies that the original sequence need not be preserved.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

E15=COB

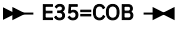


Specifies that your E15 routine is written in COBOL and temporarily overrides the MODS statement for E15. If you specify E15=COB but do not identify an E15 module with a MODS statement, the E15=COB is ignored.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

E35=COB

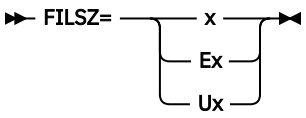


Specifies that your E35 routine is written in COBOL and temporarily overrides the MODS statement for E35. If you specify E35=COB but do not identify an E35 module with a MODS statement, the E35=COB is ignored.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

FILSZ



Specifies either the exact number of records to be sorted or merged, or an estimate of the number of records to be sorted. This record count is used by DFSORT for two purposes:

1. To check that the actual number of records sorted or merged is equal to the exact number of records expected. FILSZ=x causes this check to be performed and results in termination with message ICE047A if the check fails.
2. To determine the input file size for a sort application. DFSORT performs calculations based on the user supplied record count and other parameters (such as AVGRLEN) to estimate the total number of bytes to be sorted. This value is important for sort applications, since it is used for several internal optimizations as well as for dynamic work data set allocation (see OPTION DYNALLOC). If no input record count (or only an estimate) is supplied for the sort application, DFSORT attempts to automatically compute the file size to be used for the optimizations and allocations.

The type of FILSZ value specified (x, Ex, Ux, or none) controls the way DFSORT performs the previous two functions, and can have a significant effect on performance and work data set allocation. See “Specify input/output data set characteristics accurately” on page 749 and “Allocation of work data sets” on page 799 for more information on file size considerations.

x

specifies the exact number of records to be sorted or merged. This value is always used for both the record check and file size calculations. FILSZ=x can be used to force DFSORT to perform file size calculations based on x, and to cause DFSORT to terminate the sort or merge application if x is not exact.

If installation option FSZEST=NO is in effect and FILSZ=x is specified, DFSORT terminates if the actual number of records is different from the specified value (x), the actual number of records placed in the IN field of message ICE047A (or message ICE054I) before termination. However, if installation option FSZEST=YES is in effect, DFSORT treats FILSZ=x like FILSZ=Ex; it does not terminate when the actual number of records does not equal x.

The specified value (x) must take into account the number of records in the input data sets, records to be inserted or deleted by exit E15 or E32, and records to be deleted by the INCLUDE/OMIT statement, SKIPREC, and STOPAFT. x must be changed whenever the number of records to be sorted or merged changes in any way.

FILSZ=0 causes Hipersorting, dataspace sorting, and dynamic allocation of work space not to be used, and results in termination with the message ICE047A unless the number of records sorted or merged is 0.

Limit: 28 digits (15 significant digits)

Ex

specifies an estimated number of records to be sorted. This value is not used for the record check. It is used for file size calculations, but only if DFSORT could not automatically compute the file size. In all other cases, this value is ignored by DFSORT. See “Dynamic allocation of work data sets” on page 800 for details on exactly when FILSZ=Ex is used or ignored by DFSORT.

The specified value (x) should take into account the number of records in the input data sets, records to be inserted or deleted by exit E15, and records to be deleted by the INCLUDE/OMIT statement, SKIPREC, and STOPAFT. x should be changed whenever the number of records to be sorted changes significantly.

FILSZ=E0 will always be ignored.

Limit: 28 digits (15 significant digits)

Ux

specifies the number of records to be sorted. This value is not used for the record check, but is always used for file size calculations. FILSZ=Ux can be used to force DFSORT to perform file size calculations based on x, while avoiding termination if x is not exact.

The FSZEST installation option has no effect on FILSZ=Ux processing.

The specified value (x) should take into account the number of records in the input data sets, records to be inserted or deleted by exit E15, and records to be deleted by the INCLUDE/OMIT statement, SKIPREC, and STOPAFT. x should be changed whenever the number of records to be sorted changes significantly.

FILSZ=U0 causes Hipersorting, dataspace sorting, and dynamic allocation of work space not to be used, and can cause degraded performance or termination with the message ICE046A, if the actual number of records to be sorted is significantly larger than 0.

Limit: 28 digits (15 significant digits)

Table 5 on page 43 summarizes: the differences for the three FILSZ variations. Note that FILSZ=n is equivalent to FILSZ=En if installation option FSZEST=YES is specified.

Table 5. FILSZ Variations Summary.

Conditions	**FILSZ=n**	**FILSZ=Un**	**FILSZ=En**
Number of records	Exact	Estimate	Estimate
Applications	Sort, merge	Sort	Sort
Terminate if wrong?	Yes	No	No
Use for file size calculation?	Yes	Yes	When DFSORT cannot compute file size
n includes records:			
In input data sets	Yes	Yes	Yes
Inserted/deleted by E15	Yes	Yes	Yes
Inserted by E32	Yes	No	No
Deleted by INCLUDE/OMIT statement	Yes	Yes	Yes
Deleted by SKIPREC	Yes	Yes	Yes
Deleted by STOPAFT	Yes	Yes	Yes

Table 5. FILSZ Variations Summary. (continued)

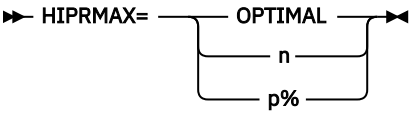
Conditions	FILSZ=n	FILSZ=Un	FILSZ=En
Update n when number of records changes:	In any way	Significantly	Significantly
Effects of n=0	Hipersorting and DYNALLOC not used	Hipersorting and DYNALLOC not used	None

Note: Using the FILSZ parameter to supply inaccurate information to DFSORT can negatively affect DFSORT's performance, and when work space is dynamically allocated, can result in wasted disk space or termination with message ICE083A or ICE046A. Therefore, it is important to update the record count value whenever the number of records to be sorted changes significantly.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

HIPRMAX



Temporarily overrides the HIPRMAX installation option, which specifies the maximum amount of Hiperspace to be used for Hipersorting. For more information, see the discussion of the HIPRMAX option in “OPTION control statement” on page 173.

OPTIMAL
specifies that DFSORT determines dynamically the maximum amount of Hiperspace to be used for Hipersorting.

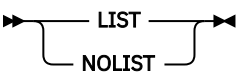
n
specifies that DFSORT determines dynamically the maximum amount of Hiperspace to be used for Hipersorting, subject to a limit of nMB. n must be a value between 0 and 32767. If n is 0, Hipersorting is not used.

%p
specifies that DFSORT determines dynamically the maximum amount of hiperspace to be used for Hipersorting, subject to a limit of p percent of an appropriate portion of central storage. p must be a value between 0 and 100. If p is 0, Hipersorting is not used. The value calculated for p% is limited to 32767MB, and is rounded down to the nearest MB.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

LIST or NOLIST



Temporarily overrides the LIST installation option, which specifies whether DFSORT program control statements should be written to the message data set. See *z/OS DFSORT Messages, Codes and Diagnosis Guide* for full details on use of the message data set.

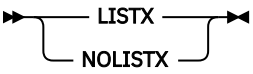
LIST
specifies that all DFSORT control statements are printed on the message data set.

NOLIST
specifies that DFSORT control statements are not printed.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

LISTX or NOLISTX



Temporarily overrides the LISTX installation option, which specifies whether DFSORT writes to the message data set the program control statements returned by an EFS program. See *z/OS DFSORT Messages, Codes and Diagnosis Guide* for full details on use of the message data set.

LISTX
specifies that control statements returned by an EFS program are printed to the message data set.

NOLISTX
specifies that control statements returned by an EFS program are not printed to the message data set.

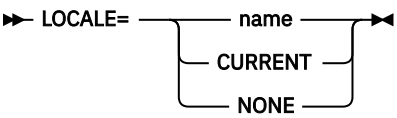
Note:

1. If EFS=NONE is in effect after final override rules have been applied, NOLISTX will be set in effect.
2. LISTX and NOLISTX can be used independently of LIST and NOLIST.
3. For more information on printing EFS control statements, see *z/OS DFSORT Messages, Codes and Diagnosis Guide*.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

LOCALE



Temporarily overrides the LOCALE installation option, which specifies whether locale processing is to be used and, if so, designates the active locale. For more information, see the discussion of the LOCALE option in “OPTION control statement” on page 173.

name
specifies that locale processing is to be used and designates the name of the locale to be made active during DFSORT processing.

The locales are designated using a descriptive name. For example, to set the active locale to represent the French language and the cultural conventions of Canada, specify LOCALE=FR_CA. You can specify up to 32 characters for the descriptive locale name. The locale names themselves are not case-sensitive. See "Locale naming conventions" in *z/OS XL C/C++ Programming Guide* for complete locale naming conventions.

You can use IBM-supplied and user-defined locales.

The state of the active locale prior to DFSORT being entered will be restored on DFSORT's completion.

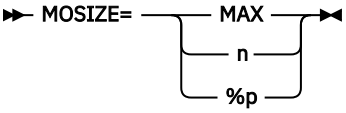
CURRENT
specifies that locale processing is to be used, and the current locale active when DFSORT is entered will remain the active locale during DFSORT processing.

NONE
specifies that locale processing is not to be used. DFSORT will use the binary encoding of the code page defined for your data for collating and comparing.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

MOSIZE



Temporarily overrides the MOSIZE installation option, which specifies the maximum amount of memory object storage to be used for memory object sorting. For more information, see the discussion of the MOSIZE option in “OPTION control statement” on page 173.

MAX
specifies that DFSORT determines dynamically the maximum amount of memory object storage to be used for memory object sorting.

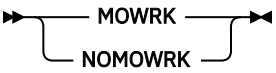
n
specifies that DFSORT determines dynamically the maximum amount of memory object storage to be used for memory object sorting, subject to a limit of nMB. n must be a value between 0 and 2147483646. If n is 0, memory object sorting is not used.

%p
specifies that DFSORT determines dynamically the maximum amount of memory object storage to be used for memory object sorting, subject to a limit of p percent of the available central storage. p must be a value between 0 and 100. If p is 0, memory object sorting is not used. The value calculated for p% is limited to 2147483646MB, and is rounded down to the nearest MB.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

MOWRK or NOMOWRK



Temporarily overrides the MOWRK installation option, which specifies whether the memory object storage available to DFSORT for memory object sorting can be used as intermediate work space. DFSORT has the capability of using memory object storage as intermediate work space (similar to the way Hiperspace is used but more efficient), or as an extension of main storage. Using memory object storage as intermediate work space is the preferred and recommended choice, but can be disabled, if appropriate.

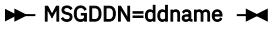
MOWRK
specifies that memory object storage can be used as intermediate work space, or as an extension of main storage, as appropriate.

NOMOWRK
specifies that memory object storage can only be used as an extension of main storage.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

MSGDDN



Temporarily overrides the MSGDDN installation option, which specifies an alternate ddname for the message data set. For more information, see the discussion of the MSGDDN option in "OPTION control statement" on page 173.

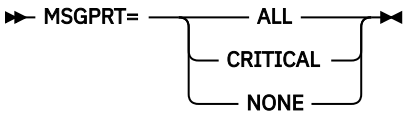
The ddname can be any 1- through 8-character name, but must be unique within the job step; do not use a name that is used by DFSORT (for example, SORTIN). If the ddname specified is not available at run-time, SYSOUT is used instead. For details on using the message data set, see *z/OS DFSORT Messages, Codes and Diagnosis Guide*.

Note: MSGDD=ddname can be used instead of MSGDDN=ddname.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

MSGPRT



Temporarily overrides the MSGPRT installation option, which specifies the class of messages to be written to the message data set. See *z/OS DFSORT Messages, Codes and Diagnosis Guide* for full details on use of the message data set.

ALL
specifies that all messages except diagnostic messages ICE800I to ICE999I are printed on the message data set. Control statements are printed only if LIST is in effect.

CRITICAL
specifies that only critical messages are printed on the message data set. Control statements are printed only if LIST is in effect.

NONE
specifies that no messages or control statements are printed.

Note: The forms FLAG(I)|FLAG(U)|NOFLAG, and MSG={NO|NOF|AB|AP|AC|CB|CC|CP|PC|SC|SP} are also accepted. The following table lists the equivalent MSGPRT/MSGCON specifications for these alternate forms:

Table 6. Aliases for MSGPRT/MSGCON Options

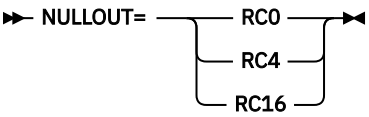
Option	MSGPRT	MSGCON
NO	NONE	NONE
NOF	NONE	NONE
AB	ALL	ALL
AP	ALL	CRITICAL

Table 6. Aliases for MSGPRT/MSGCON Options (continued)		
Option	**MSGPRT**	**MSGCON**
AC	NONE	ALL
CB	CRITICAL	CRITICAL
CC	NONE	CRITICAL
CP	CRITICAL	CRITICAL
PC	ALL	ALL
SC	ALL	CRITICAL
SP	CRITICAL	ALL
NOFLAG	NONE	CRITICAL
FLAG(I)	ALL	CRITICAL
FLAG(U)	CRITICAL	CRITICAL

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

NULLOUT



Temporarily overrides the NULLOUT installation option, which specifies the action to be taken by DFSORT when there are no records for the SORTOUT data set. For more information, see the discussion of the NULLOUT option in “OPTION control statement” on page 173.

RC0
specifies that DFSORT should issue message ICE173I, set a return code of 0, and continue processing when there are no records for the SORTOUT data set.

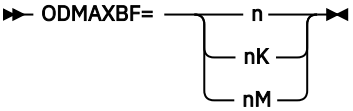
RC4
specifies that DFSORT should issue message ICE173I, set a return code of 4, and continue processing when there are no records for the SORTOUT data set.

RC16
specifies that DFSORT should issue message ICE206A, terminate, and give a return code of 16 when there are no records for the SORTOUT data set.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

ODMAXBF



Temporarily overrides the ODMAXBF installation option, which specifies the maximum buffer space DFSORT can use for each OUTFIL data set. For more information, see the discussion of the ODMAXBF option in "OPTION control statement" on page 173.

n
specifies that a maximum of n bytes of buffer space is to be used for each OUTFIL data set. If you specify less than 262144, 262144 is used. If you specify more than 16777216, 16777216 is used.

Limit: 8 digits

nK
specifies that a maximum of n times 1024 bytes of buffer space is to be used for each OUTFIL data set. If you specify less than 256K, 256K is used. If you specify more than 16384K, 16384K is used.

Limit: 5 digits

nM
specifies that a maximum of n times 1048576 bytes of buffer space is to be used for each OUTFIL data set. If you specify 0M, 256K is used. If you specify more than 16M, 16M is used.

Limit: 2 digits

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

OUTREL or NOOUTREL

Temporarily overrides the OUTREL installation option, which specifies whether unused temporary output data set space is to be released.

OUTREL
specifies that unused temporary output data set space is released.

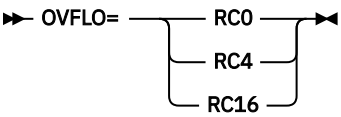
NOOUTREL
specifies that unused temporary output data set space is not released.

Note: RLSOUT and NORLSOUT can be used instead of OUTREL and NOOUTREL, respectively.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

OVFLO



Temporarily overrides the OVFLO installation option, which specifies the action to be taken by DFSORT when BI, FI, PD or ZD summary fields overflow. For more information, see the discussion of the OVFLO option in "OPTION control statement" on page 173.

RC0
specifies that DFSORT should issue message ICE152I (once), set a return code of 0 and continue processing when summary fields overflow.

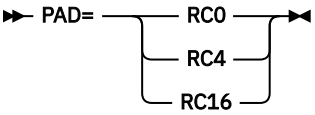
RC4
specifies that DFSORT should issue message ICE152I (once), set a return code of 4 and continue processing when summary fields overflow.

RC16
specifies that DFSORT should issue message ICE195A, terminate and give a return code of 16 when summary fields overflow.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

PAD



Temporarily overrides the PAD installation option, which specifies the action to be taken by DFSORT when the SORTOUT LRECL is larger than the SORTIN/SORTINnn LRECL, for the cases where DFSORT allows LRECL padding. For more information, see the discussion of the PAD option in "OPTION control statement" on page 173.

RC0
specifies that DFSORT should issue message ICE171I (once), set a return code of 0 and continue processing when the SORTOUT LRECL is larger than the SORTIN/SORTINnn LRECL.

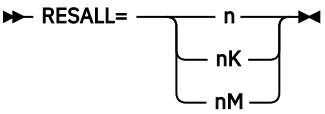
RC4
specifies that DFSORT should issue message ICE171I, set a return code of 4 and continue processing when the SORTOUT LRECL is larger than the SORTIN/SORTINnn LRECL.

RC16
specifies that DFSORT should issue message ICE196A, terminate and give a return code of 16 when the SORTOUT LRECL is larger than the SORTIN/SORTINnn LRECL.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

RESALL



Temporarily overrides the RESALL installation option, which specifies the number of bytes to be reserved in a REGION for system use when SIZE/MAINSIZE=MAX is in effect. For more information, see the discussion of the RESALL option in "OPTION control statement" on page 173.

n
specifies that n bytes of storage are to be reserved. If you specify less than 4096, 4096 is used.

Limit: 8 digits.

nK
specifies that n times 1024 bytes of storage are to be reserved. If you specify less than 4K, 4K is used.

Limit: 5 digits.

nM
specifies that n times 1048576 bytes of storage are to be reserved. If you specify 0M, 4K is used.

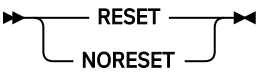
Limit: 2 digits.

Note: RESERVE=value can be used instead of RESALL=value.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

RESET or NORESET



Temporarily overrides the RESET installation option, which specifies whether DFSORT should process a VSAM output data set defined with REUSE as a NEW or MOD data set.

RESET
specifies that DFSORT processes a VSAM output data set defined with REUSE as a NEW data set. The high-used RBA is reset to zero and the output data set is effectively treated as an initially empty cluster.

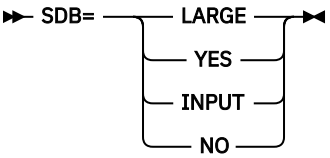
NORESET
specifies that DFSORT processes a VSAM output data set defined with REUSE as a MOD data set. The high-used RBA is not reset and the output data set is effectively treated as an initially non-empty cluster.

Note: A VSAM output data set defined without REUSE is processed as a MOD data set.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

SDB



Temporarily overrides the SDB installation option, which specifies whether DFSORT should use the system-determined optimum block size for output data sets when the block size is specified as zero or defaulted to zero. For more information, see the discussion of the SDB option in "OPTION control statement" on page 173.

LARGE
specifies that DFSORT is to use the system-determined optimum block size for an output data set when its block size is zero. SDB=LARGE allows DFSORT to select a block size greater than 32760 bytes for a tape output data set, when appropriate.

YES
specifies that DFSORT is to use the system-determined optimum block size for an output data set when its block size is zero, but is to limit the selected block size to a maximum of 32760 bytes.

INPUT
specifies that DFSORT is to use the system-determined optimum block size for an output data set when its block size is zero, but is to limit the selected block size to a maximum of 32760 bytes if the input block size is less than or equal to 32760 bytes.

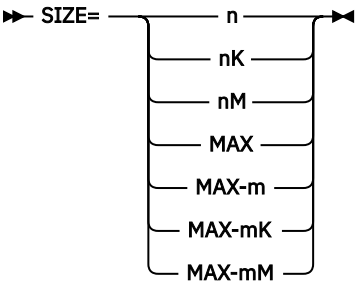
NO
specifies that DFSORT is not to use the system-determined optimum block size.

Note: SDB, SDB=ON, and SDB=SMALL can be used instead of SDB=YES. NOSDB and SDB=OFF can be used instead of SDB=NO.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

SIZE



Temporarily overrides the SIZE installation option, which specifies the amount of main storage available to DFSORT. For more information, see the discussion of the MAINSIZE option in "OPTION control statement" on page 173.

n
specifies that n bytes of storage are to be allocated. If you specify more than 2097152000, 2097152000 is used.

Limit: 10 digits.

nK
specifies that n times 1024 bytes of storage are to be allocated. If you specify more than 2048000K, 2048000K is used.

Limit: 7 digits.

nM
specifies that n times 1048576 bytes of storage are to be allocated. If you specify more than 2000M, 2000M is used.

Limit: 4 digits.

MAX
instructs DFSORT to calculate the amount of virtual storage available and allocate an amount of storage up to the TMAXLIM or DSA value when Blockset is selected, or up to the MAXLIM value when Blockset is not selected.

MAX-m
specifies the RESALL value (m) in bytes. MAX-m instructs DFSORT to calculate the amount of storage available and allocate this amount up to the MAX value *minus* the amount of storage reserved for system and application use (RESALL).

If you specify less than 4096 for m, 4096 is used.

Limit for m: 8 digits.

MAX-mK
specifies the RESALL value (m times 1024) in KBs. MAX-mK instructs DFSORT to calculate the amount of storage available and allocate this amount up to the MAX value *minus* the amount of storage reserved for system and application use (RESALL).

If you specify less than 4K for m, 4K is used.

Limit for m: 5 digits.

MAX-mM
specifies the RESALL value (m times 1048576) in s. MAX-mM instructs the program to calculate the amount of storage available and allocate this amount up to the MAX value *minus* the amount of storage reserved for system and application use (RESALL).

If you specify 0M for m, 4K is used.

Limit for m: 2 digits.

Note: The forms SIZE(value), CORE=value, and CORE(value) can be used instead of SIZE=value.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

SKIPREC

Specifies the number of records (z) you want to skip (delete) before starting to sort or copy the input data set. SKIPREC is typically used to bypass records not processed from the previous DFSORT job. For more information, see the discussion of the SKIPREC option in “OPTION control statement” on page 173.

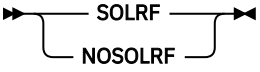
z
specifies the number of records to be skipped.

Limit: 28 digits (15 significant digits).

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

SOLRF or NOSOLRF



Temporarily overrides the SOLRF installation option, which specifies whether DFSORT should set the SORTOUT LRECL to the reformatted record length when the SORTOUT LRECL is unknown. For more information, see the discussion of the SOLRF and NOSOLRF options in “OPTION control statement” on page 173.

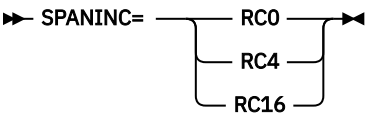
SOLRF
specifies that DFSORT should use the reformatted record length for the SORTOUT LRECL when the SORTOUT LRECL is not specified or available.

NOSOLRF
specifies that DFSORT should not use the reformatted record length for the SORTOUT LRECL.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

SPANINC



Temporarily overrides the SPANINC installation option, which specifies the action to be taken by DFSORT when one or more incomplete spanned records are detected in a variable spanned input data set. For more information, see the discussion of the SPANINC option in "OPTION control statement" on page 173.

RC0
specifies that DFSORT should issue message ICE197I (once), set a return code of 0 and eliminate all incomplete spanned records it detects.

RC4
specifies that DFSORT should issue message ICE197I (once), set a return code of 4 and eliminate all incomplete spanned records it detects.

RC16
specifies that DFSORT should issue message ICE204A, terminate and give a return code of 16 when an incomplete spanned record is detected.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805

STOPAFT

Specifies the maximum number of records you want accepted for sorting or copying (that is, read from SORTIN or inserted by E15 and not deleted by SKIPREC, E15, or an INCLUDE/OMIT statement). For more information, see the discussion of the STOPAFT option in "OPTION control statement" on page 173.

n
specifies the maximum number of records to be accepted.

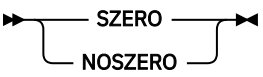
Limit: 28 digits (15 significant digits).

Note: If you specify (1) FILSZ=x in the EXEC PARM, or (2) SIZE=x or FILSZ=x on the OPTION or SORT statement, and the number of records accepted for processing does not equal x, DFSORT issues an error message and terminates unless installation option FSZEST=YES was specified.

Default: None; optional. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

SZERO or NOSZERO



Temporarily overrides the SZERO installation option, which specifies-+ whether DFSORT should treat numeric 0 and 0 values as signed (that is, different) or unsigned (that is, the same) for collation, comparisons, editing, conversions, minimums and maximums. For more information, see the discussion of the SZERO and NOSZERO options in "OPTION control statement" on page 173.

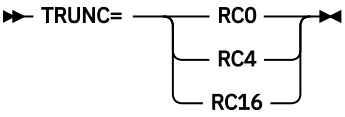
SZERO
specifies that DFSORT should treat numeric zero values as signed.

NOSZERO
specifies that DFSORT should treat numeric zero values as unsigned.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

TRUNC



Temporarily overrides the TRUNC installation option, which specifies the action to be taken by DFSORT when the SORTOUT LRECL is smaller than the SORTIN/SORTINnn LRECL, for the cases where DFSORT allows LRECL truncation. For more information, see the discussion of the TRUNC option in “OPTION control statement” on page 173.

RC0
specifies that DFSORT should issue message ICE171I, set a return code of 0 and continue processing when the SORTOUT LRECL is smaller than the SORTIN/SORTINnn LRECL.

RC4
specifies that DFSORT should issue message ICE171I, set a return code of 4 and continue processing when the SORTOUT LRECL is smaller than the SORTIN/SORTINnn LRECL.

RC16
specifies that DFSORT should issue message ICE196A, terminate and give a return code of 16 when the SORTOUT LRECL is smaller than the SORTIN/SORTINnn LRECL.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

VERIFY or NOVERIFY

Temporarily overrides the VERIFY installation option, which specifies whether sequence checking of the final output records must be performed.

VERIFY
specifies that sequence checking is performed.

NOVERIFY
specifies that sequence checking is not performed.

Note:

1. Using VERIFY can degrade performance.
2. SEQ=YES can be used instead of VERIFY. SEQ=NO can be used instead of NOVERIFY.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

VLLONG or NOVLLONG

Temporarily overrides the VLLONG installation option, which specifies whether DFSORT is to truncate "long" variable-length output records. For more information, see the discussion of the VLLONG and NOVLLONG options in “OPTION control statement” on page 173.

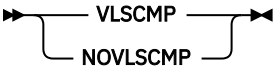
VLLONG
specifies that DFSORT truncates long variable-length output records to the LRECL of the SORTOUT or OUTFIL data set.

NOVLLONG
specifies that DFSORT terminates if a long variable-length output record is found.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

VLSCMP or NOVLSCMP



Temporarily overrides the VLSCMP installation option, which specifies whether DFSORT is to pad "short" variable-length INCLUDE/OMIT compare fields with binary zeroes. For more information, see the discussion of the VLSCMP and NOVLSCMP options in “OPTION control statement” on page 173.

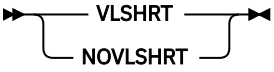
VLSCMP
specifies that short variable-length compare fields are padded with binary zeros.

NOVLSCMP
specifies that short variable-length compare fields are not padded.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

VLSHRT or NOVLSHRT



Temporarily overrides the VLSHRT installation option, which specifies whether DFSORT is to continue processing if a "short" variable-length SORT/MERGE control field, INCLUDE/OMIT compare field, or SUM summary field is found. For more information, see the discussion of the VLSHRT and NOVLSHRT options in “OPTION control statement” on page 173.

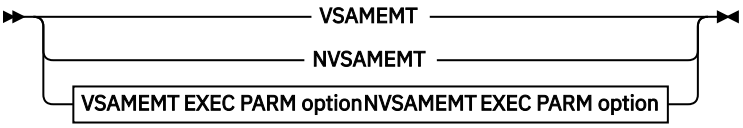
VLSHRT
specifies that DFSORT continues processing if a short control field or compare field is found.

NOVLSHRT
specifies that DFSORT terminates if a short control field or compare field is found.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

VSAMEMT or NVSAMEMT



Temporarily overrides the VSAMEMT installation option, which specifies whether DFSORT should accept an empty VSAM input data set.

VSAMEMT
specifies that DFSORT accepts an empty VSAM input data set and processes it as having zero records.

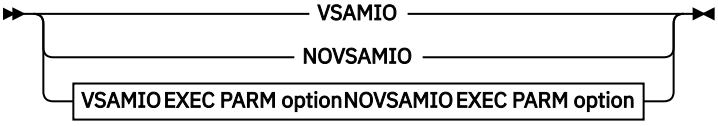
NVSAMEMT
specifies that DFSORT terminates if an empty VSAM input data set is found.

Note: VSAMEMT=YES can be used instead of VSAMEMT. VSAMEMT=NO can be used instead of NVSAMEMT.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805

VSAMIO or NOVSAMIO



Temporarily overrides the VSAMIO installation option, which specifies whether DFSORT should allow a VSAM data set defined with REUSE to be sorted in-place.

VSAMIO
specifies that DFSORT can use the same VSAM data set for input and output: when all of the following conditions are met

- The application is a sort.
- RESET is in effect.
- The VSAM data set was defined with REUSE.

These conditions ensure that the VSAM data set is processed as NEW for output and will contain the sorted input records; that is it will be sorted in-place.

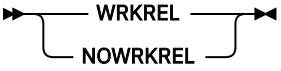
DFSORT terminates if the same VSAM data set is specified for input and output, and any of the previous conditions are not met.

NOVSAMIO
specifies that DFSORT terminates if the same VSAM data set is specified for input and output.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

WRKREL or NOWRKREL



Temporarily overrides the WRKREL installation option, which specifies whether unused temporary SORTWKdd data set space will be released.

WRKREL
specifies that unused space is released.

NOWRKREL
specifies that unused space is not released.

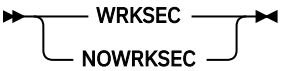
Note:

1. If you have dedicated certain volumes for SORTWKdd data sets, and you do not want unused temporary space to be released, you should specify NOWRKREL.
2. If WRKREL is in effect, DFSORT releases space for the SORTWKdd data sets just prior to termination. Space is released only for those SORTWKdd data sets that were used for the sort application.
3. RELEASE=OFF and RLS=0 can be used instead of NOWRKREL. RELEASE=ON and RLS=n (n greater than 0) can be used instead of WRKREL.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

WRKSEC or NOWRKSEC



Temporarily overrides the WRKSEC installation option, which specifies whether DFSORT uses automatic secondary allocation for temporary JCL SORTWKdd data sets.

WRKSEC
specifies that automatic secondary allocation for temporary JCL SORTWKdd data sets is used and that 25 percent of the primary allocation will be used as the secondary allocation.

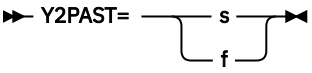
NOWRKSEC
specifies that automatic secondary allocation for temporary JCL SORTWKdd data sets is not used.

Note: SECOND=OFF and SEC=0 can be used instead of NOWRKSEC. SECOND=ON and SEC=n (n greater than 0) can be used instead of WRKSEC.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

Y2PAST



Temporarily overrides the Y2PAST installation option, which specifies the sliding (s) or fixed (f) century window. The century window is used with DFSORT's Y2 formats to correctly interpret two-digit year data values as four-digit year data values.

s
specifies the number of years DFSORT is to subtract from the current year to set the beginning of the sliding century window. Since the Y2PAST value is subtracted from the current year, the century window slides as the current year changes. For example, Y2PAST=81 would set a century window of 1925-2024 in 2006 and 1926-2025 in 2007. s must be a value between 0 and 100.

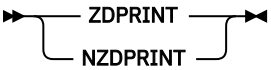
f
specifies the beginning of the fixed century window. For example, Y2PAST=1962 would set a century window of 1962-2061. f must be a value between 1000 and 3000.

Note: CENTWIN=value can be used instead of Y2PAST=value.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

ZDPRINT or NZDPRINT



Temporarily overrides the ZDPRINT installation option, which specifies whether positive zoned-decimal (ZD) fields resulting from summing must be converted to printable numbers. For more information, see the discussion of the ZDPRINT and NZDPRINT options in “OPTION control statement” on page 173.

ZDPRINT
means convert positive ZD summation results to printable numbers.

NZDPRINT
means do not convert positive ZD summation results to printable numbers.

Note: ZDPRINT=YES can be used instead of ZDPRINT. ZDPRINT=NO can be used instead of NZDPRINT.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

Aliases for PARM options

For compatibility reasons, the following EXEC/DFSPARM PARM options can be specified by using the aliases shown in the following table. See the indicated PARM options for complete details.

Table 7. Aliases for PARM Options

Aliases for PARM OptionsAlias	PARM Option
CENTWIN=value	Y2PAST=value
CORE=value	SIZE=value
FLAG(I)	MSGPRT=ALL
FLAG(U)	MSGPRT=CRITICAL
L5=value	AVGRLEN=value
MSG=value	MSGPRT=value
MSGDD=value	MSGDDN=value
NOFLAG	MSGPRT=NONE
NORC16	NOABEND
NORLSOUT	NOOUTREL
NOSDB	SDB=NO
RC16=ABE	ABEND

Table 7. Aliases for PARM Options (continued)

Aliases for PARM OptionsAlias	PARM Option
RELEASE=ON	WRKREL
RELEASE=OFF	NOWRKREL
RESERVE=value	RESALL=value
RESERVEX=value	ARESALL=value
RLS=n	WRKREL
RLS=0	NOWRKREL
RLSOUT	OUTREL
SDB	SDB=YES
SDB=ON	SDB=YES
SDB=OFF	SDB=NO
SDB=SMALL	SDB=YES
SEC=n	WRKSEC
SEC=0	NOWRKSEC
SECOND=ON	WRKSEC
SECOND=OFF	NOWRKSEC
SEQ=YES	VERIFY
SEQ=NO	NOVERIFY
VSAMEMT=YES	VSAMEMT
VSAMEMT=NO	NVSAMEMT
ZDPRINT=YES	ZDPRINT
ZDPRINT=NO	NZDPRINT

Using DD statements

A DFSORT job always requires DD statements after the EXEC statement. DD: statements fall into two categories

- System DD statements (discussed in detail in “System DD statements” on page 63)
- Program DD statements (discussed in detail in “Program DD statements” on page 64).

System DD statements, and some program DD statements, are usually supplied automatically when you use a cataloged procedure. Others you must always supply yourself.

The DD statement parameters, the conditions under which they are required, and the default values, are summarized in Table 8 on page 61. The subparameters of the DCB parameter (a DD statement parameter) are described similarly in Table 9 on page 61.

Note:

1. Performance is enhanced if the LRECL subparameter of the DCB is accurately specified for variable-length records. The maximum input record length you can specify for your particular configuration is given in “Data set notes and limitations” on page 13.
2. When using DFSORT applications, FREE=CLOSE cannot be used on any DD statements except DFSPARM.

Table 8. DD Statement Parameters Used by DFSORT

Parameter	When Required	Parameter Values	Default Value
{AMP | BUFSP}	When password-protected VSAM data sets are used and the password is supplied through E18, E38, or E39.	Minimum buffer pool value given when creating the data set.	None.
DCB	Required when 7-track tape is used; for input on tape without standard labels; and when the default values are not applicable.	Specifies information used to fill the data control block (DCB) associated with the data set.	(See separate subparameters in Table 9 on page 61.)
DISP	When the default value is not applicable.	Indicates the status and disposition of the data set.	The system assumes (NEW, DELETE).
DSNAME or DSN	When the DD statement defines a labeled input data set (for example, SORTIN), or when the data set being created is to be kept or cataloged (for example, SORTOUT), or passed to another step.	Specifies the fully qualified or temporary name of the data set.	The system assigns a unique name.
LABEL	When the default value is not applicable.	Specifies information about labeling and retention for the data set.	The system assumes standard labeling.
SPACE	When the DD statement defines a new data set on disk.	Specifies the amount of space needed to contain the data set.	See *z/OS MVS JCL Reference*.
UNIT	When the input data set is neither cataloged nor passed or when the data set is being created.	Specifies (symbolically or actually) the type and quantity of I/O units required by the data set.	See *z/OS MVS JCL Reference*.
VOLUME or VOL	When the input data set is neither cataloged nor passed, for multireel input or when the output data set is on disk and is to be kept or cataloged.	Specifies information used to identify the volume or volumes occupied by the data set.	See *z/OS MVS JCL Reference*.

Table 9. DCB Subparameters Used by DFSORT

Subparameter	Condition When Required	Subparameter Values	Default Value
BUFOFF	When processing data in ASCII format.	Specifies the length of the buffer offset or specifies that the buffer offset is the block length indicator.	
DEN	When the data set is located on a 7-track tape unit.	Specifies the density at which the tape was recorded.	800 bpi
OPTCD	When processing data in ASCII format.	Specifies that the tape processed is in ASCII format.	
TRTCH	When the data set is located on a tape device with IDRC and system IDRC is not used.	Specifies whether data set is compacted.	System default option.

Table 9. DCB Subparameters Used by DFSORT (continued)

Subparameter	Condition When Required	Subparameter Values	Default Value
BLKSIZE [1, 2]	When the DCB parameter is required and the default value is not suitable except on SORTWKdd statements.	Specifies the maximum length (in bytes) of the physical records in the data set.	• For old data sets, the value in the data set label. • For new output data sets, appropriate values based on the input data set or RECORD statement values. Unless SDB=NO is in effect, Blockset uses the system-determined optimum block size when the output data set block size is zero. Applications which require a specific output data set block size should be changed to specify that block size **EXPLICITY**. • No default if input on unlabeled tape or BLP or NSL specified.
LRECL [2, 3]		Specifies the maximum length (in bytes) of the logical records in the data set.	
RECFM		Specifies the format of the records in the data set.	

Duplicate ddnames

If you specify a particular ddname (such as SORTIN) more than once within the same step, DFSORT uses the first ddname and ignores subsequent duplicates. Processing continues normally.

In addition, SORTIN0, SORTIN1...SORTIN9 can be specified *instead of* SORTIN00, SORTIN01...SORTIN09, respectively. If you specify both SORTINn and SORTIN0n in the same job step, DFSORT treats them as duplicates, and ignores each usage after the first. For example, SORTIN2 and SORTIN02 are treated as duplicates and only SORTIN2 is used.

Note: For a Conventional merge, SORTINn will not be recognized because of the existing restriction which allows only SORTIN01, SORTIN02...SORTIN16. Duplicates of these accepted ddnames will be ignored.

Duplicate OUTFIL ddnames are ignored at the OUTFIL statement level as explained in "OUTFIL statements notes" on page 360.

Shared tape units

The following pairs of DFSORT data sets can be assigned to a single tape unit:

- The SORTIN data set and the SORTWK01 data set (tape work data set sorts only)
- The SORTIN data set and the SORTOUT data set or one OUTFIL data set (sort applications only).

If you want to associate the SORTIN data set with SORTWK01, you can include the parameter UNIT=AFF=SORTIN in the DD statement for SORTWK01. The AFF subparameter causes the system to place the data set on the same unit as the dataset with the ddname following the subparameter (SORTIN, in this case).

In the same way, you can associate the SORTIN data set with the SORTOUT data set or an OUTFIL data set by including UNIT=AFF=SORTIN in the SORTOUT or OUTFIL DD statement.

[1] See "SORTIN DD statement" on page 66 and "SORTINnn DD statement" on page 68.
[2] This is the only subparameter allowed for DD * data sets.
[3] For padding and truncating fixed-length records, see "Data set notes and limitations" on page 13.

SORTINnn tape data sets must all be on different tape units because they are read concurrently. SORTOUT and OUTFIL tape data sets must all be on different tape units because they are written concurrently.

System DD statements

If you choose not to use the SORT or SORTD cataloged procedures to invoke DFSORT, you might need to supply system DD statements in your input job stream (See also the following section for DD statements dedicated to DFSORT, such as SORTIN). The DD statements contained in the cataloged procedure (or provided by you) are:

//JOBLIB DD

Defines your program link library if it is not already known to the system.

//STEPLIB DD

Same as //JOBLIB DD.

//SYSIN DD

Contains DFSORT control statements, comment statements, blank statements and remarks when DFSORT is invoked with JCL rather than by another program. It can also contain user exit routines, in object deck format, to be bound or link-edited by DFSORT.

- If you use DFSPARM, then SYSIN is not necessary unless your job requires binding or link-editing.
- The SYSIN data set usually resides in the input stream; however, it can be defined as a sequential data set or as a member of a partitioned data set.
- The data set must be defined with a RECFM of F or FB. The LRECL can be 80, or more (when valid). If the LRECL is greater than 80, DFSORT will use the first 80 bytes of each record.

  If user exit routines are in SYSIN, the LRECL must be 80.
- DFSORT supports concatenated SYSIN data sets to the extent that the system supports "like" concatenated data sets for BSAM. Refer to *z/OS DFSMS Using Data Sets* for further information about "like" concatenated data sets.

Note: The OPTION statement keywords EFS, LIST, NOLIST, LISTX, NOLISTX, LOCALE, MSGPRT, MSGDDN, SMF, SORTDD, SORTIN, and SORTOUT are used only when they are passed by an extended parameter list or when in the DFSPARM data set. If they are specified on an OPTION statement read from the SYSIN or SORTCNTL data set, the keyword is recognized, but the parameters are ignored.

If you use the DFSPARM DD statement instead, you can specify both EXEC PARM options and DFSORT control statements in a single source data set that overrides all other sources. See “DFSPARM DD statement” on page 74.

If user exit routines are in SYSIN, make sure that:

- The LRECL of SYSIN is 80.
- The END statement is the last *control* statement.
- The user exit routines are arranged in numeric order (for example, E11 before E15).
- The user exit routines are supplied immediately after the END control statement.
- Nothing follows the last object deck in SYSIN.
- A SORTMODS DD statement is included.

If DFSORT is program invoked, and you supply the DFSORT control statements through the 24-bit or extended parameter list, SORTCNTL, or DFSPARM, SYSIN remains the source of user exit routines placed in the system input stream.

//SYSOUT DD

Identifies the DFSORT message data set. The default ddname is SYSOUT, but you can specify an alternate ddname for the message data set using the MSGDDN installation or run-time option. Always supply a DD statement for the message data set if a catalogued procedure is not used. (If you are invoking DFSORT from a COBOL program and are using the ddname SYSOUT for the message data set, the use of DISPLAY in your COBOL program can produce uncertain printing results.)

DFSORT uses RECFM=FBA, LRECL=121, and the specified BLKSIZE for the message data set. If the BLKSIZE you specify is not a multiple of 121, DFSORT uses BLKSIZE=121. If you do not specify the BLKSIZE, DFSORT selects the block size as directed by the SDBMSG installation option (see *z/OS DFSORT Installation and Customization*).

If you use a temporary or permanent message data set, it is best to specify a disposition of MOD to ensure you see all messages and control statements in the message data set.

//SYSUDUMP DD

Defines the data set for output from a system ABEND dump routine.

//SYSMDUMP DD

Same as //SYSUDUMP DD.

//SYSABEND DD

Same as //SYSUDUMP DD.

If you are using the supplied SORT cataloged procedure, the DD statements that follow are automatically supplied. If you are not using the SORT cataloged procedure and you are using the linkage editor, you must supply the following: DD statements

//SYSPRINT DD

Contains messages from the binder or linkage editor.

//SYSUT1 DD

Defines the intermediate storage data set for the linkage editor. The binder does not use this data set.

//SYSLIN DD

Defines a data set for control information for the binder or linkage editor.

//SYSLMOD DD

Defines a data set for output from the binder or linkage editor.

Note: If you do not include user routines, or if you include user routines that do *not* require binding or link-editing, you can use the supplied SORTD cataloged procedure. If you include user routines that require binding or link-editing, you can use the SORT cataloged procedure.

Program DD statements

Even if you use the SORT or SORTD cataloged procedure to invoke DFSORT, you might need to supply additional dedicated DD statements. The following list summarizes each of these statements, and a more detailed explanation of each one follows.

//SORTLIB DD

Defines the data set that contains special load modules for DFSORT. Can usually be omitted.

//SYMNAMES DD

Defines the SYMNAMES data set containing statements to be used for symbol processing. Required only if symbol processing is to be performed.

//SYMNOUT DD

Defines the data set in which SYMNAMES statements and the symbol table are to be listed. Optional if SYMNAMES DD is specified. Otherwise ignored.

//SORTIN DD

Defines the input data set for a sorting or copying application. Will not be used for a merging application.

//SORTINnn DD

Defines the input data sets for a merging application. Will not be used for a sorting or copying application.

//SORTWKdd DD

Defines intermediate storage data sets. Usually needed for a sorting application unless dynamic allocation is requested. Will not be used for a copying or merging application.

//SORTOUT DD

Defines the SORTOUT output data set for a sorting, merging, or copying application.

//outfil DD

Defines an OUTFIL output data set for a sorting, merging, or copying application.

//SORTCKPT DD

Defines the data set used to store the information that the system needs to restart the sort from the last checkpoint. This is only needed if you are using the checkpoint facility.

//SORTCNTL DD

Defines the data set from which additional or changed DFSORT control statements can be read when DFSORT is program-invoked.

//DFSPARM DD

Defines the data set from which both additional or changed DFSORT program control statements and EXEC statement PARM options can be read when DFSORT is directly invoked or program invoked.

//SORTDKdd DD

Defines the data set used for a VIO SORTWKdd allocation by DFSORT if it is dynamically reallocated; SORTDKdd must never be specified in the job stream.

//SORTDIAG DD

Specifies that all messages and control statements are printed. Used primarily for diagnostics and debugging.

//SORTSNAP DD

Defines the snap dump data set dynamically allocated by DFSORT. SORTSNAP must never be specified in the job stream.

//SORTMODS DD

Defines a temporary partitioned data set. This temporary data set must be large enough to contain all your user exit routines that appear in SYSIN for a given application. If none of your routines appear in SYSIN, this statement is not required. If your routines are in libraries, you must include DD statements defining the libraries.

DFSORT temporarily transfers the user exit routines in SYSIN to the data set defined by this DD statement. Then DFSORT calls the binder or linkage editor for processing.

SORTLIB DD statement

The SORTLIB DD statement can usually be omitted. This statement describes the data set that contains special DFSORT load modules.

When Required

If installation option SORTLIB=PRIVATE is in effect or dynamic link edit of user exits is specified

- For sort applications using tape work data sets
- For merge applications for which Blockset cannot be used (see message ICE800I).

The SORTLIB installation option determines whether DFSORT searches a system library or private library for the load modules required by tape work data set sorts and Conventional merges.

Example 1 SORTLIB DD Statement

```
//SORTLIB DD DSNAME=USORTLIB,DISP=SHR
```

This example shows DD statement parameters that define a previously cataloged input data set:

DSNAME
causes the system to search the catalog for a data set with the name USORTLIB. When the data set is found, it is associated with the ddname SORTLIB. The control program obtains the unit assignment and volume serial number from the catalog and, if the volume is not already mounted, writes a mounting message to the operator.

DISP
indicates that the data set existed before this job step, that it should be kept after this job step, and that it can be used concurrently by several jobs (SHR). None of the jobs should change the data set in any way.

For information on the parameters used in the SORTLIB DD statement, the conditions under which they are required, and the default values assumed if a parameter is not included, see Table 8 on page 61. The subparameters of the DCB parameter are described in the same detail in Table 9 on page 61. For more detailed information, see *z/OS MVS JCL Reference* and *z/OS MVS JCL User's Guide*

SYMNAMES DD and SYMNOUT DD statements

See Chapter 8, “Using symbols for fields and constants,” on page 685 for details.

SORTIN DD statement

The SORTIN DD statement describes the characteristics of the data set in which the records to be sorted or copied reside and also indicates its location.

When Required

If installation option SORTLIB=PRIVATE is in effect or dynamic link edit of user exits is specified

- For sort applications using tape work data sets
- For merge applications for which Blockset cannot be used (see message ICE800I).

The SORTLIB installation option determines whether DFSORT searches a system library or private library for the load modules required by tape work data set sorts and Conventional merges.

Data set characteristics

DFSORT accepts empty and null non-VSAM data sets, and DUMMY data sets, for sorting and copying (be sure to supply RECFM, LRECL and BLKSIZE). DFSORT also accepts empty VSAM data sets for sorting and copying provided VSAMEMT is in effect. For non-VSAM data sets, DFSORT examines the DS1LSTAR field in the format-1 DSCB to determine whether the data set is empty or null. If DS1LSTAR is zero, DFSORT treats the data set as empty or null. If the data set is a null multivolume data set and the DS1IND80 flag is off in the format-1 DSCB of the first volume of the multivolume data set, DFSORT opens the data set for output to force an end of file (EOF) mark before using the data set for input.

Note that a null data set is one that has been newly created, but never successfully closed. Null data sets cannot be processed successfully for a tape work data set sort. The "System Code" field in the data set label in the disk Volume Table of Contents (DSCB in the VTOC) indicates a data set created by the VSE operating system if it contains the letters DOS or VSE within it. Such data sets are never treated as null; however, they may be empty. DFSORT cannot process VSE disk data sets that do not have DOS or VSE within the System Code field.

DFSORT may set what it considers to be appropriate values for missing attributes (RECFM, LRECL, BLKSIZE) of input data sets based on other attributes, or may terminate due to a missing attribute. If a missing attribute results in termination, or you don't want to use a missing attribute set by DFSORT, specify that attribute explicitly (for example, specify RECFM=VB).

See “Data set considerations ” on page 13 for additional considerations.

The following rules apply to concatenated data sets:

- RECFM must be either all fixed-length or all variable-length for the data sets in the concatenation.
- BLKSIZE can vary. However, if a tape data set has the largest block size and is not first in the concatenation, you must specify BLKSIZE explicitly on its DD statement in the following two situations:
  - Blockset is selected and the tape data set has a block size greater than 32760 bytes, but the block size is not available from DFSMSrmm or ICETPEX.
  - Blockset is not selected.
- With fixed-length records, LRECL must be the same for all data sets. With variable-length records, LRECL can vary. However:
  - If Blockset is selected: If a tape data set has the largest LRECL and is not first in the concatenation, you must specify LRECL explicitly on its DD statement if the LRECL is not available from DFSMSrmm or ICETPEX.
  - If Blockset is not selected, the first data set in the concatenation must have the largest LRECL (LRECL can be specified explicitly on its DD statement).
- If the data sets are on unlike devices, you cannot use the EXLST parameter at user exit E18.
- If Blockset is not selected and BSAM is used, all null data sets must precede all non-null data sets; otherwise, the results are unpredictable.
- DFSORT forces an EOF mark on all null data sets whose format-1 DSCB DS1IND80 flag is off before using BSAM to process the null data sets.
- If you define a data set using the DUMMY parameter, do not concatenate other data sets to it; the system ignores data sets concatenated to a DUMMY data set.
- VSAM data sets must not be concatenated (system restriction).
- Input cannot consist of both VSAM and non-VSAM data sets.

General coding notes

- For a copy application, the SORTIN data set should not be the same as the SORTOUT data set or any OUTFIL data set because this can cause lost or incorrect data or unpredictable results.

- For a sort application, the SORTIN data set should not be the same as any SORTWKdd data set because this can cause lost or incorrect data or unpredictable results. The SORTIN data set can be the same as the SORTOUT data set or an OUTFIL data set, but this situation can lead to the loss of the data set if the sort application does not end successfully.
- FREE=CLOSE cannot be specified. User labels are not copied.

Example 2 SORTIN DD statement

```
//SORTIN DD DSNAME=INPUT,DISP=SHR
```

This example shows DD statement parameters that define a previously cataloged input data set:

DSNAME
causes the system to search the catalog for a data set with the name INPUT. When the data set is found, it is associated with the ddname SORTIN. The control program obtains the unit assignment and volume serial number from the catalog and, if the volume is not already mounted, writes a mounting message to the operator.

DISP
indicates that the data set existed before this job step, that it should be kept after this job step, and that it can be used concurrently by several jobs (SHR). None of the jobs should change the data set in any way.

Example 3 volume parameter on SORTIN DD

```
//SORTIN   DD  DSN=SORTIN,DISP=(OLD,KEEP),UNIT=3490,
//              VOL=SER=(75836,79661,72945)
```

If the input data set is contained on more than one reel of magnetic tape, the VOLUME parameter must be included on the SORTIN DD statement to indicate the serial numbers of the tape reels. In this example, the input data set is on three reels that have serial numbers 75836, 79661, and 72945.

If a data set is not on a disk or on a standard-labeled tape, you must specify DCB parameters in its DD statement.

SORTINnn DD statement

The SORTINnn DD statements describe the characteristics of the data sets in which records to be merged reside and indicate the locations of these data sets.

When required

SORTINnn DD statements are always needed for a merge, unless the merge is invoked from another program and all input is supplied through a routine at user exit E32.

Data set characteristics

Input data sets can be either non-VSAM or VSAM, but not both. DFSORT accepts empty and null non-VSAM data sets, and DUMMY data sets, for merging (be sure to supply RECFM, LRECL and BLKSIZE). DFSORT also accepts empty VSAM data sets for merging provided VSAMEMT is in effect. For non-VSAM data sets, DFSORT examines the DS1LSTAR field in the format-1 DSCB to determine whether the data set is null or empty. If DS1LSTAR is zero, DFSORT treats the data set as null or empty. A null data set is one that has been newly created but never successfully closed. Null data sets cannot be processed successfully by the Conventional merge technique.

RECFM must be the same for all input data sets.

BLKSIZE can vary, but for a Conventional merge, SORTIN01 must specify the largest block size.

With fixed-length records, LRECL must be the *same* for all data sets. With variable-length records, LRECL can vary.

Data sets can be multivolume but not concatenated. If a SORTINnn data set is multivolume and null, DFSORT forces an EOF mark on the data set before use.

DFSORT may set what it considers to be appropriate values for missing attributes (RECFM, LRECL, BLKSIZE) of input data sets based on other attributes, or may terminate due to a missing attribute. If a missing attribute results in termination, or you don't want to use a missing attribute set by DFSORT, specify that attribute explicitly (for example, specify RECFM=VB).

See “Data set notes and limitations” on page 13 for additional considerations.

General coding notes

- A SORTINnn data set should not be the same as the SORTOUT data set or any OUTFIL data set because this can cause lost or incorrect data or unpredictable results.
- You can merge up to 100 data sets with Blockset merge or up to 16 data sets with Conventional merge. If Conventional merge is selected, check message ICE800I for the reason Blockset could not be used and correct the indicated condition, if possible.
  - With Blockset merge, nn can be any integer from 00 (the initial zero is optional) to 99, in any order. Blockset merge treats ddnames of the form SORTINn and SORTIN0n as duplicates, and ignores any occurrences after: the first. For example, if you have

    ```
    //SORTIN4 DD . . .
    //SORTIN04 DD . . .
    ```

    the SORTIN04 DD will be ignored.
  - With Conventional merge, nn can range from 01 to 16. The first number you use must be 01 and the remainder must follow in numeric order. Numbers cannot be skipped. Conventional merge cannot use ddnames of the form SORTIN0-SORTIN9, SORTIN00 or SORTIN17-SORTIN99.
- FREE=CLOSE cannot be specified. User labels are not copied.

Example 4 SORTIN01-03 DD statements (merge)

```
//SORTIN01 DD  DSNAME=MERGE1,VOLUME=SER=000111,DISP=OLD,
//             LABEL=(,NL),UNIT=3590,
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=32000)
//SORTIN02 DD  DSNAME=MERGE2,VOLUME=SER=000121,DISP=OLD,
//             LABEL=(,NL),UNIT=3590,
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=32000)
//SORTIN03 DD  DSNAME=MERGE3,VOLUME=SER=000131,DISP=OLD,
//             LABEL=(,NL),UNIT=3590,
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=32000)
```

Example 5 SORTIN01-02 DD statements (merge)

```
//SORTIN01 DD  DSNAME=INPUT1,VOLUME=SER=000101, *
//             UNIT=3390,DISP=OLD               *DCB PARAMETERS
//SORTIN02 DD  DSNAME=INPUT2,VOLUME=SER=000201, *SUPPLIED FROM
//             UNIT=3390,DISP=OLD               *LABELS
```

SORTWKdd DD statement

The SORTWKdd DD statements describe the characteristics of the data sets used as intermediate storage areas for records to be sorted; they also indicate the location of these data sets.

Up to 255 SORTWKdd DD statements can be specified. However, if you specify more than 32 and the Blockset technique is not selected, only the first 32 are used.

When required

One or more SORTWKdd statements are required for each sort application (but not a merge or copy), unless:

- Input can be contained in main storage

- Dynamic work space allocation has been requested (DYNALLOC)
- dataspace sorting, Hipersorting, or memory object sorting is used.

For information on using work data sets, see Appendix A, "Using work space," on page 797.

Diagnostic message ICE803I gives information on intermediate storage allocation and use.

Devices

SORTWKdd data sets can be on disk or on tape, but not both. Disk types can be mixed.

Tape must be nine-track unless input is on seven-track tape, in which case work tapes can (but need not) be seven-track.

General coding notes

- Unless the input file is very large, two or three SORTWKdd data sets are usually sufficient. Two or three large SORTWKdd data sets are preferable to several small data sets. Placing each SORTWKdd data set on a separate device can improve performance.

  For optimum allocation of resources such as virtual storage, avoid specifying a large number of work data sets unnecessarily.
- A SORTWKdd data set should not be the same as the SORTIN data set, the SORTOUT data set, any OUTFIL data set, or any other SORTWKdd data set because this can cause lost or incorrect data or unpredictable results.
- Cylinder allocation is preferable for performance reasons. Temporary SORTWKdd data sets allocated in tracks or blocks (without ROUND) are readjusted to cylinders by DFSORT.
- For disk work data sets, any valid ddname of the form SORTWKdd or SORTWKd can be used (for example, SORTWK01, SORTWKC3, SORTWK2, SORTWK#5, SORTWKA, SORTWKXY and so on). The ddnames can be in any order. SORTWKd and SORTWK0d are not treated as duplicate ddnames (for example, SORTWK5 and SORTWK05 will both be used if specified, as will SORTWKQ and SORTWK0Q). If you specify more than 255 ddnames and the Blockset technique is selected, only the first 255 ddnames are used. If you specify more than 32 ddnames and the Blockset technique is not selected, only the first 32 ddnames are used.
- For tape work data sets, at least three SORTWKdd data sets must be specified. The first three ddnames must be SORTWK01, SORTWK02 and SORTWK03. Subsequent ddnames, if specified, must be in order from SORTWK04 through SORTWK32, with no numbers skipped,
- FREE=CLOSE cannot be specified.
- Spool, dummy, subsystem, VSAM and z/OS UNIX file system data sets, and z/OS UNIX files, must not be specified as work data sets
- Parameters relating to ASCII data should not be included for tape work data sets.

Disk work data set coding notes

- Data sets must be physical sequential; they cannot be partitioned or extended format.
- The SPLIT cylinder parameter must not be specified.
- If no secondary allocation is requested for temporary SORTWKdd data sets, automatic secondary allocation will be used unless NOWRKSEC is in effect. (Secondary allocation is limited to 12 work data sets in the Peerage and Vale sorting techniques only.)
- If the data set is allocated to VIO, there is no automatic secondary allocation.
- Secondary allocation can be requested for work data sets. For the Peerage and Vale sorting techniques only, secondary allocation is limited to the first 12 work data sets; if more work data sets are defined, they are used with only the primary allocation.
- DFSORT uses *only* the space on the first volume specified for a multivolume data set. Space on the second and subsequent volumes is not used. Multivolume SORTWKdd data sets are, therefore, treated as single-volume SORTWKdd data sets.

- DFSORT work data sets can exceed 65536 tracks if allocated as large format data sets on disk devices. Dynamically allocated disk work data sets are automatically allocated by DFSORT as large format. DSNTYPE=LARGE must be specified on SORTWKdd DD statements to allocate JCL work data sets as large format.
- If primary space is fragmented, all but the first fragment are handled as secondary space.

Virtual I/O

If a SORTWKdd data set is specified on a virtual device:

- With VIO=NO: DFSORT performs dynamic reallocation using the ddname SORTDKdd on a real device with the same device type as the virtual device. If a real device corresponding to the virtual device is not available in the system, DFSORT terminates with an ICE083A message; see *z/OS DFSORT Messages, Codes and Diagnosis Guide* for more information about this error. Non-VIO SORTWKdd data sets are also reallocated when VIO SORTWKdd data sets are present.
- With VIO=YES: the virtual device is used; performance may be degraded.
- DFSORT work data sets cannot be allocated as large format data sets if VIO is used.

The following is an example of a SORTWKdd DD statement using a disk work: data set

Example 6 SORTWK01 DD statement, disk work data set

```
//SORTWK01  DD  SPACE=(CYL,(15,5)),UNIT=3390
```

If you use the checkpoint/restart facility and need to make a deferred restart, you must make the following additions to the previous statement so that the sort work data set is not lost:

```
DSNAME=name1,DISP=(NEW,DELETE,CATLG)
```

Thus the same SORTWKdd DD statement for a deferred restart would be:

```
//SORTWK01  DD  DSNAME=name1,UNIT=3390,SPACE=(CYL,(15,5)),
//              DISP=(NEW,DELETE,CATLG)
```

The following is an example of SORTWKdd DD statements using three tape devices.

Example 7 SORTWK01-03 DD statement, tape intermediate storage

```
//SORTWK01  DD  UNIT=3480,LABEL=(,NL)
//SORTWK02  DD  UNIT=3480,LABEL=(,NL)
//SORTWK03  DD  UNIT=3480,LABEL=(,NL)
```

If DFSORT terminates unsuccessfully and the previous DD statements have been specified, the intermediate storage data sets remain in the system until the step has been successfully rerun or until the data sets have been deleted by some other means.

These parameters specify unlabeled data sets on three 3480 tape units. Because the DSNAME parameters are omitted, the system assigns unique names.

SORTOUT and OUTFIL DD statements

The SORTOUT and OUTFIL DD statements describe the characteristics of the data sets in which the processed records are to be placed and indicate their location.

The SORTOUT DD statement specifies the single non-OUTFIL output data set for a sort, copy, or merge application. OUTFIL processing does not apply to SORTOUT.

The FNAMES and FILES parameters of one or more OUTFIL statements specify the ddnames of the OUTFIL data sets for a sort, copy, or merge application. The parameters specified for each OUTFIL statement define the OUTFIL processing to be performed for the OUTFIL data sets associated with that statement. For specific information about OUTFIL processing, see “OUTFIL control statements” on page 221.

Although the ddname SORTOUT can actually be used for an OUTFIL data set, the term "SORTOUT" will be used to denote the single non-OUTFIL output data set.

When required

Each ddname specified in an OUTFIL statement requires a corresponding DD statement for that OUTFIL data set.

If you do not specify OUTFIL statements, a SORTOUT DD statement is required unless you provide an E35 user exit that disposes of all output. A SORTOUT DD statement is ignored if your program invokes DFSORT and passes the address of an E35 user exit in the parameter list.

If you specify OUTFIL statements, you do not have to specify a SORTOUT DD statement or an E35 user exit, although you can use either or both.

Data set characteristics

See “Data set considerations ” on page 13 for additional considerations.

Block size

Unless SDB=NO is in effect, Blockset uses the system-determined optimum block size in most cases when the output data set block size is zero. See the discussion of the SDB option in “OPTION control statement” on page 173 for complete details about DFSORT's use of system-determined block size.

For some jobs, the selection of a larger output data set block size can require an increase in the amount of storage needed for successful DFSORT processing.

Applications which require a specific output data set block size should be changed to specify that block size **explicitly**.

If SDB=NO is in effect, DFSORT selects an appropriate (though not necessarily optimum) block size for the output data set based on the available attributes from the output data set, the input data set, and the RECORD statement. The output data set block size will not necessarily be the same as the input block size.

Reblockable indicator

DFSORT sets the reblockable indicator in the output data set label when

Blockset is selected and

- DFSORT sets the system-determined optimum block size for the output data set (see “Block size” on page 72) or
- Allocation sets the system-determined optimum block size for the output data set before DFSORT gets control.

General coding notes

- For a copy application, neither the SORTOUT data set nor any OUTFIL data set should be the same as the SORTIN data set because this can cause lost or incorrect data or unpredictable results.
- For a merge application, neither the SORTOUT data set nor any OUTFIL data set should be the same as any SORTINnn data set because this can cause lost or incorrect data or unpredictable results.
- For a sort application, the SORTOUT data set or an OUTFIL data set can be the same as the SORTIN data set, but this situation can lead to the loss of the data set if the sort application does not end successfully.
- An OUTFIL data set should not be the same as the SORTOUT data set or any other OUTFIL data set because this can cause lost or incorrect data or unpredictable results.
- Do not specify OPTCD=W for a full function IBM 3480 tape unit; it is overridden. For a 3480 operating in 3420 compatibility mode (specified as 3400-9), the OPTCD=W request is not overridden, but performance might be degraded.

- If no secondary allocation is requested for a temporary or new output data set, automatic secondary allocation will be used unless NOOUTSEC is in effect.
- The RECFM, LRECL, and BLKSIZE in a tape label are used only for a tape output data set with DISP=MOD, a DD volser present, and an AL, SL, or NSL label, when appropriate.
- FREE=CLOSE cannot be specified.
- See the discussion of the SOLRF and NOSOLRF options in "OPTION control statement" on page 173 for information related to the SORTOUT LRECL.

Example 8 SORTOUT DD statement

```
//SORTOUT DD  DSN=C905460.OUTPT,UNIT=3390,SPACE=(CYL,5),
//            DISP=(NEW,CATLG)
```

DISP
specifies the data set unknown to the operating system (NEW) and catalogs (CATLG) it under the name C905460.OUTPT.

DSNAME
specifies that the data set is called C905460.OUTPT.

SPACE
requests five cylinders of storage for the data set.

UNIT
Indicates that the data set is on a 3390.

SORTCKPT DD statement

The SORTCKPT data set can be allocated on any device that operates with the Basic Sequential Access Method (BSAM). Processing must be restarted only from the last checkpoint taken.

Example 9 SORTCKPT DD statement

```
//SORTCKPT DD  DSNAME=CHECK,VOLUME=SER=000123,
//             DSP=(NEW,KEEP),UNIT=3480
```

When you allocate the SORTCKPT data set, you must include at least one work data set.

If the CKPT operand is specified on the OPTION or SORT control statement, more intermediate storage could be required.

If you want to use the Checkpoint/Restart Facility, refer to "Checkpoint/restart " on page 847.

SORTCNTL DD statement

The SORTCNTL data set can be used to supply DFSORT control statements, comment statements, blank statements, and remarks when DFSORT is invoked from another program (written, for example, in COBOL or PL/I).

- The SORTCNTL data set usually resides in the input stream, but can be defined as a sequential data set or as a member of a partitioned data set.
- The data set must be defined with RECFM of F or FB. The LRECL can be 80, or more (when valid). If the LRECL is greater than 80, DFSORT will use the first 80 bytes of each record.
- DFSORT supports concatenated SORTCNTL data sets to the extent that the system supports "like" concatenated data sets for BSAM. Refer to *z/OS DFSMS Using Data Sets* for further information about "like" concatenated data sets.
- When DFSORT is invoked from a PL/I program, the SORTCNTL or DFSPARM data set must not be used to supply a new RECORD control statement.

Example 10 SORTCNTL DD statement

```
//SORTCNTL DD  *
  OPTION MAINSIZE=8M
```

Note:

1. The OPTION statement keywords EFS, LIST, NOLIST, LISTX, NOLISTX, LOCALE, MSGPRT, MSGDDN, SMF, SORTDD, SORTIN, and SORTOUT are used only when they are passed by an extended parameter list or when in the DFSPARM data set. If they are specified on an OPTION statement read from the SYSIN or SORTCNTL data set, the keyword is recognized, but the parameters are ignored.

   If your program invokes DFSORT more than once, you can direct DFSORT to read different versions of the SORTCNTL data set at each call. See the explanation of the SORTDD parameter in "OPTION control statement" on page 173.
2. If you use the DFSPARM DD statement instead of the SORTCNTL DD statement, you can specify both EXEC PARM options and DFSORT control statements in a single source data set that overrides all other sources. See "DFSPARM DD statement" on page 74. For override rules, see Appendix B, "Specification/override of DFSORT options," on page 805.

DFSPARM DD statement

The DFSPARM DD statement can be used to supply DFSORT program control statements and EXEC statement PARM options from a single DD source. Because statements in the DFSPARM data set are read whether DFSORT is program invoked or directly invoked, you can specify EXEC PARM options when invoking DFSORT from another program (unlike SORTCNTL). DFSPARM accepts all DFSORT program control statements and all EXEC statement PARM options (including those ignored by SYSIN and SORTCNTL) and any equivalent options specified on a DFSORT OPTION statement.

DFSPARM also accepts comment statements, blank statements, and remarks.

For examples of using DFSPARM when you call DFSORT from a program, see "Overriding DFSORT control statements from programs" on page 518.

Full override and applicability details are listed as follows and in Appendix B, "Specification/override of DFSORT options," on page 805.

- If you use DFSPARM, SYSIN is not necessary unless your job requires link-editing.
- The DFSPARM data set usually resides in the input stream, but it can be defined as a sequential data set or as a member of a partitioned data set.
- The data set must be defined with RECFM of F or FB. The LRECL can be 80, or more (when valid). If the LRECL is greater than 80, DFSORT will use the first 80 bytes of each record.
- DFSORT supports concatenated DFSPARM data sets to the extent that the system supports "like" concatenated data sets for BSAM. Refer to *z/OS DFSMS Using Data Sets* for further information about "like" concatenated data sets.
- When DFSORT is invoked from a PL/I program, the SORTCNTL or DFSPARM data set must not be used to supply a new RECORD control statement.

Note: The ddname DFSPARM is used throughout this document to refer to this data set source for EXEC PARM options and DFSORT program control statements. When your system programmers installed DFSORT, they might have changed this name to one more appropriate for your site with the PARMDDN installation option. However, DFSORT will always use a DFSPARM data set of present, unless a DD statement with the PARMDDN name is also present.

General coding notes

Coding of parameters in the DFSPARM DD statement follows the same rules used for the JCL EXEC statement PARM options and the program control statements specified in SYSIN or SORTCNTL. The following exceptions apply:

- Labels are not allowed.

- PARM options and program control statements cannot be mixed on the same line, but can be specified in any order on different lines.
- PARM options must be specified without the PARM= keyword and without quote marks.
- Commas (or semicolons) are accepted, but not required, to continue PARM options to another line.
- Leading blanks are not required for PARM options, but at least one leading blank is required for program control statements.

FREE=CLOSE can be used for applicable DFSPARM data sets (for example, with temporary and permanent sequential data sets, but not with DD * data sets).

When DFSORT is called from another program, FREE=CLOSE causes the DFSPARM data set to be released when DFSORT returns to the caller. This allows another DFSPARM data set to be used for a subsequent call.

For example, if a COBOL program contains three SORT verbs, the following would cause the control statements in DP1 to be used for the first SORT verb, the control statements in DP2 to be used for the second SORT verb, and the: control statements in DP3 to be used for the third SORT verb

```
//DFSPARM DD DSN=DP1,DISP=SHR,FREE=CLOSE
//DFSPARM DD DSN=DP2,DISP=SHR,FREE=CLOSE
//DFSPARM DD DSN=DP3,DISP=SHR,FREE=CLOSE
```

Without FREE=CLOSE, DP1 would be used for all three SORT verbs.

Example 11 DFSPARM DD statement

```
//DFSPARM  DD  *
  SORT FIELDS=(1,2,CH,A),STOPAFT=300
ABEND
  OPTION SORTIN=DATAIN
  STOPAFT=500
```

In this example, the DFSPARM DD data set passes a DFSORT SORT statement, the ABEND and STOPAFT parameters equivalent to specifying PARM=ABEND,STOPAFT=500 in a JCL EXEC statement, and a DFSORT OPTION statement.

Note:

1. SORT and OPTION are control statements. ABEND and STOPAFT=500 are PARM options.
2. The PARM option STOPAFT=500 overrides the SORT control statement option STOPAFT=300.
3. When PARMDDN=DFSPARM is specified or defaulted
   - if a //DFSPARM DD data set is available at run-time, DFSORT will use it
   - if a //DFSPARM DD data set is not available at run-time, DFSORT will use a //$ORTPARM DD data set if available.

   Thus with PARMDDN=DFSPARM, you can choose to specify either a //DFSPARM DD data set or a //$ORTPARM DD data set for a particular DFSORT application.
4. When PARMDDN=ddname is specified
   - if a //ddname DD data set is available at run-time, DFSORT will use it
   - if a //ddname DD data set is not available at run-time, DFSORT will use a //DFSPARM DD data set if available.

   Thus with PARMDDN=ddname, you can choose to specify either a //ddname DD data set or a //DFSPARM DD data set for a particular DFSORT application.

Example 12 DFSPARM DD statement

```
//DFSPARM DD *
  SORT FIELDS=(5,2,CH,D),SKIPREC=10
```

```
  STOPAFT=100,BSAM,SKIPREC=5
  OPTION SORTIN=DATAIN,SKIPREC=20
```

In this example, the DFSPARM DD data set contains a SORT program control statement, three PARM options on one line, and an OPTION program control statement.

Note: Because PARM options override program control statements, DFSORT uses SKIPREC=5 and ignores the other SKIPREC specifications.

For information on the parameters used in the DFSPARM DD statement, the conditions under which they are required, and any default values assumed if a parameter is omitted, see "Specifying EXEC/DFSPARM PARM options" on page 32 and Chapter 3, "Using DFSORT program control statements," on page 77.

SORTDKdd DD statement

SORTWKdd data sets can be assigned to VIO. If the VIO installation option is specified or defaults to NO, SORTWKdd data sets are deallocated and reallocated by DFSORT using SORTDKdd ddnames. SORTDKdd ddnames are reserved for use by DFSORT.

SORTDIAG DD statement

The SORTDIAG DD statement specifies that all messages, including diagnostic messages (ICE800I through ICE999I), and control statements are to be written to the message data set. The statement can be used for all DFSORT techniques and provides information on EXCP counts, intermediate storage allocation and use, and so on. The SORTDIAG DD statement has no effect on console messages. The statement is intended as a *diagnostic tool*.

When SORTDIAG is used, a SYSOUT DD statement or a ddname DD statement (where ddname is the alternate message data set ddname specified during installation or run-time) should be provided. If installation option NOMSGDD=QUIT is in effect and neither an alternate message data set ddname statement nor a SYSOUT ddname statement is provided, DFSORT terminates with a return code of 20.

Example 13 SORTDIAG DD statement

```
//SORTDIAG DD  DUMMY
```

SORTSNAP DD statement

The SORTSNAP DD statement defines the data set where the snap dumps requested by the ESTAE recovery routine, or the snap dumps requested before or after a call to an EFS program are printed. SORTSNAP is dynamically allocated by DFSORT whenever it is required. The ddname, SORTSNAP, is reserved for DFSORT.

Chapter 3. Using DFSORT program control statements

Using program control statements

Program control statements direct DFSORT in processing your records. Some program control statements are required while others are optional. You use the control statements to:

- Indicate whether a sort, merge, or copy is performed.
- Describe the control fields to be used.
- Indicate program exits for transferring control to your own routines.
- Describe DFSORT functions you want to have invoked.
- Describe input and output files.
- Indicate various options you want to use during processing.

You can supply program control statements to DFSORT from:

- A SYSIN data set
- A SORTCNTL data set
- A DFSPARM data set
- A 24-Bit parameter list
- An extended parameter list

See Appendix B, “Specification/override of DFSORT options,” on page 805 for an explanation of when to use each source.

This chapter begins with a summary of DFSORT program control statements and coding rules. A detailed description of each statement follows.

Control statement summary

Describing the primary task

The only required program control statement in a DFSORT application is a SORT, MERGE, or OPTION statement that specifies whether you want to sort, merge, or copy records. (Copying can be specified on any of the three statements.)

SORT

Describes control fields if you are coding a sort application, or specifies a copy application. Indicates whether you want ascending or descending order for the sort.

MERGE

Describes control fields if you are coding a merge application, or specifies a copy application. Indicates whether you want ascending or descending order for the merge.

OPTION

Overrides installation defaults (such as EQUALS, CHALT, and CHECK) and supplies optional information (such as DYNALLOC and SKIPREC). Can specify a copy application.

Including or omitting records

You can specify whether certain records are included in the output data sets or omitted from them.

INCLUDE

Specifies that only records whose fields meet certain criteria are included.

OMIT

Specifies that any records whose fields meet certain criteria are deleted.

OUTFIL

Specifies the records to be included or omitted in multiple output data sets.

Reformatting and editing records

You can modify individual records by deleting and reordering fields and inserting blanks, zeros, or constants.

INREC

Specifies how records are reformatted before they are sorted, copied, or merged.

OUTREC

Specifies how records are reformatted after they are sorted, copied, or merged.

OUTFIL

Specifies how records are reformatted in multiple output data sets.

Producing multiple output and reports and converting records

You can produce multiple output data sets and reports, convert variable-length records to fixed-length records, and convert fixed-length records to variable-length records.

OUTFIL
Specifies the output data sets and which records are to appear in each. Specifies how records are to be converted from variable-length to fixed-length or from fixed-length to variable-length.

Joining two files

You can perform various types of "join" applications on two files (F1 and F2) by one or more keys.

JOINKEYS
One JOINKEYS statement is required for each input file to indicate the ddname of the file, describe the keys, indicate whether the file is already sorted by those keys, and so on.

JOIN
An inner join is performed by default, but a JOIN statement can be used to specify a different type of join.

REFORMAT
Describes the fields from the two files to be included in the joined records, and optionally an indicator of where the key was found ('B' for both files, '1' for file1 only or '2' for file2 only.

The three JOINKEYS application control statements (JOINKEYS, JOIN and REFORMAT) are completely described in Chapter 4, “Using a JOINKEYS application for joining two files,” on page 439.

Invoking additional functions and options

You can use the remaining control statements to perform a variety of tasks.

ALTSEQ

Specifies changes to the ALTSEQ translation table to be used for SORT, MERGE, INCLUDE or OMIT fields with format AQ, for INREC, OUTREC, and OUTFIL fields with TRAN=ALTSEQ, and for INREC, OUTREC, and OUTFIL fields with format AQ in logical expressions.

DEBUG

Specifies various diagnostic options.

END

Causes DFSORT to discontinue reading SYSIN, SORTCNTL, or DFSPARM.

MODS

Specifies use of one or more user exit routines in a DFSORT application. See Chapter 5, "Using your own user exit routines," on page 467 for information about user exit routines.

RECORD

Can be used to supply length and type information.

SUM

Specifies that numeric summary fields in records with equal control fields are summed in one record and that the other records are deleted.

Using symbols

You can define and use a symbol for any field or constant in the following DFSORT control statements: INCLUDE, INREC, JOINKEYS, MERGE, OMIT, OUTFIL, OUTREC, REFORMAT, SORT and SUM. You can use a symbol for a number associated with various keywords in the following DFSORT control statements: INREC, JOINKEYS, MERGE, OPTION, OUTFIL, OUTREC and SORT. You can also use a symbol for an output column in the following DFSORT control statements: INREC, OUTFIL and OUTREC. This makes it easy to create and reuse collections of symbols (that is, mappings) representing information associated with various record layouts. You can use system symbols (for example, &JOBNAME.) in your symbol constants. You can use SET and PROC symbols in your symbol constants. See Chapter 8, "Using symbols for fields and constants," on page 685 for complete details.

General coding rules

See "Inserting comment statements" on page 83 for an explanation of how to use comment statements, blank statements, and remarks. DFSORT program control statements and EXEC PARM options can also be specified together in a user-defined DD data set. See "DFSPARM DD statement" on page 74 for special coding conventions that apply to this DD source.

All other DFSORT control statements have the same general format, shown in Figure 4 on page 80. The illustrated format does *not* apply to control statements you supply in a parameter list. See Chapter 6, "Invoking DFSORT from a program," on page 517 for information on the special rules that apply.

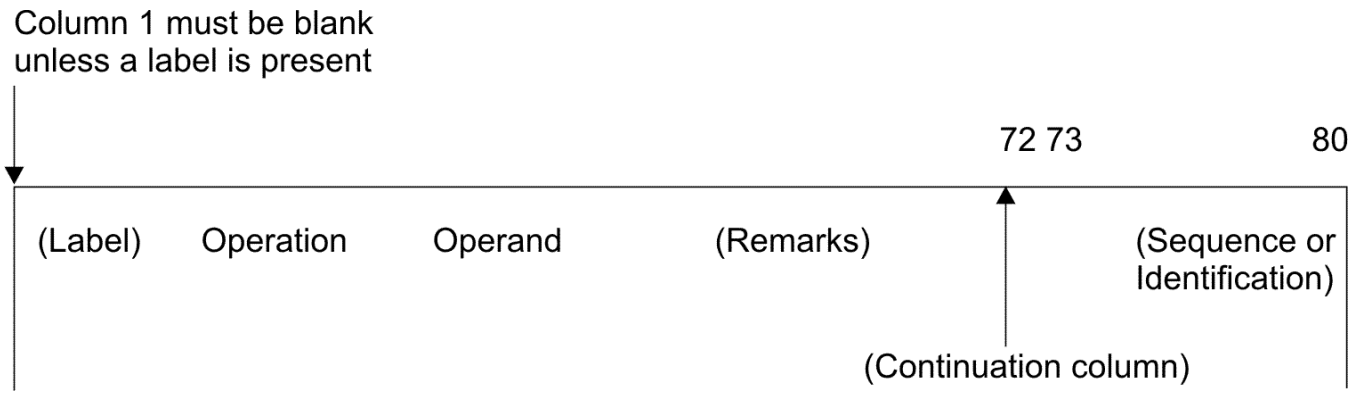


Figure 4. Control Statement Format

The control statements are free-form; that is, the operation definer, operands, and comment field can appear anywhere in a statement, provided they appear in the proper order and are separated by one or more blank characters. Column 1 of each control statement must be blank, unless the first field is a label or a comment statement (see "Inserting comment statements" on page 83).

- **Label Field**

  A label can be specified on any control statement in SYSIN or SORTCNTL. A label is never required. If present, a label must begin in column 1 with any character other than a blank or asterisk (*). A label can be 1 to 70 characters and ends when a blank character is found. Any character can be used in a label. A label followed only by blanks is printed but otherwise not processed.

  Labels cannot be specified in the parameter list, in DFSPARM or in continuation lines.

  To skip the label, specify one or more blanks starting in column 1.

  The following illustrates the use of control statements with and without labels:

```
    OPTION EQUALS
MYSORT SORT FIELDS=(5,4,CH,A)
  OUTREC FIELDS=(1,20,51,30)
OUT_1  OUTFIL FNAMES=OUT1,INCLUDE=(5,1,CH,EQ,C'A')
OUT_2  OUTFIL FNAMES=OUT2,INCLUDE=(5,1,CH,EQ,C'B')
```

- **Operation Field**

  This field can appear anywhere between column 2 and column 71 of the first line. It contains a word (for example, SORT or MERGE) that identifies the statement type to the program. In the example shown later in this section, the operation definer, SORT, is in the operation field of the sample control statement.

- **Operand Field**

  The operand field is composed of one or more operands separated by commas or semicolons. This field must follow the operation field, and be separated from it by at least one blank. No blanks are allowed within the parameters, but a blank is required at the end of all parameters. If the statement occupies more than one line, the operand must begin on the first line. Each operand has an operand definer, or parameter (a group of characters that identifies the operand type to DFSORT). A value or a list of values can be associated with a parameter. The five possible operand forms shown in this chapter are:

  – parameter. Examples: CHALT, NOCHALT, REMOVECC
  – parameter(c). Examples: DATE1(/), TIME2(:), Y2W(-)
  – parameter=value. An operand shown in the form parameter=value can also be specified in the equivalent form parameter(value) or parameter=(value). Examples: AVGRLEN=100, AVGRLEN(100), and AVGRLEN=(100) are equivalent, DYNALLOC=SYSDA, DYNALLOC(SYSDA), and DYNALLOC=(SYSDA) are equivalent, SAMPLE=20, SAMPLE(20), and SAMPLE=(20) are equivalent.

- parameter=(list). An operand shown in the form parameter=(list) can also be specified in the equivalent form parameter(list). Examples: FIELDS=(21,5,CH,A) and FIELDS(21,5,CH,A) are equivalent, BUILD=(1,20,HEX) and BUILD(1,20,HEX) are equivalent.
- parameter=(value). An operand shown in the form parameter=(value) can also be specified in the equivalent form parameter(value). Examples: DATE=(4MD/) and DATE(4MD/) are equivalent, TIMENS=(12) and TIMENS(12) are equivalent, NOMATCH=(C'***') and NOMATCH(C'***') are equivalent.

The following example illustrates the parameter, parameter=value and parameter=(list) forms.

```
SORT EQUALS,FORMAT=CH,FIELDS=(10,30,A),STOPAFT=1000
```

The following example illustrates the parameter, parameter(value), parameter(list) and parameter=(value) forms.

```
SORT EQUALS,FORMAT(CH),FIELDS(10,30,A),STOPAFT=(1000)
```

The two previous SORT statements are equivalent.

- **Remark Field**

  This field can contain any information. It is not required, but if it is present, it must be separated from the last operand field by at least one blank.

- **Continuation Column (72)**

  Any character other than a blank in this column indicates that the present statement is continued on the next line. However, as long as the last character of the operand field on a line is a comma or semicolon or colon followed by a blank, the program assumes that the next line is a continuation line. The nonblank character in column 72 is required only when a remark field is to be continued or when an operand is broken at column 71.

- **Columns 73 through 80**

  This field can be used for any purpose.

Continuation lines

The format of the DFSORT continuation line is shown in Figure 5 on page 81.

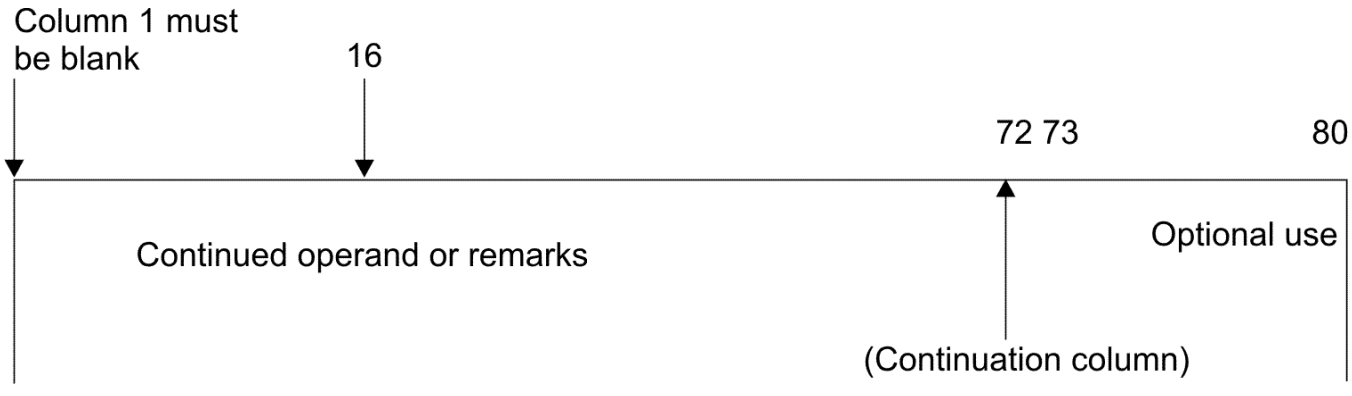


Figure 5. Continuation Line Format

The continuation column and columns 73 through 80 of a continuation line have the same purpose as they do on the first line of a control statement. Column 1 must be blank.

A continuation line is treated as a logical extension of the preceding line. Either an operand or a remark field can begin on one line (referred to as "line 1" in the bullets that follow) and continue on the next line (referred to as "line 2" in the bullets that follow). The following are the rules for continuation illustrated with examples (these different types of continuation can be intermixed):

- **Implicit continuation in 2-71:** If line 1 breaks at a comma-blank or semicolon-blank or colon-blank, DFSORT continues on line 2 with the first nonblank character it finds in columns 2-71. For example:

```
*        1         2         3         4         5         6         7
*234567890123456789012345678901234567890123456789012345678901234567890123456789012
 INCLUDE COND=(5,4,CH,EQ,
  C'ABCD')
 SORT FIELDS=(9,
              3,
              ZD,
              A)
 OUTREC FIELDS=(1,27,2X,       FIRST FIELD AND TWO BLANKS
                51,2,BI,M11,   SECOND FIELD
                60:9,3,ZD,PD)  THIRD FIELD
```

  The previous statements will be treated as if they were specified as:

```
 INCLUDE COND=(5,4,CH,EQ,C'ABCD')
 SORT FIELDS=(9,3,ZD,A)
 OUTREC FIELDS=(1,27,2X,51,2,BI,M11,60:9,3,ZD,PD)
```

- **Explicit continuation in 16:** If line 1 breaks at column 71 with a nonblank in column 72, and columns 2-15 of line 2 are blank, DFSORT continues on line 2 with whatever character it finds in column 16 (blank or nonblank). For example:

```
*        1         2         3         4         5         6         7
*234567890123456789012345678901234567890123456789012345678901234567890123456789012
                                                  INCLUDE COND=(5,4,CH,E*
               Q,C'ABCD')
                                                       SORT FIELDS=(9,3,*
               ZD,A)
                                  OUTREC FIELDS=(1,80,C'BLANK WITHIN A*
                LITERAL')
```

  The previous statements will be treated as if they were specified as:

```
 INCLUDE COND=(5,4,CH,EQ,C'ABCD')
 SORT FIELDS=(9,3,ZD,A)
 OUTREC FIELDS=(1,80,C'BLANK WITHIN A LITERAL')
```

Attention: You should only start with a blank in column 16 of line 2 if you need a blank as the first character of the continued operand, as shown in the previous OUTREC statement. A blank in column 16 of line 2 will be included in the operand and will result in invalid syntax if incorrectly placed. For example:

```
*        1         2         3         4         5         6         7
*234567890123456789012345678901234567890123456789012345678901234567890123456789012
                                                     SORT FIELDS=(5,4,Z*
               D,A)
                                                      SUM FIELDS=(5,4,Z*
                D)
```

  The previous statements will be treated as if they were specified as:

```
 SORT FIELDS=(5,4,ZD,A)
 SUM FIELDS=(5,4,Z D)
```

  With the 'D' in column 16 of line 2, we get 'ZD' in the SORT statement. But with the 'D' in column 17 of line 2, we get 'Z D' in the SUM statement instead of 'ZD', resulting in a syntax error.

- **Explicit continuation in 2-15:** If line 1 breaks at column 71 with a nonblank in column 72, and columns 2-15 of line 2 are nonblank, DFSORT continues on line 2 with the first nonblank character it finds in columns 2-15. For example:

```
*        1         2         3         4         5         6         7
*234567890123456789012345678901234567890123456789012345678901234567890123456789012
                                        INCLUDE COND=(5,4,CH,EQ,C'AB*
   CD')
                                                    SORT FIELDS=(9,3,*
    ZD,A)
                                                  OUTREC FIELDS=(5,4,2X*
      ,9,3,ZD,M26,80:X)
```

  The previous statements will be treated as if they were specified as:

```
 INCLUDE COND=(5,4,CH,EQ,C'ABCD')
 SORT FIELDS=(9,3,ZD,A)
 OUTREC FIELDS=(5,4,2X,9,3,ZD,M26,80:X)
```

- **Remark continuation in 2-71:** If a statement ends on line 1 with a blank before column 72 and a nonblank in column 72, DFSORT treats the first nonblank character it finds in columns 2-71 of line 2 as the start of a remark. For example:

```
*        1         2         3         4         5         6         7
*234567890123456789012345678901234567890123456789012345678901234567890123456789012
   SORT FIELDS=(9,3,ZD,A)                THIS IS A                    *
                                           CONTINUED REMARK
```

  Tip: A simpler way to do the same thing (without continuation) is to use a comment statement for line 2. For example:

```
*        1         2         3         4         5         6         7
*234567890123456789012345678901234567890123456789012345678901234567890123456789012
   SORT FIELDS=(9,3,ZD,A)                THIS IS A
*                                          CONTINUED REMARK
```

Inserting comment statements

- Specify comment statements by coding an asterisk (*) in column 1. A comment statement is printed along with other DFSORT program control statements but is not otherwise processed.
- A statement with blanks in columns 1 through 71 is treated as a comment statement.
- Comment statements are allowed only in the DFSPARM, SYSIN, and SORTCNTL data sets.

Coding restrictions

The following rules apply to control statement preparation:

- Operation definers and operands must be in uppercase EBCDIC.
- Column 1 of each control statement can be used only for a label or for a comment statement that begins with an asterisk in column 1.
- If present, a label must begin in column 1. Labels are allowed only in the SYSIN and SORTCNTL data sets.
- The entire operation definer must be contained on the first line of a control statement.
- The first operand must begin on the first line of a control statement. The last operand in a statement must be followed by at least one blank.
- Blanks are not allowed in operands. Anything following a blank is considered part of the remark field.
- Remarks are allowed only in the DFSPARM, SYSIN, and SORTCNTL data sets.
- Commas, semicolons, and blanks can be used only as delimiters. They can be used in values only if the values are constants.
- Each type of program control statement can appear only once within a single source (for example, the SYSIN data set).

EFS restrictions when an EFS program is in effect

In addition to the previous items, the following restrictions apply to control statement preparation for an EFS program.

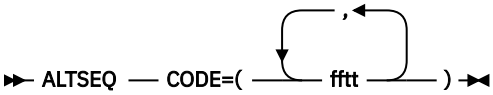
- Non-DFSORT operation definers can be up to 8 bytes long.
- An operation definer with no operands is allowed only if:
  - It is supplied through SYSIN, SORTCNTL, or DFSPARM.
  - It is the only operation definer on a line; column 72 must contain a blank.

Using control statements from other IBM programs

The INPFIL control statement, which is used by other IBM sort programs, is accepted but not processed. However, control statement errors can result from continuation of an INPFIL statement. The information contained in the INPFIL statement for other IBM sort programs is supplied to DFSORT with DD statements.

Because DFSORT uses the OPTION control statement, OPTION control statements in any job streams from other IBM sort programs cause DFSORT to terminate unless the parameters from the other program conform to the DFSORT OPTION control statement parameters.

ALTSEQ control statement



The ALTSEQ control statement can be used to change the alternate translation table (ALTSEQ table). Any modifications you specify are applied to the standard EBCDIC translation table. The modified ALTSEQ table overrides the installation default ALTSEQ table (the shipped default is the EBCDIC translation table).

The ALTSEQ table can be used in two ways as follows:

- To apply an alternate collating sequence for SORT, MERGE, INCLUDE, or OMIT fields, or for INREC, OUTREC, or OUTFIL fields in logical expressions with format AQ (or format CH with CHALT in effect). In this case, the ALTSEQ table is used to change only the order in which data is collated, not the data itself. If you specify AQ (or CH with CHALT) without specifying an ALTSEQ control statement, DFSORT uses the installation default ALTSEQ table.

  For example, if you want to specify that the character $ (X'5B') is to collate at position X'EA', after uppercase Z (X'E9'), you should specify:

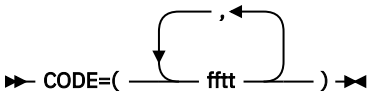
  ```
  ALTSEQ CODE=(5BEA)
  ```

- To convert characters for INREC, OUTREC, or OUTFIL OUTREC fields with TRAN=ALTSEQ. In this case, the ALTSEQ table is used to change the actual data. If you specify TRAN=ALTSEQ without specifying an ALTSEQ control statement, DFSORT uses the installation default ALTSEQ table.

  For example, if you want to change the character $ (X'5B') to the character * (X'5C'), you should specify:

  ```
  ALTSEQ CODE=(5B5C)
  ```

CODE



Specifies the original and modified EBCDIC collating positions.

ff
specifies, in hexadecimal, the character whose position is to be changed in the ALTSEQ table.

tt
specifies, in hexadecimal, the new position the character is to occupy in the ALTSEQ table.

The order in which the parameters are specified is not important.

Note:

1. If CHALT is in effect, control fields with format CH are collated using the ALTSEQ table, in addition to those with format AQ.
2. If you use locale processing for SORT, MERGE, INCLUDE, or OMIT fields, you must not use CHALT. If you need alternate sequence processing for a particular field, use format AQ.
3. Using ALTSEQ can degrade performance.

Default: Usually the installation option. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

Altering EBCDIC collating sequence—examples

Example 1

```
SORT FIELDS=(18,20,AQ,A)
ALTSEQ CODE=(5BEA)
```

The character $ (X'5B') is to collate at position X' EA', that is, after uppercase Z (X'E9').

Example 2

```
MERGE FIELDS=(25,7,A,1,10,D),FORMAT=CH
OPTION CHALT
ALTSEQ CODE=(F0B0,F1B1,F2B2,F3B3,F4B4,F5B5,F6B6,
     F7B7,F8B8,F9B9)
```

The numerals 0 through 9 are to collate before uppercase letters (but after lowercase letters).

Example 3

```
SORT FIELDS=(55,8,AQ,A)
ALTSEQ CODE=(C1F1,C2F2)
```

The uppercase A (X'C1') is to collate at the **same** position as the numeral 1 (X'F1') and the uppercase B (X'C2') is to collate at the **same** position as the numeral 2 (X'F2').

Note that this ALTSEQ statement does NOT cause collating of A before or after 1, or of B before or after 2.

Example 4

```
SORT FIELDS=(55,8,AQ,A)
ALTSEQ   CODE=(81C1,82C2,83C3,84C4,85C5,86C6,87C7,
         88C8,89C9,91D1,92D2,93D3,94D4,95D5,96D6,
         97D7,98D8,99D9,A2E2,A3E3,A4E4,A5E5,A6E6,
         A7E7,A8E8,A9E9)
```

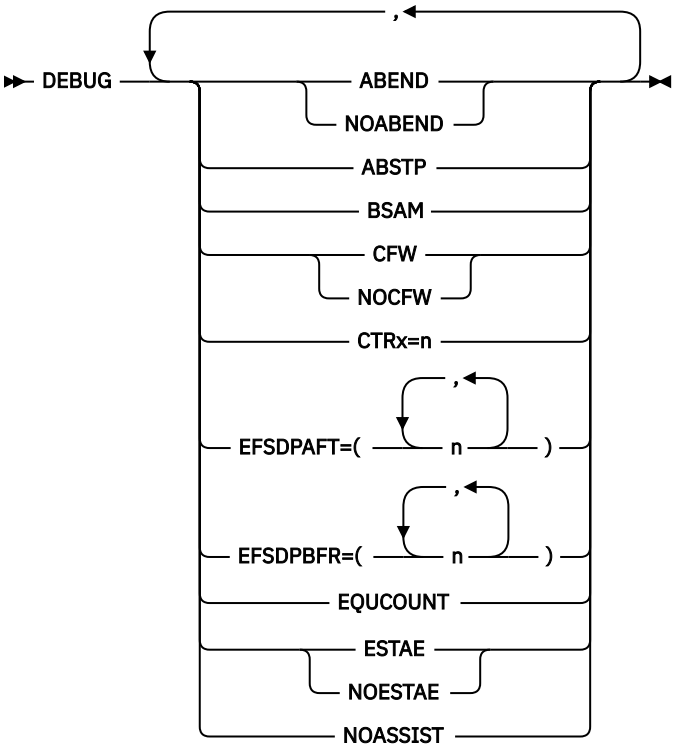
Each lowercase letter is to collate at the **same** position as the corresponding uppercase letter. For example, the lowercase a (X'81') is to collate at the same position as the uppercase A (X'C1'). This results in case-insensitive collating.

Example 5

```
OPTION COPY
ALTSEQ CODE=(0040)
OUTREC FIELDS=(1,80,TRAN=ALTSEQ)
```

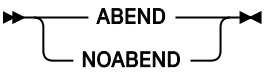
Each binary zero (X'00') is changed to a space (X'40').

DEBUG control statement



The DEBUG control statement is not intended for regular use; only ABEND, NOABEND, and BSAM are of general interest. For a tape work sort or a Conventional merge, only the ABEND or NOABEND parameters of the DEBUG statement are used. For more information about problem diagnosis, see *z/OS DFSORT Messages, Codes and Diagnosis Guide*.

ABEND or NOABEND



Temporarily overrides the ERET installation option, which specifies whether DFSORT abends or terminates with a return code of 16, if your sort, copy, or merge is unsuccessful.

ABEND
Specifies that if your sort, copy, or merge is unsuccessful, DFSORT abends with a user completion code equal to the appropriate message number or with a user-defined number between 1 and 99, as set during installation with installation option ABCODE=n.

When DEBUG ABEND is in effect, a user abend code of zero might be issued when a tape work data set sort or Conventional merge is unsuccessful.

NOABEND
Specifies that an unsuccessful sort, copy, or merge terminates with a return code of 16.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

ABSTP

Prevents loss of needed information in a dump when Blockset terminates. This option overrides ERET, ABEND, and NOABEND. If the DFSORT application is unsuccessful, an abend is forced with a completion code equal to the appropriate message number, or with the user ABEND code set with installation option ABCODE=MSG or ABCODE=n. The message is not written if NOESTAE is in effect.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

BSAM

Temporarily bypasses the EXCP access method for input and output data sets. BSAM is ignored for VSAM input and output data sets.

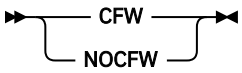
Attention: If Blockset is not selected and BSAM processing is used with concatenated SORTIN input, and both null and non-null data sets are specified, all null data sets must precede all non-null data sets; otherwise, the results are unpredictable.

Attention: This option can degrade performance.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

CFW or NOCFW



Temporarily overrides the CFW installation option, which specifies whether DFSORT can use cache fast write when processing SORTWKdd data sets that reside on devices connected to cached 3990 control units.

CFW
Specifies that DFSORT can use cache fast write when processing SORTWKdd data sets.

NOCFW
Specifies that DFSORT cannot use cache fast write.

Attention: The NOCFW option can degrade performance.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

CTRx

Keeps a count of the input and output records, and abends with code 0C1 when the count reaches n. The numbers that can be assigned to x are:

2
Counts the input records being moved from the input buffer (not used for a copy).

3
Counts the output records being moved to the output buffer (not used for a copy or merge).

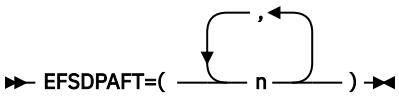
4
Counts the input records inserted by E15 (not used for Blockset).

5
Counts the output records deleted by E35 (not used for Blockset).

Default: None; optional. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

EFSDPAFT



Initiates a SNAP dump after a Major Call to an EFS program. Any combination of the numbers can be specified.

The numbers have the following meanings:

2
Takes the SNAP dump after Major Call 2 to the EFS program.

3
Takes the SNAP dump after Major Call 3 to the EFS program.

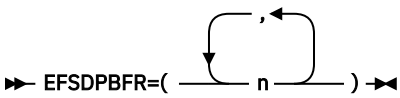
4
Takes the SNAP dump after Major Call 4 to the EFS program.

5
Takes the SNAP dump after Major Call 5 to the EFS program.

Default: None; optional. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

EFSDPBFR



Initiates a SNAP dump before a Major Call to an EFS program. Any combination of the numbers can be specified.

The numbers have the following meanings:

2
Takes the SNAP dump before Major Call 2 to the EFS program.

3
Takes the SNAP dump before Major Call 3 to the EFS program.

4
Takes the SNAP dump before Major Call 4 to the EFS program.

5
Takes the SNAP dump before Major Call 5 to the EFS program.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

EQUCOUNT

Determines the number of records having equal keys (that is, duplicate keys) which have been sorted by the Blockset technique (printed in message ICE184I). For variable-length records, EQUCOUNT can only be used with either Hiperspace (when Hipersorting is used) or work data sets.

Note:

1. Using EQUCOUNT can degrade performance.
2. ICETOOL's UNIQUE and OCCUR operators provide unique and non-unique key reporting capabilities that may be more useful for your application than EQUCOUNT.
3. If VLSHRT is in effect, EQUCOUNT will not be used.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

ESTAE or NOESTAE

Temporarily overrides the ESTAE installation option, which determines whether DFSORT should delete its ESTAE recovery routine early or use it for the entire run.

DFSORT normally establishes an ESTAE recovery routine at the beginning of a run. If an abend occurs and the ESTAE option is in effect, the system passes control to the recovery routine. The routine terminates the run after attempting to:

- Print additional abend information
- Continue a sort, merge, or copy application after successful SORTOUT output
- Call the EFS program at Major Calls 4 and 5 for cleanup and housekeeping
- Write an SMF record
- Call the ICETEXIT termination exit.

If an abend occurs and the ESTAE option is not in effect, these functions might not be performed.

ESTAE
specifies that DFSORT can use its ESTAE recovery routine for the entire run.

NOESTAE
specifies that DFSORT is to delete its ESTAE recovery routine at a point early in its processing. If DFSORT terminates or abends before this point is reached, it will not delete its ESTAE recovery routine; that is, NOESTAE will not be in effect.

Note: See Appendix E, “DFSORT abend processing,” on page 847 for more information on the DFSORT ESTAE recovery routine.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

NOASSIST

DFSORT uses Sorting Instructions when possible. If you do not want to use these instructions, you can temporarily bypass them by specifying this parameter.

Attention: This option can degrade performance.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

Specifying diagnostic options—examples

Example 1

```
SORT FIELDS=(1,4,CH,A)
DEBUG EQUCOUNT
```

If the input records contain the following keys:

- KEYA, KEYA, KEYB, KEYB, KEYC, KEYD, KEYD, KEYE

the following message will be issued:

- ICE184I THE NUMBER OF RECORDS SORTED WITH EQUAL KEYS IS 3

The three equal keys are KEYA, KEYB, and KEYD.

Note: ICETOOL's UNIQUE and OCCUR operators provide full equal key reporting capabilities and should be used instead of EQUCOUNT.

Example 2

```
SORT FIELDS=(12,2,BI,D)
DEBUG BSAM,ABEND
```

Directs DFSORT to use the BSAM access method for the SORTIN and SORTOUT data sets and to abend if the sort application is unsuccessful.

END control statement

The END control statement allows DFSORT to discontinue reading SYSIN, DFSPARM, or SORTCNTL before end of file (EOF).

When you link-edit user exit routines dynamically, the END statement marks the end of the DFSORT control statements and the beginning of exit routine object decks in SYSIN.

Discontinue reading control statements—examples

Example 1

```
//SYSIN DD *
  SORT FIELDS=(1,6,A,28,5,D),FORMAT=CH
  RECORD TYPE=V,LENGTH=(200,,,,80)
```

```
  END
  OPTION DYNALLOC
```

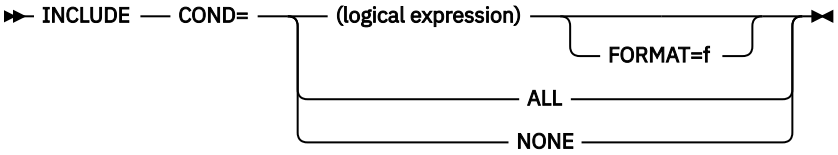
Because the OPTION statement appears after the END statement, it is not read.

Example 2

```
//SYSIN DD *
  SORT FIELDS=(5,8,CH,A)
  MODS E15=(E15,1024,SYSIN,T)
  END
←object deck for E15 user exit here→
```

The END statement precedes the E15 user exit routine object deck in SYSIN.

INCLUDE control statement

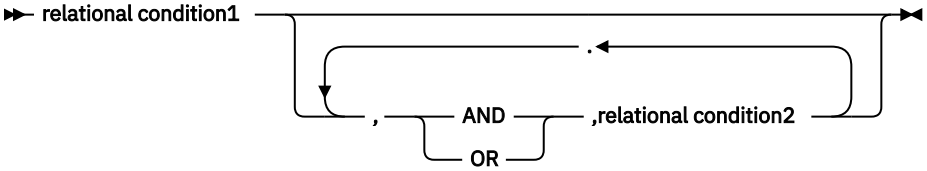


Use an INCLUDE statement if you want only certain records to appear in the output data set. The INCLUDE statement selects the records you want to include.

You can specify either an INCLUDE statement or an OMIT statement in the same DFSORT run, but not both.

The way in which DFSORT processes short INCLUDE/OMIT compare fields depends on the settings for VLSCMP/NOVLSCMP and VLSHRT/NOVLSHRT. A short field is one where the variable-length record is too short to contain the entire field, that is, the field extends beyond the record. For details about including or omitting short records, see the discussion of the VLSCMP and NOVLSCMP options in “OPTION control statement” on page 173.

A logical expression is one or more relational conditions logically combined, based on fields in the input record, and can be represented at a high level as follows:



If the logical expression is true for a given record, the record is included in the output data set.

Five types of relational conditions can be used as follows:

1. **Comparisons:**

   Compare two compare fields or a compare field and a decimal, hexadecimal, character, or current, future, or past date constant.

   For example, you can compare the first 6 bytes of each record with its last 6 bytes, and include only those records in which those fields are identical. Or you can compare a date field with today's date, yesterday's date, or tomorrow's date, and include records accordingly.

   See “Comparisons” on page 94 for information about comparisons.

2. **Substring Comparison Tests:**

   Search for a constant within a field value or a field value within a constant.

For example, you can search the value in a 6-byte field for the character constant C'OK', and include only those records for which C'OK' is found somewhere in the field. Or you can search the character constant C'J69,L92,J82' for the value in a 3-byte field, and include only those records for which C'J69', C'L92', or C'J82' appears in the field.

See "Substring comparison tests" on page 105 for information about substring comparison tests.

3. **Bit Logic Tests:**

   Test the state (on or off) of selected bits in a binary field using a bit or hexadecimal mask or a bit constant.

   For example, you can include only those records which have bits 0 and 2 on in a 1-byte field. Or you can include only those records which have bits 3 and 12 on and bits 6 and 8 off in a 2-byte field.

   See "Bit logic tests" on page 111 for information about bit logic tests.

4. **Date Comparisons:**

   Compare a two-digit year date field to a two-digit year date constant, a current, future or past two-digit year date, or to another two-digit year date field, using the century window in effect.

   For example, you can include only those records for which a Z'yymm' date field is between January 1996 and March 2005. Or you can include only those records for which a P'dddyy' field is less than another P'dddyy' field. Or you can include only those records for which a C'yyddd' field is between today's date and 5 days earlier than today's date.

   See "Date comparisons" on page 117 for information about date comparisons.

5. **Numeric Tests:**

   Test a field for numerics or non-numerics in character, zoned decimal or packed decimal format.

   For example, you can include only those records in which a 5-byte field contains only '0'-'9' characters (that is, numerics). Or you can include only those records in which a 9-byte field contains invalid ZD numeric data (that is, non-numerics). Or you can include only those records in which a 12-byte field contains valid PD numeric data (that is, numerics).

   See "Numeric tests" on page 120 for information about numeric tests.

6. **Alphanumeric Tests:**

   Test a field for alphanumerics or non-alphanumerics in character format. Various combinations of uppercase characters (A-Z), lowercase characters (a-z) and numeric characters (0-9) can be used.

   For example, you can include only those records in which a 10-byte field contains only 'A'-'Z' characters (that is, uppercase characters) or '0'-'9' characters (that is, numeric characters). Or you can include only those records in which a 20-byte field contains characters other than 'a'-'z' (that is, lowercase characters).

   See "Alphanumeric tests" on page 121 for information about alphanumeric tests.

7. **Unicode Comparisons:**

   Compare a field in Unicode format to another field in Unicode format. For example, you can include only those records for which a UTF8 field is less than another UTF8 field. Or you can exclude those records for which a UTF16 field is equal to another UTF16 field. Or you can include only those records for which a UTF32 field is greater than another UTF32 field.

   See "Unicode comparisons" on page 123 for information about Unicode comparisons.

By nesting relational conditions within parentheses, you can create logical expressions of higher complexity.

Although comparisons, substring comparison tests, bit logic tests, date comparisons, numeric tests, and alphanumeric tests are explained separately later in this section for clarity, they can be combined to form logical expressions.

The INCLUDE control statement differs from the INCLUDE parameter of the OUTFIL statement in the following ways:

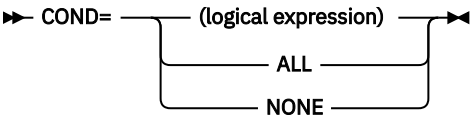
- The INCLUDE statement applies to all input records; the INCLUDE parameter applies only to the OUTFIL input records for its OUTFIL group.
- FORMAT=f can be specified with the INCLUDE statement but not with the INCLUDE parameter. Thus, you can use FORMAT=f and p,m or p,m,f fields with the INCLUDE statement, but you must only use p,m,f fields with the INCLUDE parameter. For example:

```
INCLUDE FORMAT=BI,
   COND=(5,4,LT,11,4,OR,21,4,EQ,31,4,OR,
      61,20,SS,EQ,C'FLY')

OUTFIL INCLUDE=(5,4,BI,LT,11,4,BI,OR,21,4,BI,EQ,31,4,BI,OR,
      61,20,SS,EQ,C'FLY')
```

- D2 format can be specified with the INCLUDE statement but not with the INCLUDE parameter.

See "OUTFIL control statements" on page 221 for more details on the OUTFIL INCLUDE parameter.

COND



logical expression
specifies one or more relational conditions logically combined, based on fields in the input record. If the logical expression is true for a given record, the record is included in the output data sets.

ALL
specifies that all of the input records are to be included in the output data sets.

NONE
specifies that none of the input records are to be included in the output data sets.

Default: ALL. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

FORMAT

FORMAT=f can be used to specify a particular format for one or more compare fields. f from FORMAT=f is used for p,m fields. f from FORMAT=f is ignored for p,m,f fields. For example, the following are all equivalent:

```
INCLUDE COND=(5,5,ZD,EQ,12,3,PD,OR,21,3,PD,NE,35,5,ZD)
INCLUDE FORMAT=ZD,COND=(5,5,EQ,12,3,PD,OR,21,3,PD,NE,35,5)
INCLUDE COND=(5,5,ZD,EQ,12,3,OR,21,3,NE,35,5,ZD),FORMAT=PD
```

The permissible field formats for comparisons are shown in Table 10 on page 95. SS (substring) is the only permissible field format for substring comparison tests. BI (unsigned binary) is the only permissible field format for bit logic tests. The Y2x formats are the only permissible field formats for date comparisons. The FS or CSF (floating sign), ZD (zoned decimal), and PD (packed decimal) formats are the only permissible field formats for numeric tests. BI (unsigned binary) is the only permissible format for alphanumeric tests.

Default: None. FORMAT=f must be specified if any field is specified as p,m rather than p,m,f. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

Note: DFSORT issues an informational message and ignores FORMAT=f if all of the fields are specified as p,m,f.

Relational condition

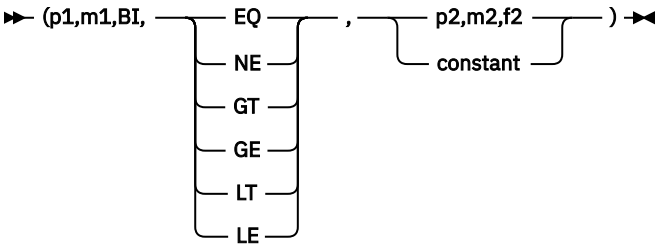
A relational condition specifies a comparison, substring comparison test, bit logic test, date comparison, numeric test or alphanumeric test to be performed. Relational conditions can be logically combined, with AND or OR, to form a logical expression. If they are combined, the following rules apply:

- AND statements are evaluated before OR statements unless parentheses are used to change the order of evaluation; expressions inside parentheses are always evaluated first. (Nesting of parentheses is limited only by the amount of storage available.)
- The symbols & (AND) and | (OR) can be used instead of the words.

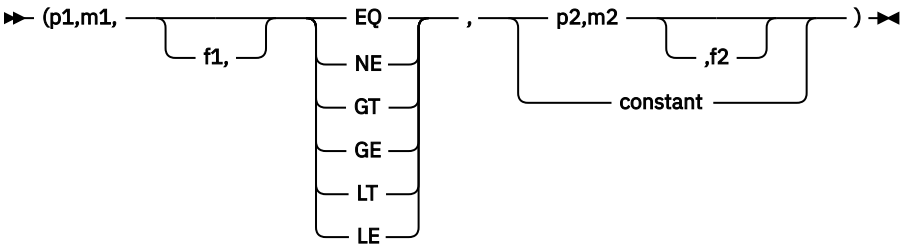
Comparisons

Relational condition format

Two formats for the relational condition can be used:



Or, if the FORMAT=f operand is used:



Comparison operators are as follows:

EQ
Equal to

NE
Not equal to

GT
Greater than

GE
Greater than or equal to

LT
Less than

LE
Less than or equal to.

Fields

p1,m1,f1

These variables specify a field in the input record to be compared either to another field in the input record or to a constant.

- p1 specifies the first byte of the compare field relative to the beginning of the input record.[4] The first data byte of a fixed-length record (FLR) has relative position 1. The first data byte of a variable-length (VLR) record has relative position 5 (because the first 4 bytes contain the record descriptor word). All compare fields must start on a byte boundary, and no compare field can extend beyond byte 32752.
- m1 specifies the length of the compare field. Acceptable lengths for different formats are in Table 10 on page 95.
- f1 specifies the format of the data in the compare field. Permissible formats are given in Table 10 on page 95.

  You can use p1,m1 rather than p1,m1,f1 if you use FORMAT=f to supply the format for the field.

Table 10. Compare Field Formats and Lengths

Format Code	Length	Description
CH	1 to 256 bytes	Character [5]
AQ	1 to 256 bytes	Character with alternate collating sequence
ZD	1 to 256 bytes	Signed zoned decimal
PD	1 to 255 bytes	Signed packed decimal
PD0	2 to 8 bytes	Packed decimal with sign and first digit ignored
FI	1 to 256 bytes	Signed fixed-point
BI	1 to 256 bytes	Unsigned binary
AC	1 to 256 bytes	ASCII character
CSF or FS	1 to 32 bytes	Signed numeric with optional leading floating sign
UFF	1 to 44 bytes	Unsigned free form numeric
SFF	1 to 44 bytes	Signed free form numeric
CSL or LS	2 to 256 bytes	Signed numeric with leading separate sign
CST or TS	2 to 256 bytes	Signed numeric with trailing separate sign
CLO or OL	1 to 256 bytes	Signed numeric with leading overpunch sign
CTO or OT	1 to 256 bytes	Signed numeric with trailing overpunch sign
ASL	2 to 256 bytes	Signed ASCII numeric with leading separate sign

[4] If your E15 user exit routine formats the record, p1 must refer to the record as reformatted by the exit.

Table 10. Compare Field Formats and Lengths (continued)

Format Code	Length	Description
AST	2 to 256 bytes	Signed ASCII numeric with trailing separate sign
D2	1 to 256 bytes	User-defined data type (requires an EFS program)
AUF	1 to 44 bytes	Unsigned ASCII free form numeric
ASF	1 to 44 bytes	Signed ASCII free form numeric

Note: See Appendix C, "Data format descriptions," on page 823 for detailed format descriptions.

p2,m2,f2

These variables specify another field in the input record with which the p1,m1,f1 field will be compared. Permissible comparisons between compare fields with different formats are shown in Table 11 on page 96 and Table 12 on page 96.

AC, ASL, and AST formats sequence EBCDIC data using the ASCII collating sequence.

You can use p2,m2 rather than p2,m2,f2 if you use FORMAT=f to supply the format for the field.

Table 11. Permissable Field-to-Field Comparisons for INCLUDE/OMIT (Group 1)

Field Format	BI	CH	ZD	PD	FI	CSF or FS	UFF	SFF	CSL or LS	CST or TS	AUF	ASF
BI	X	X										
CH	X	X										
ZD			X	X								
PD			X	X								
FI					X							
CSF or FS						X	X	X	X	X	X	X
UFF						X	X	X	X	X	X	X
SFF						X	X	X	X	X	X	X
CSL or LS						X	X	X	X	X	X	X
CST or TS						X	X	X	X	X	X	X
AUF						X	X	X	X	X	X	X
ASF						X	X	X	X	X	X	X

Table 12. Permissable Field-to-Field Comparisons for INCLUDE/OMIT (Group 2)

Field Format	PD0	AC	ASL	AST	CLO or OL	CTO or OT	AQ	D2
PD0	X							
AC		X						
ASL			X	X				

[5] If CHALT is in effect, CH is treated as AQ.

Table 12. Permissable Field-to-Field Comparisons for INCLUDE/OMIT (Group 2) (continued)

Field Format	PD0	AC	ASL	AST	CLO or OL	CTO or OT	AQ	D2
AST			X	X				
CLO or OL					X	X		
CTO or OT					X	X		
AQ							X	
D2								X

Note: D2 field formats are user-defined.

Constants

A constant can be a decimal number (n, +n, -n), character string (C'xx...x'), or hexadecimal string (X'yy...yy').

The **current date** can be used as a decimal number (DATE1P, &DATE1P, DATE2P, &DATE2P, DATE3P, &DATE3P) or character string (DATE1, &DATE1, DATE1(c), &DATE1(c), DATE2, &DATE2, DATE2(c), &DATE2(c), DATE3, &DATE3, DATE3(c), &DATE3(c), DATE4, &DATE4, DATE5, &DATE5).

A **future date** can be used as a decimal number (DATE1P+d, &DATE1P+d, DATE2P+m, &DATE2P+m, DATE3P+d, &DATE3P+d) or character string (DATE1+d, &DATE1+d, DATE1(c)+d, &DATE1(c)+d, DATE2+m, &DATE2+m, DATE2(c)+m, &DATE2(c)+m, DATE3+d, &DATE3+d, DATE3(c)+d, &DATE3(c)+d).

A **past date** can be used as a decimal number (DATE1P-d, &DATE1P-d, DATE2P-m, &DATE2P-m, DATE3P-d, &DATE3P-d) or character string (DATE1-d, &DATE1-d, DATE1(c)-d, &DATE1(c)-d, DATE2-m, &DATE2-m, DATE2(c)-m, &DATE2(c)-m, DATE3-d, &DATE3-d, DATE3(c)-d, &DATE3(c)-d).

The different constants are explained in detail in the following table. Permissible comparisons between compare fields and constants are shown in Table 13 on page 97.

Table 13. Permissible Field-to-Constant Comparisons for INCLUDE/OMIT

Permissible Field-to-Constant Comparisons for INCLUDE/OMITField Format	Self-Defining Term		
	Decimal Number	Character String	Hexadecimal String
BI	X	X	X
CH		X	X
ZD	X		
PD	X		
PD0			X
FI	X		
AC		X	X
ASL	X		
AST	X		
CSF or FS	X		
UFF	X		
SFF	X		
CSL or LS	X		
CST or TS	X		

Table 13. Permissible Field-to-Constant Comparisons for INCLUDE/OMIT (continued)

Permissible Field-to-Constant Comparisons for INCLUDE/OMITField Format	Self-Defining Term		
	Decimal Number	Character String	Hexadecimal String
CLO or OL	X		
CTO or OT	X		
AQ		X	X
D2	X	X	X

Note: D2 field formats are user-defined.

Decimal number format

The format for coding a decimal constant is:

[±]**n**

When an FI field is compared with a decimal constant, n or +n cannot be larger than +9223372036854775807 and -n cannot be smaller than -9223372036854775808.

When a BI field is compared with a decimal constant, n or +n cannot be larger than +18446744073709551615 nor smaller than +0. A BI field cannot be compared to a negative number (-n). A BI field cannot be compared to -0 even if NOSZERO is in effect.

Examples of valid and invalid decimal constants are:

Table 14. Valid and Invalid Decimal Constants

Valid	Invalid	Explanation
15	++15	Too many sign characters
+15	15+	Sign in wrong place
-15	1.5	Contains invalid character
18000000	1,500	Contains invalid character

Current date as decimal number

DATE1P, &DATE1P, DATE2P, &DATE2P, DATE3P, or &DATE3P can be used to generate a decimal number for the current date of the run.

Table 15 on page 98 shows the form of the decimal number constant generated for each current date operand along with an example of the actual decimal number generated when the date of the run is June 21, 2005. yyyy represents the year, mm represents the month (01-12), dd represents the day (01-31) and ddd represents the day of the year (001-366).

Table 15. Decimal Numbers for Current Date

Decimal Numbers for Current DateFormat of Operand	Format of Constant	Example of Constant
DATE1P	+yyyymmdd	+20050621
DATE2P	+yyyymm	+200506
DATE3P	+yyyyddd	+2005172

Note: You can precede each of the operands in the table with an & with identical results.

Future date as decimal number

DATE1P+d, &DATE1P+d, DATE2P+m, &DATE2P+m, DATE3P+d, or &DATE3P+d can be used to generate a decimal number for a future date relative to the current date of the run. d is days in the future and m is months in the future. d and m can be 0 to 9999.

Table 16 on page 99 shows the form of the decimal number constant generated for each future date operand along with an example of the actual decimal number generated when the date of the run is June 21, 2005. yyyy represents the year, mm represents the month (01-12) , dd represents the day (01-31) and ddd represents the day of the year (001-366).

Table 16. Decimal Numbers for Future Dates

Decimal Numbers for Future DatesFormat of Operand	**Format of Constant**	**Example of Operand**	**Example of Constant**
DATE1P+d	+yyyymmdd	DATE1P+11	+20050702
DATE2P+m	+yyyymm	DATE2P+2	+200508
DATE3P+d	+yyyyddd	DATE3P+200	+2006007

Note: You can precede each of the operands in the table with an & with identical results.

Past date as decimal number

DATE1P-d, &DATE1P-d, DATE2P-m, &DATE2P-m, DATE3P-d, or &DATE3P-d can be used to generate a decimal number for a past date relative to the current date of the run. d is days in the past and m is months in the past. d and m can be 0 to 9999.

Table 17 on page 99 shows the form of the decimal number constant generated for each past date operand along with an example of the actual decimal number generated when the date of the run is June 21, 2005. yyyy represents the year, mm represents the month (01-12), dd represents the day (01-31) and ddd represents the day of the year (001-366).

Table 17. Decimal Numbers for Past Dates

Decimal Numbers for Past DatesFormat of Operand	**Format of Constant**	**Example of Operand**	**Example of Constant**
DATE1P-d	+yyyymmdd	DATE1P-30	+20050522
DATE2P-m	+yyyymm	DATE2P-12	+200406
DATE3P-d	+yyyyddd	DATE3P-172	+2004366

Note: You can precede each of the operands in the table with an & with identical results.

Character string format

The format for coding a character string constant is:

```
C'xx...x'
```

The value x may be any EBCDIC character (the EBCDIC character string is translated appropriately for comparison to an AC or AQ field). You can specify up to 256 characters.

If you want to include a single apostrophe in the character string, you must specify it as two single apostrophes. Thus:

```
Required:  O'NEILL  Specify: C'O''NEILL'
```

Examples of valid and invalid character string constants are shown in Table 18 on page 100:

Table 18. Valid and Invalid Character String Constants

Valid	Invalid	Explanation
C'JDCO'	C'''''	Apostrophes not paired
C'$@#'	'ABCDEF'	C identifier missing
C'+0.193'	C'ABCDEF	Apostrophe missing
C'Frank''s'	C'Frank's'	Two single apostrophes needed for one

Double-byte data may be used in a character string for INCLUDE/OMIT comparisons. The start of double-byte data is delimited by the shift-out (SO) control character (X'0E'), and the end by the shift-in (SI) control character (X'0F'). SO and SI control characters are part of the character string and must be paired with zero or an even number of intervening bytes. Nested shift codes are not allowed. All characters between SO and SI must be valid double-byte characters. No single-byte meaning is drawn from the double-byte data.

Examples of valid and invalid character string constants containing double-byte characters are shown in Table 19 on page 100 using:

- < to represent SO
- > to represent SI
- Dn to represent a double-byte character

Table 19. Valid and Invalid Strings with Double-Byte Data

Valid and Invalid Strings with Double-Byte DataValid	Invalid	Explanation
C'Q<D1D2>T'	C'Q<R>S'	Single-byte data within SO/SI
C'<D1D2D3>'	C'D1D2D3'	Missing SO/SI; treated as single-byte data
C'Q<D1>R<D2>'	C'Q<D1<D2>>'	Nested SO/SI

Tip: X'0E', X'0F', and X'7D' are treated as the special characters shift-out, shift-in, and single apostrophe in a character string. If you don't want to treat one or more of these characters as special in a particular value, use a hexadecimal string instead of a character string. For example, if you want to treat the binary value 000E0E7D as its decimal equivalent of 921213, use X'000E0E7D'; 0E will not be treated as shift-out and 7D will not be treated as a single apostrophe.

Current date as character string

DATE1, &DATE1, DATE1(c), &DATE1(c), DATE2, &DATE2, DATE2(c), &DATE2(c), DATE3, &DATE3, DATE3(c), &DATE3(c), DATE4, &DATE4, DATE5 or &DATE5 can be used to generate a character string for the current date of the run.

Table 20 on page 101 shows the form of the character string constant generated for each current date operand along with an example of the actual character string generated when the date of the run is June 21, 2005 at 04:42:45 PM, using (/) for (c) where relevant. yyyy represents the year, mm (for date) represents the month (01-12), dd represents the day (01-31), ddd represents the day of the year (001-366), hh represents the hour (00-23), mm (for time) represents the minutes (00-59), ss represents the seconds (00-59), nnnnnn represents the microseconds (000000-999999) and c can be any character *except* a blank.

Table 20. Character Strings for Current Date

Character Strings for Current DateFormat of Operand	Format of Constant	Example of Constant
DATE1	C'yyyymmdd'	C'20050621'
DATE1(c)	C'yyyycmmcdd'	C'2005/06/21'
DATE2	C'yyyymm'	C'200506'
DATE2(c)	C'yyyycmm'	C'2005/06'
DATE3	C'yyyyddd'	C'2005172'
DATE3(c)	C'yyyycddd'	C'2005/172'
DATE4	C'yyyy-mm-dd-hh.mm.ss'	C'2005-06-21-16.52.45'
DATE4	C'yyyy-mm-dd-hh.mm.ss'	C'2005-06-21-16.52.45'
DATE5	C'yyyy-mm-dd-hh.mm.ss.nnnnnn'	C'2005-06-21-16.52.45.582013'

Note: You can precede each of the operands in the table with an & with identical results.

Tip: When a field is shorter than the character string it's compared to, DFSORT truncates the string on the right. You can take advantage of this to compare a field to only part of the DATE4 timestamp when appropriate. For example:

```
INCLUDE COND=(1,13,CH,GT,DATE4)
```

would compare the field in positions 1-13 to the truncated DATE4 constant C'yyyy-mm-dd-hh'.

Future date as character string

DATE1+d, &DATE1+d, DATE1(c)+d, &DATE1(c)+d, DATE2+m, &DATE2+m, DATE2(c)+m, &DATE2(c)+m, DATE3+d, &DATE3+d, DATE3(c)+d or &DATE3(c)+d can be used to generate a character string for a future date relative to the current date of the run. d is days in the future and m is months in the future. d and m can be 0 to 9999.

Table 21 on page 101 shows the form of the character string constant generated for each future date operand along with an example of the actual character string generated when the date of the run is June 21, 2005. yyyy represents the year, mm represents the month (01-12), dd represents the day (01-31), ddd represents the day of the year (001-366), and c can be any character *except* a blank.

Table 21. Character Strings for Future Dates

Character Strings for Future DatesFormat of Operand	Format of Constant	Example of Operand	Example of Constant
DATE1+d	C'yyyymmdd'	DATE1+11	C'20050702'
DATE1(c)+d	C'yyyycmmcdd'	DATE1(/)+90	C'2005/09/19'
DATE2+m	C'yyyymm'	DATE2+2	C'200508'
DATE2(c)+m	C'yyyycmm'	DATE2(.)+25	C'2007.07'
DATE3+d	C'yyyyddd'	DATE3+200	C'2006007'
DATE3(c)+d	C'yyyycddd'	DATE3(-)+1	C'2005-171'

Note: You can precede each of the operands in the table with an & with identical results.

Past date as character string

DATE1-d, &DATE1-d, DATE1(c)-d, &DATE1(c)-d, DATE2-m, &DATE2-m, DATE2(c)-m, &DATE2(c)-m, DATE3-d, &DATE3-d, DATE3(c)-d or &DATE3(c)-d can be used to generate a character string for a past date relative to the current date of the run. d is days in the past and m is months in the past. d and m can be 0 to 9999.

Table 22 on page 102 shows the form of the character string constant generated for each past date operand along with an example of the actual character string generated when the date of the run is June 21, 2005. yyyy represents the year, mm represents the month (01-12), dd represents the day (01-31), ddd represents the day of the year (001-366), and c can be any character *except* a blank.

Table 22. Character Strings for Past Dates

Character Strings for Past DatesFormat of Operand	**Format of Constant**	**Example of Operand**	**Example of Constant**
DATE1-d	C'yyyymmdd'	DATE1-1	C'20050620'
DATE1(c)-d	C'yyyycmmcdd'	DATE1(-)-60	C'2005-04-22'
DATE2-m	C'yyyymm'	DATE2-6	C'200412'
DATE2(c)-m	C'yyyycmm'	DATE2(/)-1	C'2005/05'
DATE3-d	C'yyyyddd'	DATE3-300	C'2004238'
DATE3(c)-d	C'yyyycddd'	DATE3(.)-21	C'2005.151'

Note: You can precede each of the operands in the table with an & with identical results.

Hexadecimal string format

The format for coding a hexadecimal string constant is:

```
X'yy...yy'
```

The value yy represents any pair of hexadecimal digits. You can specify up to 256 pairs of hexadecimal digits.

Because the first digit and sign are ignored in a PD0 field, you should not include the first digit or sign in a hexadecimal constant to be compared to a PD0 field. For example, 3-byte PD0 values like X'01234C' and X'01234D' would be equal to a hexadecimal constant of X'1234'.

Examples of valid and invalid hexadecimal constants are shown in the following table.

Table 23. Valid and Invalid Hexadecimal Constants

Valid	**Invalid**	**Explanation**
X'ABCD'	X'ABGD'	Invalid hexadecimal digit
X'BF3C'	X'BF3'	Incomplete pair of digits
X'AF050505'	'AF050505'	Missing X identifier
X'BF3C'	'BF3C'X	X identifier in wrong place

Padding and truncation

In a field-to-field comparison, the shorter compare field is padded appropriately. In a field-to-constant comparison, the constant is padded or truncated to the length of the compare field.

Character and hexadecimal strings are truncated and padded on the right.

The padding characters are:

- X'40' For a character string
- X'00' For a hexadecimal string.

Decimal constants are padded and truncated on the left. Padding is done with zeros in the proper format.

Cultural environment considerations

DFSORT's collating behavior can be modified according to your cultural environment. The cultural environment is established by selecting the active locale. The active locale's collating rules affect INCLUDE and OMIT processing as follows:

- DFSORT includes or omits records for output according to the collating rules defined in the active locale. This provides inclusion or omission for single- or multi-byte character data, based on defined collating rules which retain the cultural and local characteristics of a language.

If locale processing is to be used, the active locale will only be used to process character (CH) compare fields and character and hexadecimal constants compared to character (CH) compare fields.

For more information on locale processing, see “Cultural environment considerations ” on page 6 or LOCALE in “OPTION control statement” on page 173.

Including records in the output data set—comparison examples

Example 1

```
INCLUDE  COND=(5,8,GT,13,8,|,105,4,LE,1000),FORMAT=CSF
```

This example illustrates how to only include records in which:

- The floating sign number in bytes 5 through 12 is greater than the floating sign number in bytes 13 through 20

  OR

- The floating sign number in bytes 105 through 108 is less than or equal to 1000.

Note that all three compare fields have the same format.

Example 2

```
INCLUDE   COND=(1,10,CH,EQ,C'STOCKHOLM',
          AND,21,8,ZD,GT,+50000,
          OR,31,4,CH,NE,C'HERR')
```

This example illustrates how to only include records in which:

- The first 10 bytes contain STOCKHOLM (this nine-character string was padded on the right with a blank) AND the zoned-decimal number in bytes 21 through 28 is greater than 50 000

  OR

- Bytes 31 through 34 do not contain HERR.

Note that the AND is evaluated before the OR. (“Omitting records from the output data set—example” on page 172 illustrates how parentheses can be used to change the order of evaluation.) Also note that ending a line with a comma or semicolon followed by a blank indicates that the parameters continue on the next line, starting in any position from columns 2 through 71.

Example 3

```
INCLUDE  FORMAT=CH,
  COND=((5,1,EQ,8,1),&,
        ((20,1,EQ,C'A',&,30,1,FI,GT,10),|,
         (20,1,EQ,C'B',&,30,1,FI,LT,100),|,
         (20,1,NE,C'A',&,20,1,NE,C'B')))
```

This example illustrates how to only include records in which:

- Byte 5 equals byte 8

  AND

- At least one of the following is true:
  - Byte 20 equals 'A' and byte 30 is greater than 10
  - Byte 20 equals 'B' and byte 30 is less than 100
  - Byte 20 is not equal to 'A' or 'B'.

Note that p,m,FI is used for the FI fields, and p,m with FORMAT=CH is used for all of the CH fields. With FORMAT=f, you can mix p,m and p,m,f fields when that's convenient such as when all or most of the fields have the same format (although you can always code p,m,f for all fields and not use FORMAT=f, if you prefer).

Example 4

```
 INCLUDE COND=(7,2,CH,EQ,C'T1',OR,
              (1,2,BI,GE,X'001A',AND,20,2,CH,EQ,25,2,CH))
```

This example shows the effects of VLSCMP/NOVLSCMP and VLSHRT/NOVLSHRT on INCLUDE processing when short records are present.

Consider the records shown in Figure 6 on page 105:

- If VLSCMP is in effect, the first record is included because bytes 7-8 are equal to C'T1', even though the comparison of bytes 20-21 to 25-26 involves short fields. The second record is included or omitted based on the comparison of bytes 20-21 to bytes 25-26.
- If NOVLSCMP and VLSHRT are in effect, the first record is omitted because the comparison of bytes 20-21 to 25-26 involves short fields. The second record is included or omitted based on the comparison of bytes 20-21 to bytes 25-26.
- If NOVLSCMP and NOVLSHRT are in effect, the first record causes message ICE015A or ICE218A to be issued because the comparison of bytes 20-21 to bytes 25-26 involves short fields.

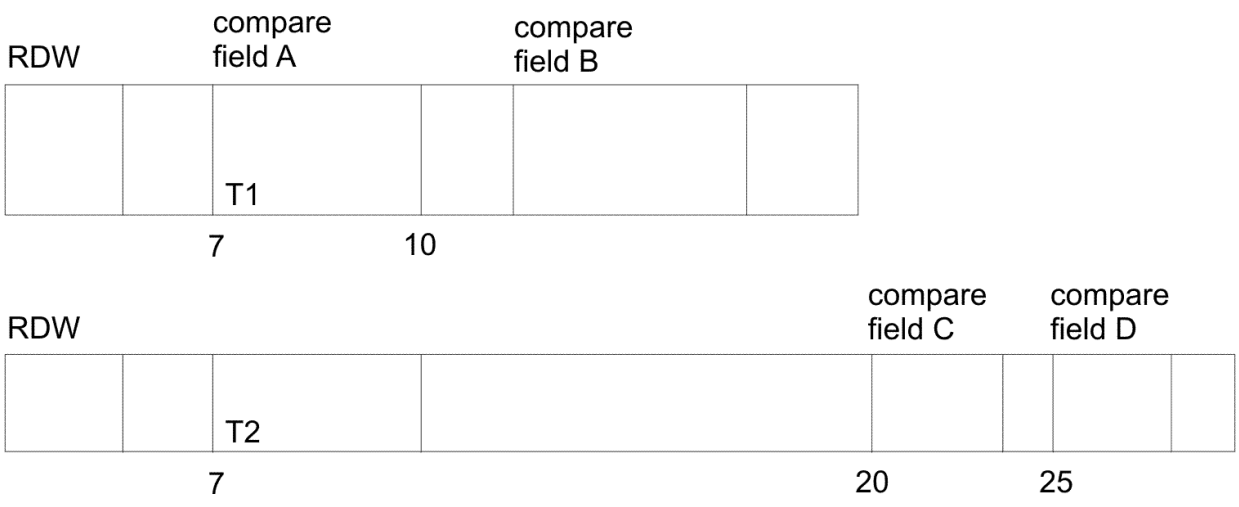


Figure 6. Sample Records

Example 5

```
INCLUDE COND=(21,8,ZD,GT,DATE1P)
```

This example illustrates how to include records in which a zoned-decimal date of the form Z'yyyymmdd' in bytes 21-28 is greater than today's date. DATE1P generates a decimal number for the current date in the form +yyyymmdd.

Example 6

```
INCLUDE COND=(15,7,CH,GE,DATE3-7,AND,15,7,CH,LE,DATE3+7)
```

This example illustrates how to include records in which a character date of the form C'yyyyddd' in bytes 15-21 is between 7 days in the past and 7 days in the future, relative to the current date. DATE3-7 generates a character constant in the form C'yyyyddd' where yyyyddd is the current date minus 7 days. DATE3+7 generates a character constant in the form C'yyyyddd' where yyyyddd is the current date plus 7 days.

Example 7

```
INCLUDE COND=(21,10,CH,GE,DATE1(-)-365)
```

This example illustrates how to include records in which a character date of the form C'yyyy-mm-dd' in bytes 21-30 is within 365 days of the current date. DATE1(-)-365 generates a character constant in the form C'yyyy-mm-dd' where yyyymmdd is the current date minus 365 days.

Substring comparison tests

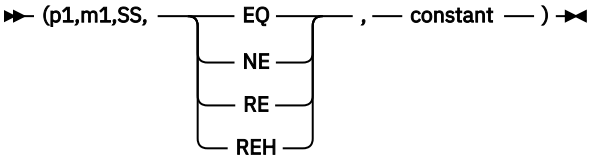
Four types of substring comparison tests are offered, as follows:

1. Find a constant within a field value. For example, you can search the value in a 6-byte field for the character constant C'OK'. If the field value is, for example, C'**OK**' or C'****OK', the relational condition is true; if the field value is C'**ERR*', the relational condition is false.
2. Find a field value within a constant. For example, you can search the character constant C'J69,L92,J82' for the value in a 3-byte field. If the field value is C'J69', C'L92', or C'J82', the relational condition is true; if the field value is C'X24', the relational condition is false. Note that the comma is used within the constant to separate the valid 3-character values; any character that will not appear in the field value can be used as a separator in the constant.

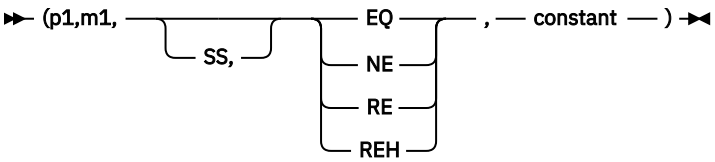
3. Find a field value or a constant within a field value using Regular expressions. For example, you can search the value in a 20 byte field for the character constant C'(iP).*[0-9]'. If the field value for example C'IP' or C'ip' or C'Ip' or C'iP' and a numeric 0-9, the relational condition is true; if the field value is C'It', the relational condition is false.
4. Find a field value or a constant within a field value using Regular expressions with hexadecimal strings. For example, you can search the value in a 20 byte field for the character constant C'(.*Z+)|(.*\XC2\XF1.*)'. If the field value for example C'Fuzzy' or C'AnZ}' or X'C2F1', the relational condition is true; if the field value is C'EAN', the relational condition is false.

Relational condition format

Four formats for the relational condition can be used:



Or, if the FORMAT=SS operand is used:



Restriction: FORMAT=SS can precede COND but cannot follow it.

Substring comparison operators are as follows:

EQ
Equal to

NE
Not equal to

RE
Regular expressions

REH
Regular expressions with hexadecimal string

Fields

p1,m1

These variables specify the character field in the input record for the substring test.

- p1 specifies the first byte of the character input field for the substring test, relative to the beginning of the input record.[6] The first data byte of a fixed-length record (FLR) has relative position 1. The first data byte of a variable-length (VLR) record has relative position 5 (because the first 4 bytes contain the record descriptor word). All fields to be tested must start on a byte boundary and must not extend beyond byte 32752.
- m1 specifies the length of the field to be tested. The length can be 1 to 32752 bytes.

[6] If your E15 user exit routine formats the record, p1 must refer to the record as reformatted by the exit.

Constant

The constant can be a character string or a hexadecimal string. See "Character string format" on page 99 and "Hexadecimal string format" on page 102 for details.

If m1 is greater than the length of the constant, the field value will be searched for the constant and the condition will be true if a match is found when the EQ comparison operator is specified or if a match is not found when the NE comparison operator is specified.

If m1 is smaller than the length of the constant, the constant will be searched for the field value and the condition will be true if a match is found when the EQ comparison operator is specified or if a match is not found when the NE comparison operator is specified.

Regular expressions

A Regular expression is a pattern which specifies a string of characters used to match certain strings. By default, DFSORT treats regular expressions as **case-insensitive**. The input regular expression can be in any case (lowercase, uppercase or mixed case) and output results are also case-insensitive. DFSORT compares the NULL-terminated string specified by INCLUDE/OMIT field against the compiled regular expression. For comparison DFSORT internally adds NULL byte to the specified Regular expression. The specified Regular expression is then compiled and if an error is detected, DFSORT terminates with an error message.

This support is based on the following Standards /Extensions:

- XPG4 (X/Open Common Applications Environment Specification, System Interfaces and Headers, Issue 4.)
- XPG4.2 (X/Open Common Applications Environment Specification, System Interfaces and Headers, Issue 4, Version 2.)
- Single UNIX Specification, Version 3 (IEEE Std 1003.1-2001.)
- z/OS UNIX (functions that provide z/OS UNIX support beyond the defined standards.)

Two versions of regular expressions are supported:

- Basic Regular expressions (BRE)
- Extended Regular expressions (ERE)

Regular expressions can be made up of normal characters or special characters, sometimes called metacharacters. Basic and extended regular expressions differ only in the metacharacters they can contain.

The basic regular expression metacharacters are:

```
¬ $ . * \( \) [ \{ \} \
```

The extended regular expression metacharacters are:

```
| ¬ $ . * + ? ( ) [ { }
```

Table 24. Regular expression metacharacters

Symbol	Description
.	The period symbol matches any one character except the terminal newline character.
[character–character]	The hyphen symbol, within square brackets, means "through." It fills in the intervening characters according to the current collating sequence. For example, [a–z] can be equivalent to [abc...xyz] or [aAbBcC...xXyYzZ].

Table 24. Regular expression metacharacters (continued)	
Symbol	**Description**
[string]	A string within square brackets specifies any of the characters in string. Thus [abc], if compared to other strings, would match any that contained a, b, or c.
{n} {n,} {n,u}	Integer values enclosed in {} indicate the number of times to apply the preceding regular expression. n is the minimum number, and u is the maximum number. If you specify only n, it indicates the exact number of times to apply the regular expression. {n,} is equivalent to {n,u}. They both match n or more occurrences of the expression.
*	The asterisk symbol indicates 0 or more of any characters. For example, [a*e] is equivalent to any of the following: 99ae9, aaaaae, a999e99.
$	The dollar symbol matches the end of the string.
character+	The plus symbol specifies one or more occurrences of a character. Thus, smith+ern is equivalent to, for example, smithhhern.
[^string]	The caret symbol, when inside square brackets, negates the characters within the square brackets. Thus [^abc], if compared to other strings, would fail to match any that contains even one a, b, or c.
(expression)	Groups a sub-expression allowing an operator, such as *, +, or [].], to work on the sub-expression enclosed in parentheses. For example, (a*(cb+)*) $0.

Note:

1. Do not use multibyte characters.
2. You can use the] (right square bracket) alone within a pair of square brackets, but only if it immediately follows either the opening left square bracket or if it immediately follows [^. For example: []–] matches the] and – characters.
3. All the preceding symbols are special. You precede them with \ to use the symbol itself. For example, a\.e is equivalent to a.e.
4. You can use the – (hyphen) by itself, but only if it is the first or last character in the expression. For example, the expression []--0] matches either the] or else the characters – through 0. Otherwise, use \–.

The following patterns are given as examples, along with descriptions of what they match:

abc
Matches any record containing the three letters abc in that order.

a.c
Matches any string beginning with the letter a, followed by any character, followed by the letter c.

^.$
Matches any record containing exactly one character.

.* [a–z]+ .*
Matches any record containing a word, consisting of lowercase/uppercase/mixed case alphabetic characters, delimited by at least one space on each side.

Johny.*Johny
Matches any record containing at least two occurrences of the string Johny.

^ibm
Matches records beginning with "ibm"

ibm$
Matches records ending with "ibm"

^ibm$
Matches records with exactly "ibm"

Regular expressions application notes

Can only be used if Blockset technique is selected.

The following existing DFSORT functions are not compatible when used with Regular expressions:

- Locale
- EFS
- COBOL E15 and E35 user exits with COPY
- COBOL E15 user exit when regular expression is used in INCLUDE/OMIT control statement
- COBOL E35 user exit when regular expression is used in OUTFIL's INCLUDE/OMIT control statement.
- For a JOINKEYS application, no more than one task can use regular expressions. Regular expressions can only be used in the main task or subtask 1 or subtask 2, and not in any combination of tasks.

Regular expressions application examples

Example 1: Include all records that contain string 'Department'(case-insensitive) using Regular expressions:

```
//STEP0100 EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SORTDIAG DD DUMMY
//SORTIN   DD *
lower case              florida department of Palm trees
UPPER CASE                 FLORIDA DEPARTMENT OF ORANGE GROVES
exclude this record     California state
alternate case            FlOrIdA DePaRtMeNt Of PaLm TrEeS
sentence                     Florida Department Of Revenue
Mixed case              ARIZONA DEPARTment of Motor Vehicles
//SORTOUT  DD SYSOUT=*
//SYSIN    DD *
  OPTION COPY
  INCLUDE COND=(25,50,SS,RE,C'DEPARTMENT')
//*
```

Using the string C'department' or C'DePaRtMeNt' will also produce the same exact result.

This example illustrates how to include only records in which:

- DEPARTMENT (case-insensitive) is found somewhere within bytes 25 through 74
- Using the string C'department' or C'DePaRtMeNt' will also produce the same exact results.

The Output from the above job is:

```
lower case              florida department of Palm trees
UPPER CASE                 FLORIDA DEPARTMENT OF ORANGE GROVES
alternate case            FlOrIdA DePaRtMeNt Of PaLm TrEeS
sentence                     Florida Department Of Revenue
Mixed case              ARIZONA DEPARTment of Motor Vehicles
```

Example 2: Omit records which contain one or more occurrences of a character or hex strings using Regular expressions.

```
//STEP0100 EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
```

```
//SORTIN   DD *
record - 1 1234500\x00a ANZ]'' string CD0
record - 2 12345111EAND'45678901234567890
record - 3 123452289012RRAA"890....567890
record - 4 1234533\x00a SIMPLE string CD0
record - 5 12345441EANx000!           B1
record - 6 123456789012345fizZZ
record - 7 12345678901234567890123456789O
/*
//SORTOUT  DD SYSOUT=*
//SYSIN    DD *
  OPTION COPY
  OUTFIL OMIT=(25,032,SS,REH,C'(.*Z+)|(.*\XC2\XF1.*)')
//*
```

This example illustrates how to exclude records in which:

- One or more character ‘z’ (Case-insensitive) is followed by any character OR the records that contain hex values of X’C2F1’ which is equivalent to characters ‘B1’ are found somewhere within bytes 25 through 56.

The output from this job is:

```
record - 2 12345111EAND'45678901234567890
record - 3 123452289012RRAA"890....567890
record - 4 1234533\x00a SIMPLE string CD0
record - 7 12345678901234567890123456789O
```

Example 3: Include all records that contain string 'ER'(case-sensitive) followed by a numeric (0-9) using Regular expressions.

```
//STEP0100 EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SORTIN   DD *
MORGAN STANLEY
 ERIC
      HOLMER CA 90201
  WALTER  ALBUQUERQUENEW MEXICO
TONY NY 10000
 JERRY 10029 NEW YORK
SHERLOCK
//SORTOUT  DD SYSOUT=*
//SYSIN    DD *
  OPTION COPY
  INCLUDE COND=(1,50,SS,RE,C'(ER).*[0-9]')
//*
```

This example illustrates how to include only records in which:

- ER (case-insensitive) is found somewhere within bytes 1 through 50 and have a number (0-9) following the string ER.

The output from this job is:

```
    HOLMER CA 90201
   JERRY 10029 NEW YORK
```

Example 4: Show the usage of Regular expressions with JOINKEYS application.

```
//STEP0100 EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//INA      DD DISP=SHR,DSN=Your.input.file1
//INB      DD DISP=SHR,DSN=Your.input.file2
//SORTOUT  DD SYSOUT=*
//SYSIN    DD *
  OPTION COPY
  JOINKEYS F1=F1,FIELDS=(10,4,A)
  JOINKEYS F2=F2,FIELDS=(10,4,A)
  JOIN UNPAIRED,F1,F2
  REFORMAT FIELDS=(F1:10,4,F2:10,4,F1:3,6,F2:3,6)
/*
//JNF1CNTL DD *
  OMIT COND=(10,4,SS,RE,C'8D[0-9A-F]{2,2}')
```

```
/*
//JNF2CNTL DD *
  OMIT COND=(10,4,CH,EQ,C' ')
/*
```

Note:

Regular expressions can only be used in the main task or subtask 1 or subtask 2, and not in any combination of tasks.

Including records in the output data set—substring comparison example

Example

```
  INCLUDE FORMAT=SS,COND=(11,6000,EQ,C'OK',OR,5,3,EQ,C'J69,L92,J82')
```

This example illustrates how to include only records in which:

- OK is found somewhere within bytes 11 through 6010

  OR

- Bytes 5 through 7 contain J69, L92 or J82.

Bit logic tests

Two methods for bit logic testing are offered as follows:

- Bit operator with hexadecimal or bit mask
- Bit comparison tests

While any bit logic test can be specified using either of the two methods, each of them offers unique advantages not found with the other.

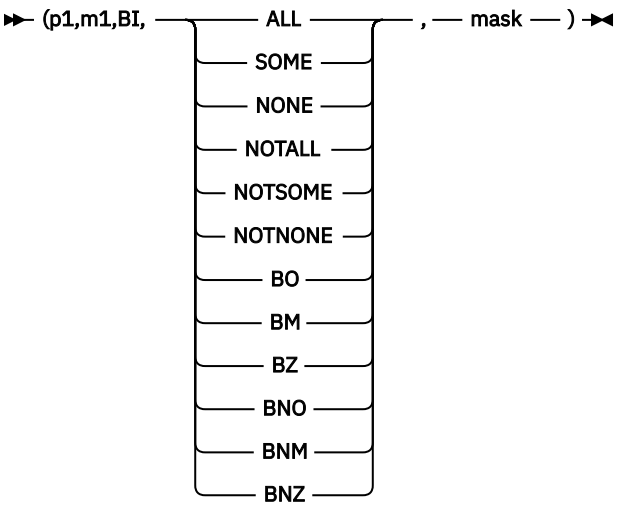
The ability to specify selected bits in a field, by either of the two methods, can greatly reduce the number of INCLUDE conditions that must be specified to achieve a given result, because the need to account for unspecified bits is eliminated.

Method 1: Bit operator tests

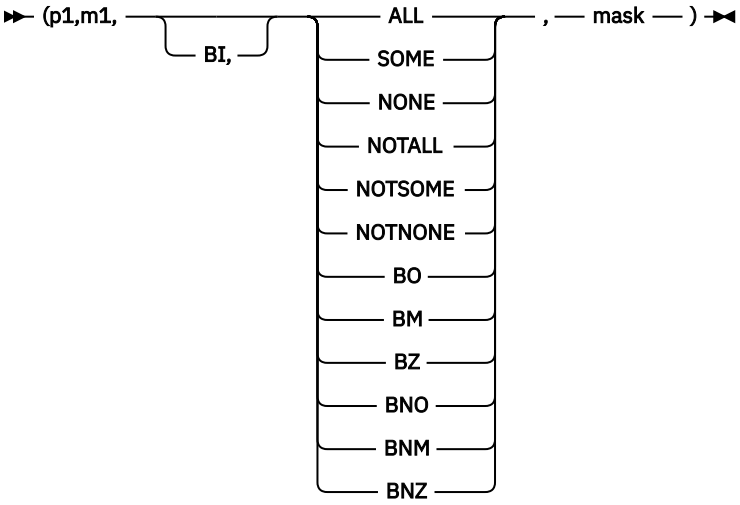
This method of bit logic testing allows you to test whether selected bits in a binary field are all on, all off, in a mixed on-off state, or in selected combinations of these states. While this method allows you to test many different possible bit combinations with a single operation, similar to the Test Under Mask (TM) machine instruction, it is less suited to determine if a field contains exactly one particular combination of on and off bits than Method 2 described later in this section.

Relational condition format

Two formats for the relational condition can be used:



Or, if the FORMAT=BI operand is used:



Bit operators describe the input field to mask relationship to be tested as follows:

ALL or BO
All mask bits are on in the input field

SOME or BM
Some, but not all mask bits are on in the input field

NONE or BZ
No mask bits are on in the input field

NOTALL or BNO
Some or no mask bits are on in the input field

NOTSOME or BNM
All or no mask bits are on in the input field

NOTNONE or BNZ
All or some mask bits are on in the input field

The first set of operators (ALL, SOME, and so on) are intended for those who like meaningful mnemonics. The second set of operators (BO, BM, and so on) are intended for those familiar with the conditions associated with the Test Under Mask (TM) instruction.

Fields

p1,m1

These variables specify the binary field in the input record to be tested against the mask.

- p1 specifies the first byte of the binary input field to be tested against the mask, relative to the beginning of the input record.[7] The first data byte of a fixed-length record (FLR) has relative position 1. The first data byte of a variable-length (VLR) record has relative position 5 (because the first 4 bytes contain the record descriptor word). All fields to be tested must start on a byte boundary and must not extend beyond byte 32752.
- m1 specifies the length of the field to be tested. The length can be 1 to 256 bytes.

Mask

A hexadecimal string or bit string that indicates the bits in the field selected for testing. If a mask bit is on (1), the corresponding bit in the field is tested. If a mask bit is off (0), the corresponding bit in the field is ignored.

Hexadecimal string format

The format for coding a hexadecimal string mask is:

```
X'yy...yy'
```

The value yy represents any pair of hexadecimal digits that constitute a byte (8 bits). Each bit must be 1 (test bit) or 0 (ignore bit). You can specify up to 256 pairs of hexadecimal digits.

Bit string format

The format for coding a bit string mask is:

```
B'bbbbbbbb...bbbbbbbb'
```

The value bbbbbbbb represents 8 bits that constitute a byte. Each bit must be 1 (test bit) or 0 (ignore bit). You can specify up to 256 groups of 8 bits. The total number of bits in the mask must be a multiple of 8. A bit mask string can only be used with a bit operator.

Padding and truncation

The hexadecimal or bit mask is truncated or padded on the right to the byte length of the binary field. The padding character is X'00' (all bits off and thus not tested).

Including records in the output data set—bit operator test examples

Example 1

```
 INCLUDE   COND=(27,1,CH,EQ,C'D',AND,18,1,BI,ALL,B'10000000')
```

This example illustrates how to only include records in which:

- Byte 27 contains D

  AND

- Byte 18 has bit 0 on.

[7] If your E15 user exit routine formats the record, p1 must refer to the record as reformatted by the exit.

Example 2

```
INCLUDE   COND=(11,1,BI,BM,X'85')
```

This example illustrates how to only include records in which byte 11 has some, but not all of bits 0, 5 and 7 on. Results for selected field values are shown as follows:

Table 25. Bit Comparison Example 2: Results for Selected Field Values

Bit Comparison Example 2: Results for Selected Field Values11,1,BI Value	11,1,BI Result	Action
X'85'	False	Omit Record
X'C1'	True	Include Record
X'84'	True	Include Record
X'00'	False	Omit Record

Example 3

```
INCLUDE   COND=(11,2,ALL,B'0001001000110100',
               OR,21,1,NONE,B'01001100'),FORMAT=BI
```

This example illustrates how to only include records in which:

- Bytes 11 through 12 have all of bits 3, 6, 10, 11 and 13 on

  OR

- Byte 21 has none of bits 1, 4, or 5 on.

Results for selected field values are shown as follows:

Table 26. Bit Comparison Example 3: Results for Selected Field Values

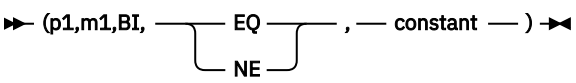
Bit Comparison Example 3: Results for Selected Field Values11,2,BI Value	11,2,BI Result	21,1,BI Value	21,1,BI Result	Action
X'1234'	True	X'4C'	False	Include Record
X'02C4'	False	X'81'	True	Include Record
X'0204'	False	X'40'	False	Omit Record
X'F334'	True	X'00'	True	Include Record
X'1238'	False	X'4F'	False	Omit Record

Method 2: Bit comparison tests

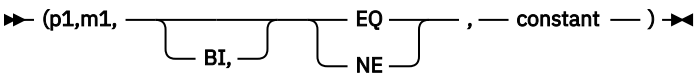
This method of bit logic testing allows you to test whether selected bits in a binary field are either in an exact pattern of on and off bits, or not in that exact pattern. Unlike Method 1 described previously, only "equal" and "unequal" comparisons are allowed; however, this method has the advantage of being able to test for a precise combination of on and off bits.

Relational condition format

Two formats for the relational condition can be used:



Or, if the FORMAT=BI operand is used:



Bit comparison operators are as follows:

EQ
Equal to

NE
Not equal to

Fields

p1,m1

These variables specify the binary field in the input record to be compared to the bit constant.

- p1 specifies the first byte of the binary input field to be compared to the bit constant, relative to the beginning of the input record.[8] The first data byte of a fixed-length record (FLR) has relative position 1. The first data byte of a variable-length (VLR) record has relative position 5 (because the first 4 bytes contain the record descriptor word). All fields to be tested must start on a byte boundary and must not extend beyond byte 32752.
- m1 specifies the length of the field to be tested. The length can be 1 to 256 bytes.

Bit constant

A bit string constant that specifies the pattern to which the binary field is compared. If a bit in the constant is 1 or 0, the corresponding bit in the field is compared to 1 or 0, respectively. If a bit in the constant is . (period), the corresponding bit in the field is ignored.

Bit string format

The format for coding a bit string constant is:

B'**bbbbbbbb...bbbbbbbb**'

The value bbbbbbbb represents 8 bits that constitute a byte. Each bit must be 1 (test bit for 1), 0 (test bit for 0) or . (ignore bit). You can specify up to 256 groups of 8 bits. The total number of bits in the mask must be a multiple of 8. A bit constant can only be used for bit comparison tests (BI format and EQ or NE operator).

Padding and truncation

The bit constant is truncated or padded on the right to the byte length of the binary field. The padding character is B'00000000' (all bits equal to 0). Note that the padded bytes are compared to the excess bytes in the binary field.

Recommendation: To ensure that comparison of the padded bytes to the excess bytes in the binary field does not cause unwanted results, shorten the field length to eliminate the padding characters, or increase the length of the bit constant to specify the exact test pattern you want.

[8] If your E15 user exit routine formats the record, p1 must refer to the record as reformatted by the exit.

Including records in the output data set—bit comparison test examples

Example 1

```
INCLUDE   COND=(27,1,CH,EQ,C'D',AND,18,1,BI,EQ,B'1.......')
```

This example illustrates how to only include records in which:

- Byte 27 contains D

  AND
- Byte 18 is equal to the specified pattern of bit 0 on.

Example 2

```
INCLUDE   COND=(11,1,BI,NE,B'10...1.1')
```

This example illustrates how to only include records in which byte 11 is not equal to the specified pattern of bit 0 on, bit 1 off, bit 5 on and bit 7 on. Results for selected field values are shown :

Table 27. Bit Comparison Example 2: Results for Selected Field Values

Bit Comparison Example 2: Results for Selected Field Values11,1,BI Value	11,1,BI Result	Action
X'85'	False	Omit Record
X'C1'	True	Include Record
X'84'	True	Include Record
X'97'	False	Omit Record

Example 3

```
INCLUDE   COND=(11,2,EQ,B'..01....0.....1',
               OR,21,1,EQ,B'01......'),FORMAT=BI
```

This example illustrates how to only include records in which:

- Bytes 11 through 12 are equal to the specified pattern of bit 2 off, bit 3 on, bit 8 off and bit 15 on

  OR
- Byte 21 is equal to the specified pattern of bit 0 off and bit 1 on.

Results for selected field values are shown in Table 28 on page 116

Table 28. Bit Comparison Example 3: Results for Selected Field Values

Bit Comparison Example 3: Results for Selected Field Values11,2,BI Value	11,2,BI Result	21,1,BI Value	21,1,BI Result	Action
X'1221'	True	X'C0'	False	Include Record
X'02C4'	False	X'41'	True	Include Record
X'1234'	False	X'00'	False	Omit Record
X'5F7F'	True	X'7F'	True	Include Record
X'FFFF'	False	X'2F'	False	Omit Record

Date comparisons

You can use DFSORT's Y2 formats in conjunction with the century window in effect, as follows:

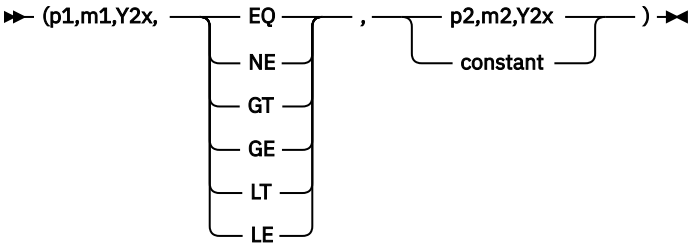
- Use the full date formats (Y2T, Y2U, Y2V, Y2W, Y2X and Y2Y) to compare a two-digit year date field to a two-digit year date constant, a current, future or past two-digit year date (Y constant), or to another two-digit year date field.
- Use the year formats (Y2C, Y2Z, Y2S, Y2P, Y2D and Y2B) to compare a two-digit year field to a two-digit year constant (Y constant) or to another two-digit year field.

For example, you can include only those records for which a Z'yymm' date field is between January 1996 and March 2005. Or you can include only those records for which a P'dddyy' field is less than another P'dddyy' field. Or you can include only those records for which a C'yyddd' field is between today's date and 5 days earlier than today's date.

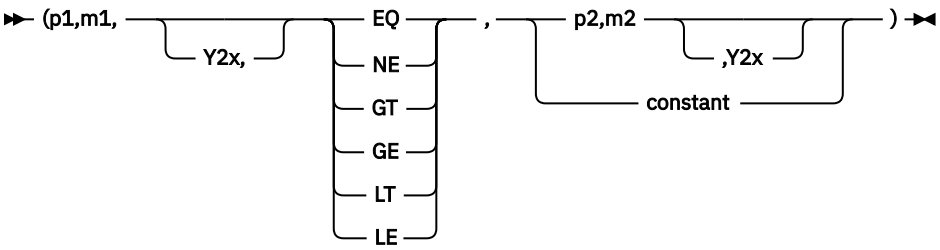
The ordering of dates and special indicators used for comparisons with Y2 fields and Y constants is the same as the ascending orders for sorting and merging Y2 fields (see “SORT control statement” on page 423 for details).

Relational condition format

Two formats for the relational condition can be used:



Or, if the FORMAT=Y2x operand is used:



Comparison operators are as follows:

EQ
Equal to

NE
Not equal to

GT
Greater than

GE
Greater than or equal to

LT
Less than

LE
Less than or equal to.

Fields

p1,m1,Y2x

These variables specify a two-digit year date field in the input record to be compared either to another two-digit year date field in the input record or to a two-digit year date constant.

- p1 specifies the first byte of the date field relative to the beginning of the input record.[9] The first data byte of a fixed-length record (FLR) has relative position 1. The first data byte of a variable-length (VLR) record has relative position 5 (because the first 4 bytes contain the record descriptor word). All date fields must start on a byte boundary, and no date field can extend beyond byte 32752.
- m1 specifies the length of the date field. Appendix C, “Data format descriptions,” on page 823 describes the length and format for each type of date field.
- Y2x specifies the Y2 format. Appendix C, “Data format descriptions,” on page 823 describes the length (m) and format (Y2x) for each type of date field.

You can use p1,m1 rather than p1,m1,Y2x if you use FORMAT=Y2x to supply the format for the date field.

p2,m2,Y2x

These variables specify another two-digit year date field in the input record with which the p1,m1,Y2x field will be compared.

You can use p2,m2 rather than p2,m2,Y2x if you use FORMAT=Y2x to supply the format for the date field.

Constant

A two-digit year date constant in the form Y'string', Y'DATE1', Y'DATE1'+d, Y'DATE1'-d, Y'DATE2', Y'DATE2'+m, Y'DATE2'-m, Y'DATE3', Y'DATE3'+d, or Y'DATE3'-d, with which the p1,m1,Y2x field will be compared.

Comparisons

A date field can be compared to a date constant or another date field with the same number of non-year (x) digits. Table 29 on page 118 shows the type of field-to-field and field-to-constant comparisons you can use. The fields shown for any type of date (for example, yyx and xyy) can be compared to any other fields shown for that type of date or to the Y constant shown for that type of date.

Table 29. Permissible Comparisons for Dates

Permissible Comparisons for Dates Type of Date	**Fields (m,f)**		**Y Constant**
yyx and xyy	3,Y2T 3,Y2W	2,Y2U 2,Y2X	Y'yyx'
yyxx and xxyy	4,Y2T 4,Y2W	3,Y2V 3,Y2Y	Y'yyxx' Y'DATE2' Y'DATE2'+m Y'DATE2'-m
yyxxx and xxxyy	5,Y2T 5,Y2W	3,Y2U 3,Y2X	Y'yyxxx' Y'DATE3' Y'DATE3'+d Y'DATE3'-d

[9] If your E15 user exit routine formats the record, p1 must refer to the record as reformatted by the exit.

Table 29. Permissible Comparisons for Dates (continued)

Permissible Comparisons for Dates Type of Date	Fields (m,f)		Y Constant
yyxxxx and xxxxyy	6,Y2T 6,Y2W	4,Y2V 4,Y2Y	Y'yyxxxx' Y'DATE1' Y'DATE1'+d Y'DATE1'-d
yy	2,Y2C 2,Y2S 1,Y2D	2,Y2Z 2,Y2P 1,Y2B	Y'yy'

Y constants for current, future, and past two-digit year dates are as follows. d can be 0 to 9999 days and m can be 0 to 9999 months.

- Y'DATE1' generates a Y constant for the current date in the form Y'yymmdd'
- Y'DATE1'+d generates a Y constant for the current date plus d days in the form Y'yymmdd'
- Y'DATE1'-d generates a Y constant for the current date minus d days in the form Y'yymmdd'
- Y'DATE2' generates a Y constant for the current date in the form Y'yymm'
- Y'DATE2'+m generates a Y constant for the current date plus m months in the form Y'yymm'
- Y'DATE2'-m generates a Y constant for the current date minus m months in the form Y'yymm'
- Y'DATE3' generates a Y constant for the current date in the form Y'yyddd'
- Y'DATE3'+d generates a Y constant for the current date plus d days in the form Y'yyddd'
- Y'DATE3'-d generates a Y constant for the current date minus d days in the form Y'yyddd'.

You must use the same number of digits in a Y constant as the type of date; leading zeros must be specified (for example, for Y'yymm', use Y'0001' for January 2000 and Y'0501' for January 2005).

You can also use Y constants for special indicators as follows:

- Y'0...0' (CH/ZD/PD zeros) and Y'9...9' (CH/ZD/PD nines) can be used with Y2T, Y2U, Y2V, Y2W, Y2X and Y2Y dates. You must use the same number of digits as the type of date (for example, Y'000' for yyq or qyy, Y'0000' for yymm or mmyy, and so forth).
- Y'LOW' (BI zeros), Y'BLANKS' (blanks) and Y'HIGH' (BI ones) can be used with Y2T, Y2W and Y2S dates.

Including records in the output data set—date comparisons

Example 1

```
INCLUDE FORMAT=Y2T,
  COND=(3,4,GE,Y'9901',AND,
        3,4,LE,Y'0312',OR,
        3,4,LE,Y'0000')
```

This example illustrates how to only include records in which:

- A C'yymm' date field in bytes 3 through 6 is between January 1999 and December 2003

  OR

- Bytes 3 through 6 contain CH zeros (C'0000'), ZD zeros (Z'0000') or BI zeros (X'00000000').

Note that the century window in effect will be used to interpret the Y'9901' and Y'0312' date constants, as well as real dates in the C'yymm' date field. However, the century window will not be used to interpret the Y'0000' special indicator constant or special indicators in the C'yymm' date field.

Example 2

```
INCLUDE COND=(2,3,Y2X,LT,36,5,Y2T)
```

This example illustrates how to only include records in which a P'dddyy' date field in bytes 2 through 4 is less than a Z'yyddd' date field in bytes 36 through 40.

Note that the century window in effect will be used to interpret real dates in the P'dddyy' and Z'yyddd' date fields. However, the century window will not be used to interpret special indicators in the P'dddyy' and Z'yyddd' date fields.

Numeric tests

You can test a field for numerics or non-numerics in character, zoned decimal or packed decimal format.

For example, you can include only those records in which a 5-byte field contains only '0'-'9' characters (that is, character numerics). Or you can include only those records in which a 9-byte field contains invalid zoned decimal data (that is, zoned decimal non-numerics). Or you can include only those records in which a 12-byte field contains valid packed decimal data (that is, packed decimal numerics).

A field to be tested for numerics in character format looks like this in hexadecimal:

```
FdFd...Fd
```

The field is considered to be character numeric if every d is 0-9. (This is equivalent to '0'-'9' for each character.) Otherwise, the field is considered to be character non-numeric. For example, '1234', '0001' and '9999' are all considered to be character numeric, whereas 'A234', '12.3', ' 1' and '123D' are all considered to be character non-numeric.

A field to be tested for numerics in zoned decimal format looks like this in hexadecimal:

```
zdzd...sd
```

The field is considered to be zoned decimal numeric if every z is F, every d is 0-9, and s is C, D or F. Otherwise, the field is considered to be zoned decimal non-numeric. For example, '1234' (X'F1F2F3F4'), '123D' (X'F1F2F3C4') and '123M' (X'F1F2F3D4') are all considered to be zoned decimal numeric, whereas 'A234' (X'C1F2F3F4'), '12.3' (X'F1F24BF3') and '123X' (X'F1F2F3E7') are all considered to be zoned decimal non-numeric.

A field to be tested for numerics in packed decimal format looks like this in hexadecimal:

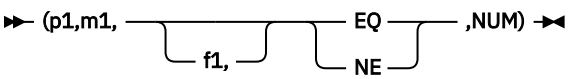
```
dddd...ds
```

The field is considered to be packed decimal numeric if every d is 0-9, and s is C, D or F. Otherwise, the field is considered to be packed decimal non-numeric. For example, X'12345C', X'12345D' and X'12345F' are all considered to be packed decimal numeric, whereas X'A2345C', X'1B345D' and X'12F45F' are all considered to be packed decimal non-numeric.

Relational condition format

Two formats for the relational condition can be used:

Or, if the FORMAT=f operand is used:



Numeric test operators are as follows:

EQ
Equal to numerics

NE
Not equal to numerics (non-numerics)

Fields

p1,m1,f1

These variables specify the field in the input record for the numeric test.

- p1 specifies the first byte of the field relative to the beginning of the input record [10]. The first data byte of a fixed-length record (FLR) has relative position 1. The first data byte of a variable-length (VLR) record has relative position 5 (because the first 4 bytes contain the record descriptor word). All fields must start on a byte boundary, and no field can extend beyond byte 32752.
- m1 specifies the length of the field. The length can be 1 to 256 bytes.
- f1 specifies the type of numerics the field is to be tested for as follows:

  FS
  test for numerics in character format ('0'-'9' (X'F0'-X'F9') in all bytes).

  Note: CSF can be used instead of FS.

  ZD
  test for numerics in zoned decimal format ('0'-'9' (X'F0'-X'F9') in all non-sign bytes; X'F0'-X'F9', X'D0'-X'D9' or X'C0'-X'C9' in the sign byte)

  PD
  test for numerics in packed decimal format (0-9 for all digits; F, D or C for the sign).

You can use p1,m1 rather than p1,m1,f1, if you use FORMAT=f to supply the format for the field.

NUM

Specifies a test for numerics or non-numerics. The condition will be true if the field is numeric when the EQ operator is specified or if the field is non-numeric when the NE operator is specified.

Including records in the output data set--numeric tests

Example 1

```
INCLUDE COND=(1,20,FS,EQ,NUM)
```

This example illustrates how to only include records in which the field in bytes 1 through 20 contains valid character numeric data (that is, '0'-'9' in all bytes).

Example 2

```
INCLUDE COND=(21,8,ZD,NE,NUM,OR,31,5,PD,NE,NUM)
```

This example illustrates how to only include records in which the field in bytes 21 through 28 contains invalid zoned decimal data, or the field in bytes 31 through 35 contains invalid packed decimal data (that is, one of the fields is non-numeric).

Alphanumeric tests

You can test a field for alphanumerics or non-alphanumerics in character format. Various combinations of uppercase characters (A-Z), lowercase characters (a-z) and numeric characters (0-9) can be used.

[10] If your E15 user exit routine formats the record, p1 must refer to the record as reformatted by the exit.

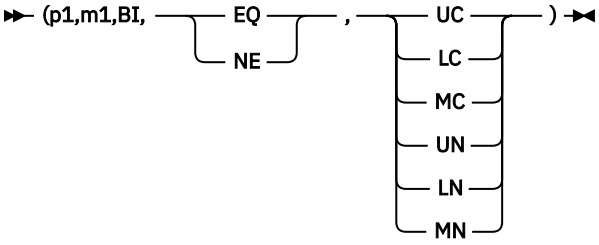
For example, you can include only those records in which a 10-byte field contains only 'A'-'Z' characters (that is, uppercase characters) or '0'-'9' characters (that is, numeric characters). Or you can include only those records in which a 20-byte field contains characters other than 'a'-'z' (that is, lowercase characters).

The following combinations of alphanumeric characters can be used:

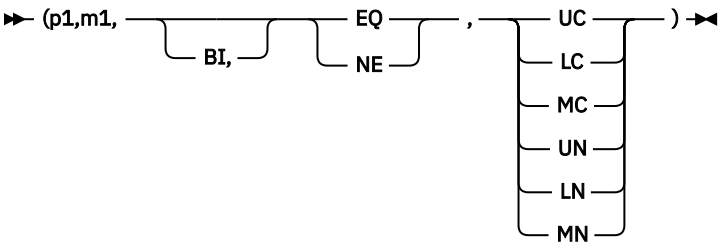
- Uppercase characters (A-Z)
- Lowercase characters (a-z)
- Mixed case characters (A-Z, a-z)
- Uppercase and numeric characters (A-Z, 0-9)
- Lowercase and numeric characters (a-z, 0-9)
- Mixed case and numeric characters (A-Z, a-z, 0-9)

Relational condition format

Two formats for the relational condition can be used:



Or, if the FORMAT=BI operand is used:



Alphanumeric test operators are as follows:

EQ
Equal to specified set of alphanumerics

NE
Not equal to specified set of alphanumerics (non-alphanumerics)

Fields:

p1,m1
These variables specify the field in the input record for the alphanumeric test.

- *p1* specifies the first byte of the field relative to the beginning of the input record[11]. The first data byte of a fixed-length record (FLR) has relative position 1. The first data byte of a variable-length (VLR) record has relative position 5 (because the first 4 bytes contain the record descriptor word). All fields must start on a byte boundary, and no field can extend beyond byte 32752.
- *m1* specifies the length of the field. The length can be 1 to 256 bytes.

[11] If your E15 user exit routine formats the record, p1 must refer to the record as reformatted by the exit.

UC
Specifies a test for the set of uppercase characters (A-Z). The condition will be true if the field is all uppercase characters when the EQ operator is specified or if the field is not all uppercase characters when the NE operator is specified.

LC
Specifies a test for the set of lowercase characters (a-z). The condition will be true if the field is all lowercase characters when the EQ operator is specified or if the field is not all lowercase characters when the NE operator is specified.

MC
Specifies a test for the set of mixed case characters (A-Z and a-z). The condition will be true if the field is all mixed case characters when the EQ operator is specified or if the field is not all mixed case characters when the NE operator is specified.

UN
Specifies a test for the set of uppercase and numeric characters (A-Z and 0-9). The condition will be true if the field is all uppercase or numeric characters when the EQ operator is specified or if the field is not all uppercase or numeric characters when the NE operator is specified.

LN
Specifies a test for the set of lowercase and numeric characters (a-z and 0-9). The condition will be true if the field is all lowercase or numeric characters when the EQ operator is specified or if the field is not all lowercase or numeric characters when the NE operator is specified.

MN
Specifies a test for the set of mixed case and numeric characters (A-Z, a-z or 0-9). The condition will be true if the field is all mixed case or numeric characters when the EQ operator is specified or if the field is not all mixed case or numeric characters when the NE operator is specified.

Including records in the output data set--alphanumeric tests

Example 1

```
INCLUDE COND=(11,10,BI,EQ,MC)
```

This example illustrates how to only include records in which the field in bytes 11 through 20 contains only mixed case characters (that is, 'A'-'Z' or 'a'-'z' in all bytes). A record with 'AaBbZRStuv' in 11-20 would be included, whereas a record with 'Aa7BtuvZQR' would not be included.

Example 2

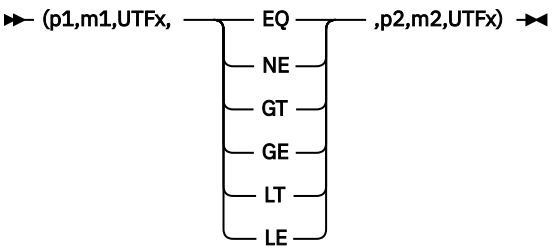
```
INCLUDE COND=(21,4,BI,NE,LN)
```

This example illustrates how to only include records in which the field in bytes 21 through 24 does not contain all lowercase or numeric characters (that is, one of the bytes is not 'a'-'z' or '0'-'9'). A record with 'a,23' would be included, whereas a record with 'a2b9' would not be included.

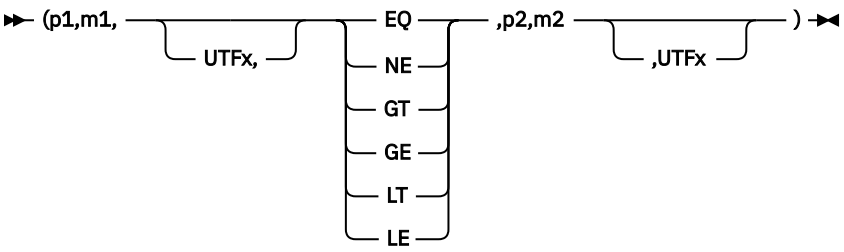
Unicode comparisons

Compare a field in Unicode format to another field in Unicode format. For example, you can include only those records for which a UTF8 field is less than another UTF8 field. Or you can exclude those records for which a UTF16 field is equal to another UTF16 field. Or you can include only those records for which a UTF32 field is greater than another UTF32 field.

Relational condition format:



Or, if the FORMAT=UTFx operand is used:



Comparison operators are as follows:

EQ
Equal to

NE
Not equal to

GT
Greater than

GE
Greater than or equal to

LT
Less than

LE
Less than or equal to

Fields:

p1,m1, UTFx
These variables specify a Unicode data field in the input record to be compared either to another Unicode data field in the input record.

- *p1* specifies the first byte of the Unicode data field relative to the beginning of the input record. The first data byte of a fixed-length record (FLR) has relative position 1. The first data byte of a variable-length (VLR) record has relative position 5 (because the first 4 bytes contain the record descriptor word). All Unicode data fields must start on a byte boundary, and no Unicode data field can extend beyond byte 32752.
- *m1* specifies the number of characters of the Unicode data field to be compared. Acceptable lengths for different formats are shown in Table 30 on page 124.
- *UTFx* specifies the UTF format. Permissible formats are shown in Table 30 on page 124.

Note that you can use p1,m1 rather than p1,m1,UTFx if you use FORMAT=UTFx to supply the format for the Unicode data field.

Table 30. Compare Field Formats and Lengths

Format code	Length	Description
UTF8	1 to 256 Unicode Characters	Unicode Data Format UTF-8

Table 30. Compare Field Formats and Lengths (continued)		
Format code	**Length**	**Description**
UTF16	1 to 256 Unicode Characters	Unicode Data Format UTF16
UTF32	1 to 256 Unicode Characters	Unicode Data Format UTF-32

p2,m2, UTFx

These variables specify another Unicode data field in the input record with which the p1,m1,UTFx field will be compared. Permissible comparisons between compare fields with different formats are shown in Table 31 on page 125.

You can use p2,m2 rather than p2,m2,UTFx if you use FORMAT=UTFx to supply the format for the Unicode data field.

Table 31. Permissible Field-to-Field Comparisons for INCLUDE/OMIT			
Field format	**UTF8**	**UTF16**	**UTF32**
UTF8	X		
UTF16		X	
UTF32			X

Note:

- When specifying the length of the compared fields, you need to specify the number of Unicode characters, but not the number of bytes.
- The lengths of the two fields with Unicode data being compared must be identical.
- The following operands are not supported with INCLUDE/OMIT processing of UTF8/UTF16/UTF32 data formats:
  - LOCALE
  - EFS
  - SUM
  - E61 exit
  - VLSHRT
  - INREC
  - OUTREC

Examples:

```
INCLUDE FORMAT=UTF8,COND=(5,4,LT,11,4)
```

Explanation: Include the records if UTF8 format data at position 5 for 4 characters is less than UTF8 format data at position 11 for 4 characters.

```
OMIT COND=(21,40,UTF16,EQ,151,40,UTF16)
```

Explanation: Omit the records if UTF16 format data at position 21 for 40 characters is equal to UTF16 format data at 151 for 40 characters.

```
OUTFIL INCLUDE=(21,4,UTF32,NE,31,4,UTF32)
```

Explanation: Include the records if UTF32 format data at position 21 for 4 characters is not equal to UTF32 format data at 31 for 4 characters.

INCLUDE/OMIT statement notes

- Floating point compare fields cannot be referenced in INCLUDE or OMIT statements.
-
- Any selection can be performed with either an INCLUDE or an OMIT statement. INCLUDE and OMIT are mutually exclusive.
- If several relational conditions are joined with a combination of AND and OR logical operators, the AND statement is evaluated first. The order of evaluation can be changed by using parentheses inside the COND expression.
- If any changes are made to record formats by user exits E15 or E32, the INCLUDE or OMIT statement must apply to the newest formats.
- DFSORT issues a message and terminates if an INCLUDE or OMIT statement is specified for a tape work data set sort or conventional merge application.
- If SZERO is in effect, -0 compares as less than +0 when numeric fields and constants are used. If NOSZERO is in effect, -0 compares as equal to +0 when numeric fields and constants are used.

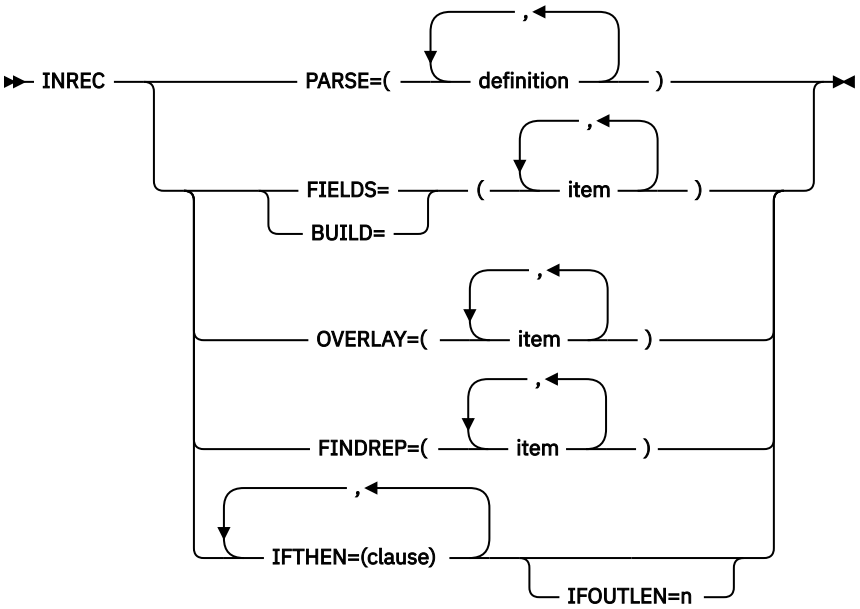
Table 32 on page 126 shows how DFSORT reacts to the result of a relational condition comparison, depending on whether the statement is INCLUDE or OMIT and whether the relational condition is followed by an AND or an OR logical operator.

When writing complex statements, the table in Table 32 on page 126 helps you get the result that you want.

Table 32. Logic Table for INCLUDE/OMIT.

	Relational Condition	**Program action if next logical operator is:**	
Statement	**Compare**	**AND**	**OR**
OMIT	True	Check next compare, or if last compare OMIT record.	OMIT record
OMIT	False	INCLUDE record	Check next compare, or if last compare, INCLUDE record.
INCLUDE	True	Check next compare, or if last compare, INCLUDE record.	INCLUDE record
INCLUDE	False	OMIT record	Check compare, or if last compare, OMIT record.

INREC control statement



The INREC control statement allows you to reformat the input records before they are sorted, merged, or copied.

The INREC control statement supports a wide variety of parsing, editing, and reformatting tasks, including:

- The use of fixed position/length fields or variable position/length fields. For fixed fields, you specify the starting position and length of the field directly. For variable fields, such as delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings (and many other types), you define rules that allow DFSORT to extract the relevant data into fixed parsed fields, and then use the parsed fields as you would use fixed fields.
- Insertion of blanks, zeros, strings, current date, future date, past date, current time, sequence numbers, decimal constants, and the results of arithmetic expressions before, between, and after the input fields in the reformatted records.
-
- Sophisticated conversion capabilities, such as find and replace, hexadecimal display, bit display, translation of EBCDIC letters from lowercase to uppercase or uppercase to lowercase, translation of characters from EBCDIC to ASCII and from ASCII to EBCDIC, translation of characters using the ALTSEQ translation table, conversion of numeric values from one format to another, left-justify or left-squeeze (remove leading blanks or all blanks and shift left), and right-justify or right-squeeze (remove trailing blanks or all blanks and shift right).
- Sophisticated editing capabilities, such as control of the way numeric fields are presented with respect to length, leading or suppressed zeros, thousands separators, decimal points, leading and trailing positive and negative signs, and so on.

  Twenty-seven pre-defined editing masks are available for commonly used numeric editing patterns, encompassing many of the numeric notations used throughout the world. In addition, a virtually unlimited number of numeric editing patterns are available via user-defined editing masks.
- Transformation of SMF, TOD, and ETOD date and time values to more usable forms.
- Conversion of input date fields of one type (CH, ZD, PD, 2-digit year, 4-digit year, Julian, Gregorian) to corresponding output date fields of another type or to a corresponding day of the week.
- Various types of arithmetic operations for input date fields.
- Selection of a character constant, hexadecimal constant, or input field from a lookup table, based on a character, hexadecimal, or bit string as input (that is, lookup and change).

You can create the reformatted INREC records in one of the following ways using unedited, edited, or converted input fields (p,m for fixed fields, or %nn for parsed fields - see PARSE), and a variety of constants:

- **BUILD or FIELDS:** Reformat each record by specifying all of its items one by one. Build gives you complete control over the items you want in your reformatted INREC records and the order in which they appear. You can delete, rearrange and insert fields and constants. Example:

```
INREC BUILD=(1,20,C'ABC',26:5C'*',
       15,3,PD,EDIT=(TTT.TT),21,30,80:X)
```

- **OVERLAY:** Reformat each record by specifying just the items that overlay specific columns. Overlay lets you change specific existing columns without affecting the entire record. Example:

```
INREC OVERLAY=(45:45,8,TRAN=LTOU)
```

- **FINDREP**: Reformat each record by doing various types of find and replace operations. Example:

```
INREC FINDREP=(IN=C'Mr.',OUT=C'Mister')
```

- **IFTHEN clauses:** Reformat different records in different ways by specifying how build, overlay, find/replace, or group operation items are applied to records that meet given criteria. IFTHEN clauses let you use sophisticated conditional logic to choose how different record types are reformatted. Example:

```
INREC IFTHEN=(WHEN=(1,5,CH,EQ,C'TYPE1'),
         BUILD=(1,40,C'**',+1,TO=PD)),
      IFTHEN=(WHEN=(1,5,CH,EQ,C'TYPE2'),
         BUILD=(1,40,+2,TO=PD,X'FFFF')),
      IFTHEN=(WHEN=NONE,OVERLAY=(45:C'NONE'))
```

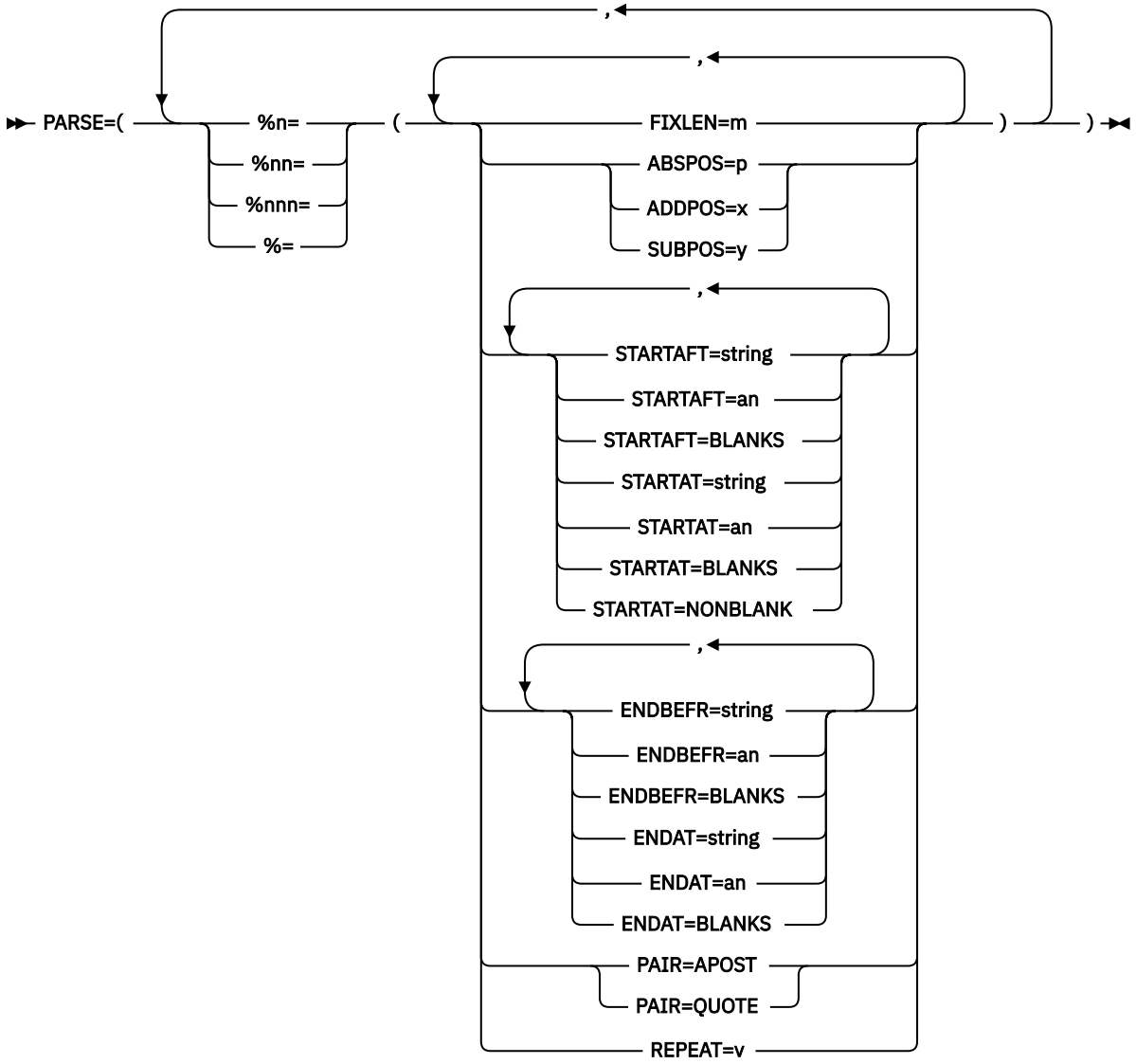
You can choose to include any or all of the following items in your reformatted INREC records:

- Fixed position/length fields or variable position/length fields. For fixed fields, you specify the starting position and length of the field directly. For variable fields, such as delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings (and many other types), you define rules that allow DFSORT to extract the relevant data into fixed parsed fields, and then use the parsed fields as you would use fixed fields.
- Blanks, binary zeros, character strings, and hexadecimal strings
- Current date, future date, past date, and current time in various forms
- Unedited input fields aligned on byte, halfword, fullword, and doubleword boundaries
- Replaced or removed strings.
- Hexadecimal or bit representations of binary input fields
- Characters translated from uppercase to lowercase, lowercase to uppercase, ASCII to EBCDIC or EBCDIC to ASCII.
- Left-justified, right-justified, left-squeezed, or right-squeezed input fields.
- Numeric input fields of various formats converted to different numeric formats, or to character format edited to contain signs, thousands separators, decimal points, leading zeros or no leading zeros, and so on.
- Decimal constants converted to different numeric formats, or to character format edited to contain signs, thousands separators, decimal points, leading zeros or no leading zeros, and so on.
- The results of arithmetic expressions combining fields, decimal constants, operators (MIN, MAX, MUL, DIV, MOD, ADD and SUB) and parentheses converted to different numeric formats, or to character format edited to contain signs, thousands separators, decimal points, leading zeros or no leading zeros, and so on.
- SMF, TOD and ETOD date and time fields converted to different numeric formats, or to character format edited to contain separators, leading zeros or no leading zeros, and so on.
- Input date fields of one type (CH, ZD, PD, 2-digit year, 4-digit year, Julian, Gregorian) converted to corresponding output date fields of another type or to a corresponding day of the week.

- The results of various types of arithmetic operations for input date fields.
- Sequence numbers in various formats.
- A character constant, hexadecimal constant or input field selected from a lookup table, based on a character, hexadecimal or bit constant as input.
- A zoned decimal group identifier, a zoned decimal group sequence number, or a field propagated from the first record of a group to all of the records of a group.

For information concerning the interaction of INREC and OUTREC, see "INREC statement notes" on page 150.

PARSE



This operand allows you to extract variable position/length fields into fixed parsed fields. Parsed fields (%n, %nn or %nnn) can be used where fixed position/length fields (p,m) can be used in the BUILD (or FIELDS) or OVERLAY operands as described later in this section.

Note: Although you can use %n (%0-%9), %nn (%00-%99) or %nnn (%000-%999) for a parsed field, for convenience in this book %nn will be used in general when referring to a parsed field. %n, %0n or %00n can be used interchangeably for parsed field n (for example, %1, %01 or %001 for parsed field 1). %nn or %0nn can be used interchangeably for parsed field nn (for example, %12 or %012 for parsed field 12).

PARSE can be used for many different types of variable fields including delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings, and so on. You can assign up to 1000 parsed fields (%0-%999) to the variable fields you want to extract.

Note that if all of the fields in your records have fixed positions and lengths, you don't need to use PARSE. But if any of the fields in your records have variable positions or lengths, you can use PARSE to treat them as fixed parsed fields in BUILD or OVERLAY. You can mix p,m fields (fixed fields) and %nn fields (parsed fields) in BUILD and OVERLAY.

See PARSE under "OUTFIL Control Statements" for complete details.

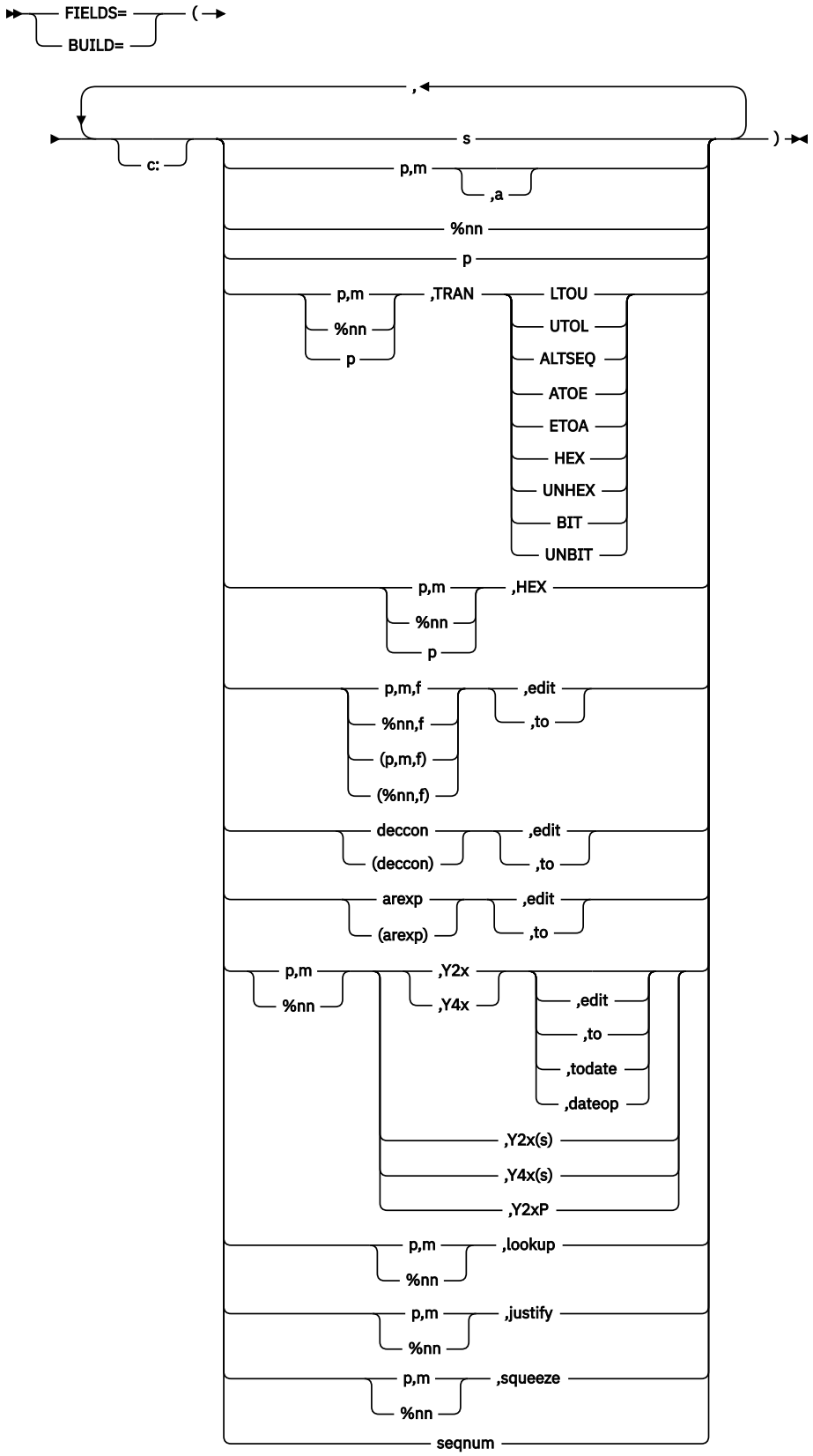
Sample Syntax

```
INREC PARSE=(%00=(ENDBEFR=C'*',FIXLEN=3),
    %01=(ENDBEFR=BLANKS,FIXLEN=6),
    %02=(STARTAT=C'MAX',FIXLEN=8),
    %03=(STARTAFT=C'(',ENDBEFR=C')',FIXLEN=6),
    %04=(STARTAFT=BLANKS,FIXLEN=5)),
  BUILD=(%03,X,%03,HEX,21:%02,31:%01,SFF,M26,LENGTH=7,
         18,6,%00,UFF,M11,LENGTH=3,%04,JFY=(SHIFT=RIGHT))
```

Default for PARSE: None; must be specified. See Appendix B, “Specification/override of DFSORT options,” on page 805.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

FIELDS or BUILD



Specifies all of the items in the reformatted INREC record in the order in which they are to be included. The reformatted INREC record consists of the separation fields, edited and unedited input fields (p,m for fixed fields, or %nn for parsed fields - see PARSE),, edited decimal constants, edited

results of arithmetic expressions, and sequence numbers you select, in the order in which you select them, aligned on the boundaries or in the columns you indicate.

For variable-length records, the first item in the BUILD or FIELDS parameter must specify or include the unedited 4-byte record descriptor word (RDW), that is, you must start with 1,m with m equal to or greater than 4. If you want to include the bytes from a specific position to the end of each input record at the end of each reformatted output record, you can specify that starting position (p) as the last item in the BUILD or FIELDS parameter. For example:

```
INREC FIELDS=(1,4,          unedited RDW
   1,2,BI,TO=ZD,LENGTH=5,   display RDW length in decimal
   C'|',                    | separator
   5)                       display input positions 5 to end
```

For fixed-length records, the first input and output data byte starts at position 1. For variable-length records, the first input and output data byte starts at position 5, after the RDW in positions 1-4.

c:

Specifies the position (column) for a separation field, input field, decimal constant, arithmetic expression, or sequence number, relative to the start of the reformatted input record. Unused space preceding the specified column is padded with EBCDIC blanks. The following rules apply:

- c must be a number between 1 and 32752.
- c: must be followed by a separation field, input field, decimal constant, or arithmetic expression.
- c must not overlap the previous input field or separation field in the reformatted input record.
- for variable-length records, c: must not be specified before the first input field (the record descriptor word) nor after the variable part of the input record.
- The colon (:) is treated like the comma (,) or semicolon (;) for continuation to another line.

Both valid and invalid examples are shown in Table 33 on page 132.

Table 33. Examples of Valid and Invalid Column Alignment . Examples of Valid and Invalid Column Alignment

Validity	Specified	Result
Valid	33:C'State '	Columns 1-32 — blank Columns 33-38 — 'State '
Valid	20:5,4,30:10,8	Columns 1-19 — blank Columns 20-23 — input field (5,4) Columns 24-29 — blank Columns 30-37 — input field (10,8)
Invalid	0:5,4	Column value cannot be zero.
Invalid	:25Z	Column value must be specified.
Invalid	32753:21,8	Invalid — column value must be less than 32753.
Invalid	5:10:2,5	Column values cannot be adjacent.
Invalid	20,10,6:C'AB'	Column value overlaps previous field.

s

Specifies that a separation field (blanks, zeros, character string, hexadecimal string, current date, future date, past date, or current time) is to appear in the reformatted input record. It can be specified before or after any input field. Consecutive separation fields can be specified. For variable-length records, separation fields must not be specified before the first input field (the record descriptor word), or after the variable part of the input record. Permissible values are nX, nZ, nC'xx...x', nX'yy...yy', and various date and time constants.

nX

Blank separation. n bytes of EBCDIC blanks (X'40') are to appear in the reformatted input records. n can range from 1 to 4095. If n is omitted, 1 is used.

Examples of valid and invalid blank separation are shown in Table 34 on page 133.

Table 34. Examples of Valid and Invalid Blank Separation

Examples of Valid and Invalid Blank SeparationValidity	Specified	Result
Valid	X or 1X	1 blank
Valid	4095X	4095 blanks
Invalid	5000X	Too many repetitions. Use two adjacent separation fields instead (2500X,2500X, for example)
Invalid	0X	0 is not allowed.

nZ

Binary zero separation. n bytes of binary zeros (X'00') are to appear in the reformatted input records. n can range from 1 to 4095. If n is omitted, 1 is used.

Examples of valid and invalid binary zero separation are shown in Table 35 on page 133.

Table 35. Examples of Valid and Invalid Binary Zero Separation

Examples of Valid and Invalid Binary Zero SeparationValidity	Specified	Result
Valid	Z or 1Z	1 binary zero
Valid	4095Z	4095 binary zeros
Invalid	4450Z	Too many repetitions. Use two adjacent separation fields instead (4000Z,450Z for example).
Invalid	0Z	0 is not allowed.

nC'xx...x'

Character string separation. n repetitions of the character string constant (C'xx...x') are to appear in the reformatted input records. n can range from 1 to 4095. If n is omitted, 1 is used. x can be any EBCDIC character. You can specify from 1 to 256 characters.

If you want to include a single apostrophe in the character string, you must specify it as two single apostrophes:

```
Required:  O'NEILL     Specify:  C'O''NEILL'
```

Examples of valid and invalid character string separation are shown in Table 36 on page 134.

Table 36. Examples of Valid and Invalid Character String Separation

Examples of Valid and Invalid Character String Separation Validity	Specified	Result	Length
Valid	C'John Doe'	John Doe	8
Valid	C'JOHN DOE'	JOHN DOE	8
Valid	C'$@#'	$@#	3
Valid	C'+0.193'	+0.193	6
Valid	4000C' '	8000 blanks	8000
Valid	20C'**FILLER**'	**FILLER** repeated 20 times	200
Valid	C'Frank''s'	Frank's	7
Invalid	C'''''	Apostrophes not paired	n/a
Invalid	'ABCDEF'	C identifier missing	n/a
Invalid	C'ABCDE	Apostrophe missing	n/a
Invalid	4450C'1'	Too many repetitions. Use two adjacent separation fields instead (4000C'1',450C'1', for example).	n/a
Invalid	0C'ABC'	0 is not allowed	n/a
Invalid	C''	No characters specified	n/a
Invalid	C'Frank's'	Two single apostrophes needed for one	n/a

nX'yy...yy'

Hexadecimal string separation. n repetitions of the hexadecimal string constant (X'yy...yy') are to appear in the reformatted input records. n can range from 1 to 4095. If n is omitted, 1 is used.

The value yy represents any pair of hexadecimal digits. You can specify from 1 to 256 pairs of hexadecimal digits. Examples of valid and invalid hexadecimal string separation are shown in Table 37 on page 135.

Table 37. Examples of Valid and Invalid Hexadecimal String Separation

Examples of Valid and Invalid Hexadecimal String Separation Validity	Specified	Result	Length
Valid	X'FF'	FF	1
Valid	X'BF3C'	BF3C	2
Valid	3X'00000F'	00000F00000F00000F	9
Valid	4000X'FFFF'	FF repeated 8000 times	8000
Invalid	X'ABGD'	G is not a hexadecimal digit	n/a
Invalid	X'F1F'	Incomplete pair of digits	n/a
Invalid	'BF3C'	X identifier missing	n/a
Invalid	'F2F1'X	X in wrong place	n/a
Invalid	8000X'01'	Too many repetitions. Use two adjacent separation fields instead (4000X'01',4000X'01', for example).	n/a
Invalid	0X'23AB'	0 is not allowed	n/a
Invalid	X''	No hexadecimal digits specified	n/a

DATEn, DATEn(c), DATEnP
Constant for current date. The current date of the run is to appear in the reformatted input records. See DATEn, DATEn(c), DATEnP under OUTFIL OUTREC for details.

&DATEn, &DATEn(c), &DATEnP
Can be used instead of DATEn, DATEn(c) and DATEnP, respectively.

DATEn+r, DATEn(c)+r, DATEnP+r
Constant for future date. A future date relative to the current date of the run is to appear in the reformatted input records. See DATEn+r, DATEn(c)+r, DATEnP+r under OUTFIL OUTREC for details.

&DATEn+r, &DATEn(c)+r, &DATEnP+r
Can be used instead of DATEn+r, DATEn(c)+r and DATEnP+r, respectively.

DATEn-r, DATEn(c)-r, DATEnP-r
Constant for past date. A past date relative to the current date of the run is to appear in the reformatted input records. See DATEn-r, DATEn(c)-r, DATEnP-r under OUTFIL OUTREC for details.

&DATEn-r, &DATEn(c)-r, &DATEnP-r
Can be used instead of DATEn-r, DATEn(c)-r and DATEnP-r, respectively.

TIMEn, TIMEn(c), TIMEnP
Constant for current time. The time of the run is to appear in the reformatted input records. See TIMEn, TIMEn(c), TIMEnP under OUTFIL OUTREC for details.

&TIMEn, &TIMEn(c), &TIMEnP
Can be used instead of TIMEn, TIMEn(c) and TIMEnP, respectively.

DATE
Specifies that the current date is to appear in the reformatted input records in the form 'mm/dd/yy'. See DATE under OUTFIL OUTREC for details.

&DATE
Can be used instead of DATE.

DATE=(abcd)
Specifies that the current date is to appear in the reformatted input records in the form 'adbdc'. See DATE=(abcd) under OUTFIL OUTREC for details.

&DATE=(abcd)
Can be used instead of DATE=(abcd).

DATENS=(abc)
Specifies that the current date is to appear in the reformatted input records in the form 'abc'. See DATENS=(ab) under OUTFIL OUTREC for details.

&DATENS=(abc)
Can be used instead of DATENS=(abc).

YDDD=(abc)
specifies that the current date is to appear in the reformatted input records in the form 'acb'. See YDDD=(abc) under OUTFIL OUTREC for details.

&YDDD=(abc)
Can be used instead of YDDD=(abc).

YDDDNS=(ab)
specifies that the current date is to appear in the reformatted input records in the form 'ab'. See YDDDNS=(ab) under OUTFIL OUTREC for details.

&YDDDNS=(ab)
can be used instead of YDDDNS=(ab).

TIME
specifies that the current time is to appear in the reformatted input records in the form 'hh:mm:ss'. See TIME under OUTFIL OUTREC for details.

&TIME
Can be used instead of TIME.

TIME=(abc)
specifies that the current time is to appear in the reformatted input records in the form 'hhcmmcss' (24-hour time) or 'hhcmmcss xx' (12-hour time). See TIME=(abc) under OUTFIL OUTREC for details.

&TIME=(abc)
Can be used instead of TIME=(abc).

TIMENS=(ab)
specifies that the current time is to appear in the reformatted input record in the form 'hhmmss' (24-hour time) or 'hhmmss xx' (12-hour time). See TIMENS=(ab) under OUTFIL OUTREC for details.

&TIMENS=(ab)
Can be used instead of TIMENS=(ab).

p,m,a

Specifies that an unedited input field is to appear in the reformatted input record.

p
Specifies the first byte of the input field relative to the beginning of the input record.[12] The first data byte of a fixed-length record has relative position 1. The first data byte of a variable-length record has relative position 5 (because the first 4 bytes contain the RDW). All fields must start on a byte boundary, and no field can extend beyond byte 32752. For special rules concerning variable-length records, see “INREC statement notes” on page 150.

[12] If your E15 user exit reformats the record, p must refer to the record as reformatted by the exit.

m
Specifies the length of the input field. It must include the sign if the data is signed, and must be an integer number of bytes. See "INREC statement notes" on page 150 for more information.

a
Specifies the alignment (displacement) of the input field in the reformatted input record relative to the start of the reformatted input record.

Permissible values of **a** are:

H
Halfword aligned. The displacement (p-1) of the field from the beginning of the reformatted input record, in bytes, is a multiple of two (that is, position 1, 3, 5, and so forth).

F
Fullword aligned. The displacement is a multiple of four (that is, position 1, 5, 9, and so forth).

D
Doubleword aligned. The displacement is a multiple of eight (that is, position 1, 9, 17, and so forth).

Alignment can be necessary if, for example, the data is to be used in a COBOL application program where items are aligned through the SYNCHRONIZED clause. Unused space preceding aligned fields will always be padded with binary zeros.

%nn
specifies that an unedited parsed input field is to appear in the reformatted input record. See PARSE for details of parsed fields. See p,m,a for further details. Note that alignment (H, F, D) is not permitted for %nn fields (for example, %nn,F results in an error message and termination).

p
specifies that the unedited variable part of the input record (that part beyond the minimum record length), is to appear in the reformatted input record, as the last field. p without m can only be used for variable-length records; not for fixed-length records.

Attention: If 1,4,p is specified (only RDW and variable part of record), "null" records containing only an RDW will result if the record length is less than p.

A value must be specified for p that is less than or equal to the minimum record length (RECORD statement L4 value) plus 1 byte.

p,m,TRAN=keyword
Specifies that an input field is to be translated as indicated by the keyword. See p,m,TRAN=keyword under OUTFIL OUTREC for details.

%nn,TRAN=keyword
Specifies that a parsed input field is to be translated as indicated by the keyword. See %nn,TRAN=keyword under OUTFIL OUTREC for details.

p,TRAN=keyword
Specifies that the variable part of the input record is to be translated as indicated by the keyword. See p,TRAN=keyword under OUTFIL OUTREC for details.

p,m,HEX
Can be used instead of p,m,TRAN=HEX. See p,m,HEX under OUTFIL OUTREC for details.

%nn,HEX
Can be used instead of %nn,TRAN=HEX. See %nn,HEX under OUTFIL OUTREC for details.

p,HEX
Can be used instead of p,TRAN=HEX. See p,HEX under OUTFIL OUTREC for details.

p,m,f,edit or (p,m,f),edit
specifies that an edited numeric input field is to appear in the reformatted input record. You can edit BI, FI, PD, PD0, ZD, FL, CSF, FS, UFF, SFF, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3 or TM4 fields using either pre-defined edit masks (M0-M26) or specific edit patterns you define. You can control the way the edited fields look with respect to length, leading or suppressed zeros, thousands separators, decimal points, leading and trailing positive and negative signs, and so on.

See p,m,f,edit under OUTFIL OUTREC for details.

Sample Syntax:

```
INREC FIELDS=(5:21,8,ZD,M19,X,46,5,ZD,M13,
              31:(35,6,FS),SIGNS=(,,+,-),LENGTH=10,
              51:8,4,PD,EDIT=(**II,IIT.TTXS),SIGNS=(,,+,-))
```

%nn,f,edit or (%nn,f),edit
specifies that an edited numeric parsed input field is to appear in the reformatted input record. See PARSE for details of parsed fields. See p,m,f,edit or (p,m,f),edit for further details.

p,m,f,to or (p,m,f),to
specifies that a converted numeric input field is to appear in the reformatted input record. You can convert BI, FI, PD, PD0, ZD, FL, CSF, FS, UFF, SFF, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3, or TM4 fields to BI, FI, PD, PDC, PDF, ZD, ZDF, ZDC, or CSF/FS fields.

See p,m,f,to under OUTFIL OUTREC for details.

Sample Syntax:

```
INREC FIELDS=(21,5,ZD,TO=PD,X,(8,4,ZD),FI,LENGTH=2,X,55,4,FL,TO=FS)
```

%nn,f,to or (%nn,f),to
specifies that a converted numeric parsed input field is to appear in the reformatted input record. See PARSE for details of parsed fields. See p,m,f,to or (p,m,f),to for further details.

deccon,edit or (deccon),edit
specifies that an edited decimal constant is to appear in the reformatted input record. The decimal constant must be in the form +n or -n where n is 1 to 31 decimal digits. The sign (+ or -) must be specified. A decimal constant produces a signed, 31-digit zoned decimal (ZD) result to be edited as specified.

See deccon,edit under OUTFIL OUTREC for details.

Sample Syntax:

```
INREC FIELDS=(5:+5000,EDIT=(T,TTT),X,
  (-25500),M18,LENGTH=8)
```

deccon,to or (deccon),to
specifies that a converted decimal constant is to appear in the reformatted input record. The decimal constant must be in the form +n or -n where n is 1 to 31 decimal digits. The sign (+ or -) must be specified. A decimal constant produces a signed, 31-digit zoned decimal (ZD) result to be converted as specified.

See deccon,to under OUTFIL OUTREC for details.

Sample Syntax:

```
INREC FIELDS=(+0,TO=PD,LENGTH=6,3Z,(-512000),FI)
```

arexp,edit or (arexp),edit
specifies that the edited result of an arithmetic expression is to appear in the reformatted input record. The arithmetic expression can consist of input fields, decimal constants, operators and

parentheses. An arithmetic expression produces a signed, 31-digit zoned decimal (ZD) result to be edited as specified.

See arexp,edit under OUTFIL OUTREC for details.

Sample Syntax:

```
INREC FIELDS=(C'**',27,2,FI,MIN,
                    83,4,PD,EDIT=(STTTTTTT),SIGNS=(+,-),
  15:(((15,5,ZD,ADD,+1),MUL,+100),DIV,62,2,PD),M25,LENGTH=10)
```

arexp,to or (arexp),to
specifies that the converted result of an arithmetic expression is to appear in the reformatted input record. The arithmetic expression can consist of input fields, decimal constants, operators and parentheses. An arithmetic expression produces a signed, 31-digit zoned decimal (ZD) result to be converted as specified.

See arexp,to under OUTFIL OUTREC for details.

Sample Syntax:

```
INREC FIELDS=((15,6,FS,SUB,+5),ADD,(-1,MUL,36,6,FS),ZD,X,
   3,2,FI,MIN,-6,LENGTH=4,TO=PD)
```

p,m,Y2x or p,m,Y4x
Specifies that an input date field is to be edited. Real Y2x dates are edited using the century window established by the Y2PAST option in effect. Y2x and Y4x dates with special indicators are expanded appropriately (for example, p,6,Y2T transforms C'000000' to C'00000000').

See "p,m,Y2x or p,m,Y4x" under OUTFIL OUTREC for details.

Sample Syntax:

```
INREC BUILD=(21,3,Y2U,X,3,8,Y4W)
```

%nn,Y2x or %nn,Y4x
Specifies that a parsed input date field is to be edited. See PARSE for details of parsed fields. See "p,m,Y2x or p,m,Y4x" for further details.

p,m,Y2x,edit or p,m,Y4x,edit
Specifies that the output for a p,m,Yxx input date field is to be edited according to the edit parameters you specify. For example, if you specify:

```
INREC BUILD=(28,5,Y4V,EDIT=(TTTT-TT-TT)
```

The P'ccyymmdd' (X'0ccyymmddC') date value will be transformed to a C'ccyymmdd' date value and then edited to a C'ccyy-mm-dd' date value.

See "p,m,Y2x or p,m,Y4x" and "p,m,f,edit" for related details.

%nn,Y2x,edit or %nn,Y4x,edit
Specifies that the output for a parsed Yxx input date field is to be edited according to the edit parameters you specify. See PARSE for details of parsed fields. See "p,m,Y2x,edit or p,m,Y4x,edit" for further details.

p,m,Y2x,to or p,m,Y4x,to
Specifies that an input date field is to be converted according to the to parameters you specify. For example, if you specify:

```
INREC BUILD=(5,6,Y2W,TO=PD,LENGTH=5)
```

The C'mmddyy' date value will be transformed to a Z'mmddccyy' date value and then converted to a P'mmddccyy' (X'0mmddccyyC') date value.

See "p,m,Y2x or p,m,Y4x" and "p,m,f,to" for related details.

%nn,Y2x,to or %nn,Y4x,to
Specifies that a parsed input date field is to be converted according to the to parameters you specify. See PARSE for details of parsed fields. See "p,m,Y2x,to or p,m,Y4x,to" for further details.

p,m,Y2x,todate or p,m,Y4x,todate
Specifies that an input date field of one type is to be converted to a corresponding output date field of another type. You can convert date fields between combinations of 2-digit and 4-digit year dates, CH/ZD and PD dates, and Julian and Gregorian dates. You can also convert a date field to a corresponding day of the week in several forms.

See "p,m,Y2x,todate or p,m,Y4x,todate" under OUTFIL OUTREC for details.

Sample Syntax:

```
* Convert a P'dddyy' input date to a C'ccyy/mm/dd' output date
  INREC BUILD=(21,3,Y2X,TOGREG=Y4T(/),X,
* Convert a C'ccyymmdd' input date to a P'ccyyddd' output date
     42,8,Y4T,TOJUL=Y4U,X,
* Convert a C'mmddyy' input date to a C'yymmdd' output date
     11,6,Y2W,TOGREG=Y2T)
```

%nn,Y2x,todate or %nn,Y4x,todate
Specifies that an input date field of one type is to be converted to a corresponding output date field of another type. See PARSE for details of parsed fields. See "p,m,Y2x,todate or p,m,Y4x,todate" for further details.

p,m,Y2x,dateop or p,m,Y4x,dateop
Specifies an arithmetic operation for an input date field. See "p,m,Y2x,dateop or p,m,Y4x,dateop" under OUTFIL OUTREC for details.

%nn,Y2x,dateop or %nn,Y4x,dateop
Specifies an arithmetic operation for a parsed input date field. See "%nn,Y2x,dateop or %nn,Y4x,dateop" under OUTFIL OUTREC for details.

p,m,Y2x(s) or p,m,Y4x(s)
Specifies that an input date field is to be edited with separators. s can be any character except a blank. Real Y2x dates are edited using the century window established by the Y2PAST option in effect. Y2x and Y4x dates with special indicators are expanded appropriately (for example, p,8,Y4T(/) transforms C'00000000' to C'0000/00/00').

See "p,m,Y2x(s) or p,m,Y4x(s)" under OUTFIL OUTREC for details.

Sample Syntax:

```
* Convert a Z'dddccyy' date to a C'ddd/ccyy' date.
  INREC BUILD=(19,7,Y4W(/),X,
* Convert a P'ccyymmdd' date to a C'ccyy-mm-dd' date.
     43,5,Y4V(-))
```

%nn,Y2x(s) or %nn,Y4x(s)
Specifies that a parsed input date field is to be edited with separators. See PARSE for details of parsed fields. See "p,m,Y2x(s) or p,m,Y4x(s)" for further details.

p,m,Y2xP
Specifies that an input date field is to be converted to a packed decimal output date field. Real Y2x dates are edited using the century window established by the Y2PAST option in effect. Y2x and Y4x dates with special indicators are expanded appropriately (for example, p,6,Y2TP transforms C'000000' to P'00000000').

See "p,m,Y2xP" under OUTFIL OUTREC for details.

Sample Syntax:

```
  INREC BUILD=(11,3,Y2XP,X,21,4,Y2WP)
```

%nn,Y2xP
Specifies that a parsed input date field is to be converted to a packed decimal output date field. See PARSE for details of parsed fields. See "p,m,Y2xP" for further details.

p,m,lookup or %nn,lookup
specifies that a character constant, hexadecimal constant, input field (p,m), or parsed input field (%nn) from a lookup table is to appear in the reformatted input record. You can use p,m,lookup or %nn,lookup to select a specified character set constant (that is, a character or hexadecimal string) or set field (that is, an input field or parsed input field) based on matching an input value against find constants (that is, character, hexadecimal, or bit constants).

See p,m,lookup or %nn,lookup under OUTFIL OUTREC for details.

Sample Syntax:

```
  INREC FIELDS=(11,1,
                CHANGE=(6,
                  C'R',C'READ',
                  C'U',C'UPDATE',
                  X'FF',C'EMPTY',
                  C'A',C'ALTER'),
                NOMATCH=(11,6),
                4X,
                21,1,
                CHANGE=(10,
                  B'.1......',C'VSAM',
                  B'.0......',C'NON-VSAM'))
```

p,m,justify
specifies that a left-justified or right-justified input field is to appear in the reformatted input record. For a left-justified field, leading blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For a right-justified field, trailing blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed. Optionally:

- specific leading and trailing characters can be changed to blanks before justification begins
- a leading string can be inserted
- a trailing string can be inserted
- the output length can be changed (it's equal to the input length by default)

See p,m,justify under OUTFIL OUTREC for details.

Sample Syntax:

```
INREC FIELDS=(1,10,
   21,20,JFY=(SHIFT=RIGHT),5X,
   52,12,JFY=(SHIFT=LEFT,PREBLANK=C'()',LEAD=C'VOL=SER=',
          LENGTH=16))
```

%nn,justify
specifies that a left-justified or right-justified parsed input field is to appear in the reformatted input record. See PARSE for details of parsed fields. See p,m,justify for further details.

p,m,squeeze
specifies that a left-squeezed or right-squeezed input field is to appear in the reformatted input record. For a left-squeezed field, all blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For a right-squeezed field, all blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed. Optionally:

- specific characters can be changed to blanks before squeezing begins
- a leading string can be inserted
- a trailing string can be inserted
- a string (for example, a comma delimiter) can be inserted wherever a group of blanks is removed between the first nonblank and the last nonblank

- blanks can be kept as is between paired apostrophes ('AB CD EF') or paired quotes ("AB CD EF")
- the output length can be changed (it's equal to the input length by default)

See p,m,squeeze under OUTFIL OUTREC for details.

Sample Syntax:

```
INREC FIELDS=(21,20,SQZ=(SHIFT=LEFT),5X,
   152,18,SQZ=(SHIFT=RIGHT,PREBLANK=X'00',
     LEAD=C'<',MID=C',',TRAIL=C'>',PAIR=APOST))
```

%nn,squeeze
specifies that a left-squeezed or right-squeezed parsed input field is to appear in the reformatted input record. See PARSE for details of parsed fields. See p,m,squeeze for further details.

seqnum
specifies that a sequence number is to appear in the reformatted input record. The sequence numbers are assigned in the order in which the records are received for INREC processing. You can create BI, PD, ZD, CSF, or FS sequence numbers and control their lengths, starting values and increment values. You can restart the sequence number at the start value each time a specified input field (p,m) or parsed input field (%nn) changes.

See seqnum under OUTFIL OUTREC for details.

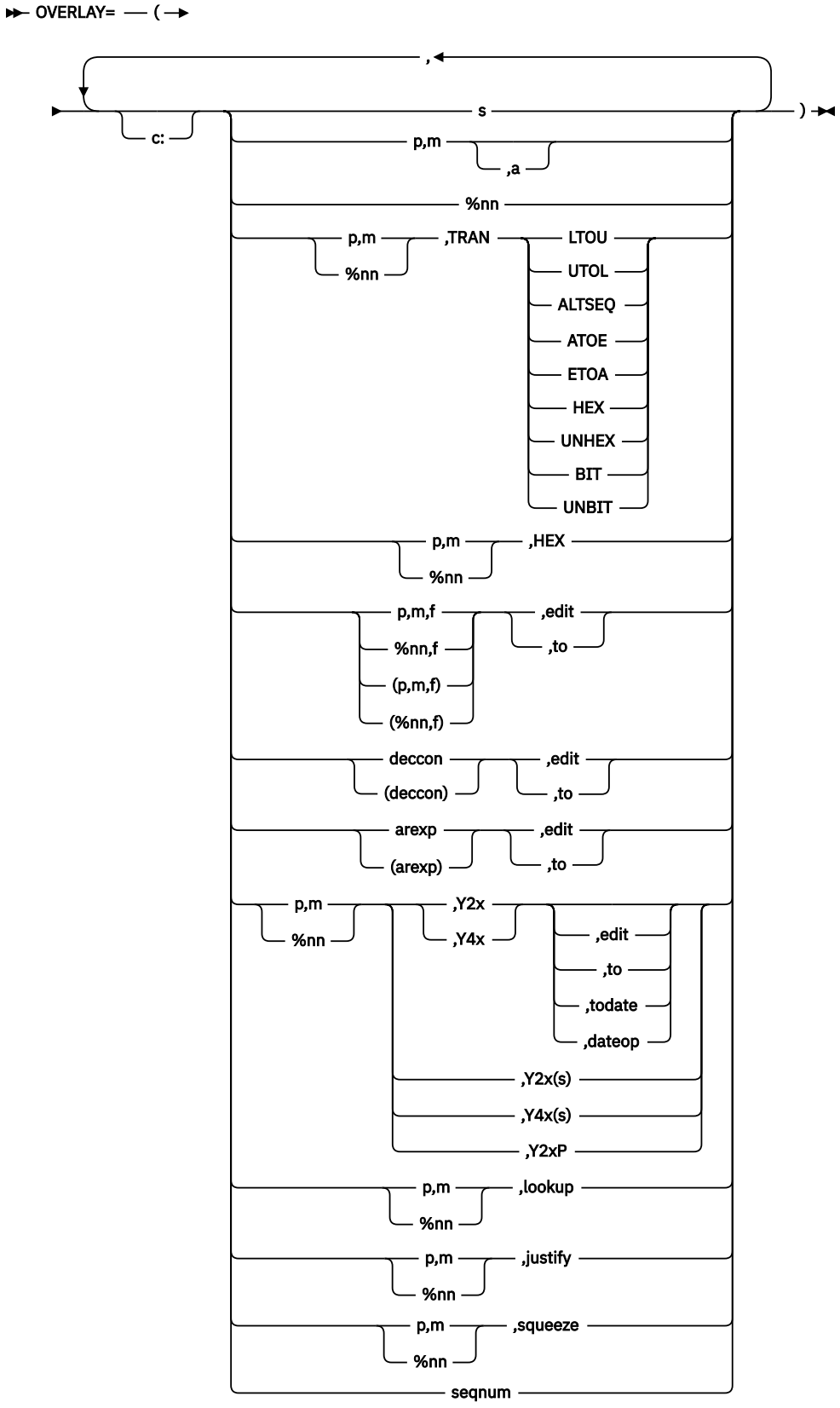
Sample Syntax:

```
  INREC FIELDS=(1,80,SEQNUM,8,ZD)
```

Default for BUILD or FIELDS: None; must be specified. See Appendix B, “Specification/override of DFSORT options,” on page 805.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

OVERLAY



Specifies each item that is to overlay specific columns in the reformatted record. Columns that are not overlaid remain unchanged. If you want to insert, rearrange, or delete fields, use BUILD or FIELDS

rather than OVERLAY. Use OVERLAY only to overlay existing columns or to add fields at the end of every record. OVERLAY can be easier to use then BUILD or FIELDS when you just want to change a few fields without rebuilding the entire record.

For fixed-length records, the first input and output data byte starts at position 1. For variable-length records, the first input and output data byte starts at position 5, after the RDW in positions 1-4.

Use c: (column) to specify the output positions to be overlaid. If you do not specify c: for the first item, it defaults to 1:. If you do not specify c: for any other item, it starts after the previous item. For example, if you specify:

```
INREC OVERLAY=(25,2,11:C'A',15,3,C'**')
```

Input positions 25-26 are placed at output positions 1-2; C'A' is placed at output position 11; input positions 15-17 are placed at output positions 12-14; and C'**' is placed at output positions 15–16. The rest of the record remains unchanged.

You can specify items in any order, you can change the same item multiple times and you can overlap output columns. **Changes to earlier items affect changes to later items.** For example, say you specify:

```
INREC OVERLAY=(21:8,4,ZD,MUL,+10,TO=ZD,LENGTH=6,
    5:5,1,TRAN=UTOL,
    5:5,1,CHANGE=(1,C'a',C'X',C'b',C'Y'),NOMATCH=(5,1))
```

and input position 5 has 'A'. The second item (UTOL) would change 'A' to 'a' and the third item (CHANGE) would change 'a' again to 'X'.

If you specify an OVERLAY item that extends the overlay record beyond the end of the input record, the reformatted record length is automatically increased to that length, and blanks are filled in on the left as needed. For variable-length records, the RDW length is also increased to correspond to the larger reformatted record length after all of the OVERLAY items are processed. For example, if your input record has a length of 40 and you specify:

```
INREC OVERLAY=(16:C'ABC',51:5C'*',35:15,2)
```

the output record is given a length of 55. Blanks are filled in from columns 41-50. For variable-length records, the length in the RDW is changed from 40 to 55 after all of the OVERLAY items are processed.

Missing bytes in specified input fields are replaced with blanks so the padded fields can be processed.

See INREC FIELDS for details of the items listed in the OVERLAY syntax diagram previously. You can specify all of the items for OVERLAY in the same way that you can specify them for BUILD or FIELDS with the following exceptions:

- You cannot specify p or p,HEX or p,TRAN=keyword for OVERLAY.
- For p,m,H or p,m,F or p,m,D fields specified for OVERLAY, fields are aligned as necessary without changing the preceding bytes.
- For variable-length records, you must not overlay positions 1-4 (the RDW) for OVERLAY, so be sure to specify the first column (c:) as 5 or greater. If you do not specify the first column, it will default to 1: which is invalid for variable-length records with OVERLAY. Whereas FIELDS=(1,m,...) is required, OVERLAY=(1,m) is not allowed, since it would overlay the RDW.

Sample Syntax:

Fixed input records:

```
INREC OVERLAY=(21:21,4,ZD,TO=PD,LENGTH=4,
         2:5,8,HEX,45:C'*',32,4,C'*',81:SEQNUM,5,ZD)
```

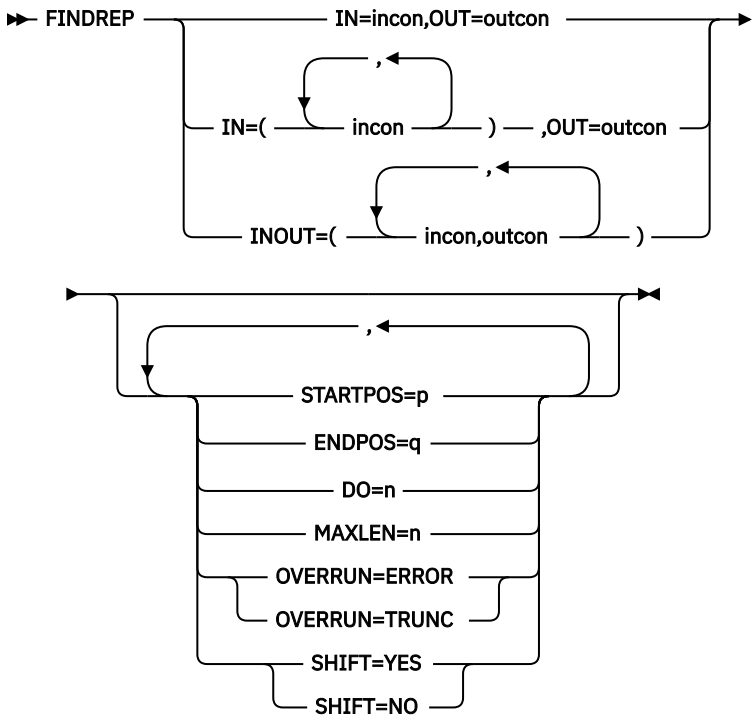
Variable input records:

```
INREC OVERLAY=(5:X'0001',28:C'Date is ',YDDDNS=(4D),
         17:5C'*')
```

Default for OVERLAY: None; must be specified. See Appendix B, “Specification/override of DFSORT options,” on page 805.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

FINDREP



You can use FINDREP to find constants anywhere in a record and replace them with other constants of the same or different lengths. You can find character or hexadecimal input constants anywhere in your records and replace them with character, hexadecimal or null output constants. As appropriate, bytes can be shifted left or right, blank padding can be added for fixed-length records, and the length can be changed for variable-length records.

Various options of FINDREP allow you to define one or more input constants and a corresponding output constant, define one or more pairs of input and output constants, start and end the find scan at specified positions, stop after a specified number of constants are replaced, increase or decrease the length of the output record, define the action to be taken if nonblank characters overrun the end of the record, and specify whether output constants are to replace or overlay input constants.

See FINDREP under "OUTFIL Control Statements" for complete details.

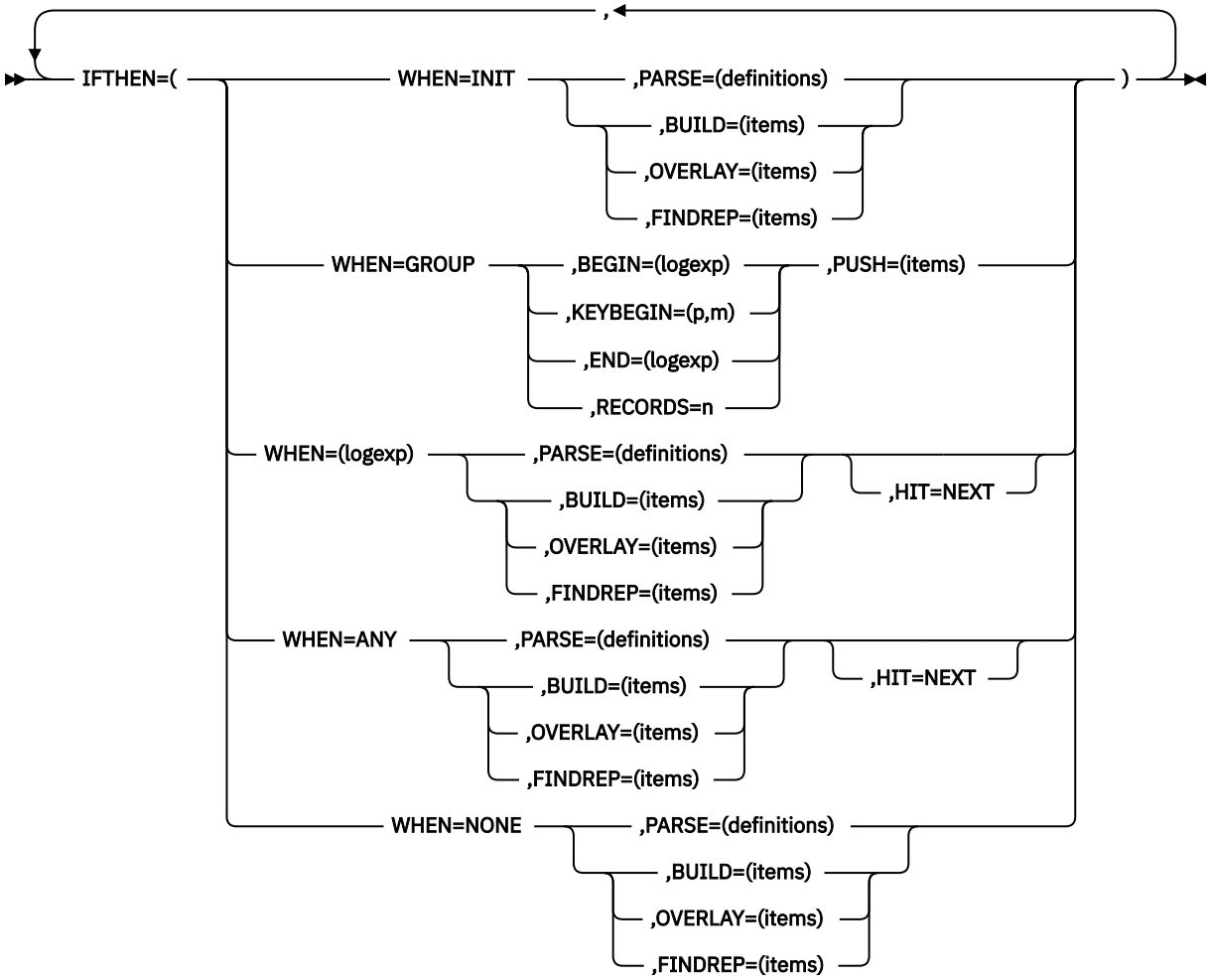
Sample syntax

```
INREC FINDREP=(IN=C'Goodbye',OUT=C'Bye')
```

Default for FINDREP: None; must be specified. See Appendix B, “Specification/override of DFSORT options,” on page 805.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

IFTHEN



IFTHEN clauses allow you to reformat different records in different ways by specifying how build, overlay, find/replace or group operation items are to be applied to records that meet given criteria. IFTHEN clauses let you use simple or complex conditional logic to choose how different record types are reformatted.

If you want to insert, rearrange, or delete fields in the same way for every record, use BUILD or FIELDS rather than IFTHEN. If you want to overlay existing columns in the same way for every record, use OVERLAY rather than IFTHEN. If you want to do find and replace operations in the same way for every record, use FINDREP rather than IFTHEN. Use IFTHEN clauses if you want to insert, rearrange, delete or overlay fields, or perform find/replace operations in different ways for different records., or if you want to perform operations on groups of records.

You can use five types of IFTHEN clauses as follows:

- **WHEN=INIT:** Use one or more WHEN=INIT clauses to apply build, overlay or find/replace items to all of your input records. WHEN=INIT clauses and WHEN=GROUP clauses are processed before any of the other IFTHEN clauses.
- **WHEN=GROUP**: Use one or more WHEN=GROUP clauses to propagate fields, identifiers and sequence numbers within specified groups of records. WHEN=INIT clauses and WHEN=GROUP clauses are processed before any of the other IFTHEN clauses.
- **WHEN=(logexp):** Use one or more WHEN=(logexp) clauses to apply build, overlay or find/replace items to your input records that meet specified criteria. A WHEN=(logexp) clause is satisfied when the logical expression evaluates as true.

- **WHEN=ANY:** Use a WHEN=ANY clause after multiple WHEN=(logexp) clauses to apply additional build, overlay or find/replace items to your input records if they satisfied the criteria for any of the preceding WHEN=(logexp) clauses.
- **WHEN=NONE:** Use one or more WHEN=NONE clauses to apply build, overlay or find/replace items to your input records that did not meet the criteria for any of the WHEN=(logexp) clauses. WHEN=NONE clauses are processed after any of the other IFTHEN clauses. If you do not specify a WHEN=NONE clause, only the WHEN=INIT and WHEN=GROUP changes (if any) are applied to input records that do not meet the criteria for any of the WHEN=(logexp) clauses.

IFTHEN clauses are processed in the following order:

- WHEN=INIT clauses and WHEN=GROUP clauses
- WHEN=(logexp) clauses and WHEN=ANY clauses
- WHEN=NONE clauses

Processing of IFTHEN clauses continues unless one of the following occurs:

- A WHEN=(logexp) or WHEN=ANY clause is satisfied, and HIT=NEXT is not specified.
- There are no more IFTHEN clauses to process. When processing of IFTHEN clauses stops, the IFTHEN record created so far is used as the output record.

Example:

```
  INREC IFTHEN=(WHEN=(12,1,BI,ALL,X'3F'),OVERLAY=(18:C'Yes')),
        IFTHEN=(WHEN=(35,2,PD,EQ,+14),BUILD=(1,40,45,3,HEX),HIT=NEXT),
        IFTHEN=(WHEN=(35,2,PD,GT,+17),BUILD=(1,40,41,5,HEX),HIT=NEXT),
        IFTHEN=(WHEN=ANY,BUILD=(1,55,C'ABC',70:X)),
        IFTHEN=(WHEN=(63,2,CH,EQ,C'AB'),OVERLAY=(18:C'No')),
        IFTHEN=(WHEN=NONE,BUILD=(1,40,51,8,TRAN=LTOU))
```

For this example, the IFTHEN clauses are processed as follows:

- If IFTHEN clause 1 is satisfied, its overlay item is applied and IFTHEN processing stops.
- If IFTHEN clause 1 is not satisfied, its overlay item is not applied and IFTHEN processing continues.
- If IFTHEN clause 2 is satisfied, its build items are applied and IFTHEN processing continues.
- If IFTHEN clause 2 is not satisfied, its build items are not applied and IFTHEN processing continues.
- If IFTHEN clause 3 is satisfied, its build items are applied and IFTHEN processing continues.
- If IFTHEN clause 3 is not satisfied, its build items are not applied and IFTHEN processing continues.
- If IFTHEN clause 4 is satisfied, its build items are applied and IFTHEN processing stops.
- If IFTHEN clause 4 is not satisfied, its build items are not applied and IFTHEN processing continues.
- If IFTHEN clause 5 is satisfied, its overlay item is applied and IFTHEN processing stops.
- If IFTHEN clause 5 is not satisfied, its overlay item is not applied and IFTHEN processing continues.
- If IFTHEN clause 6 is satisfied, its build items are applied and IFTHEN processing stops.
- If IFTHEN clause 6 is not satisfied, its build items are not applied and IFTHEN processing stops.

All of the IFTHEN clauses operate sequentially on an IFTHEN record. The IFTHEN record is created initially from the input record. Each IFTHEN clause tests and changes the IFTHEN record, as appropriate. Thus, **changes made by earlier IFTHEN clauses are "seen" by later IFTHEN clauses.** For example, if you have a 40-byte input record and specify:

```
  INREC IFTHEN=(WHEN=INIT,OVERLAY=(8:8,4,ZD,ADD,+1,TO=ZD,LENGTH=4)),
        IFTHEN=(WHEN=(8,4,ZD,EQ,+27),OVERLAY=(28:C'Yes')),
        IFTHEN=(WHEN=NONE,OVERLAY=(28:C'No'))
```

The WHEN=INIT clause adds 1 to the ZD value and stores it in the IFTHEN record. The WHEN=(8,4,ZD,EQ,+27) clause tests the incremented ZD value in the IFTHEN record rather than the original ZD value in the input record.

The IFTHEN record is adjusted as needed for the records created or changed by the IFTHEN clauses. For fixed-length records, blanks are filled in on the left as needed. For variable-length records, the RDW length is adjusted as needed each time the IFTHEN record is changed.

Missing bytes in specified input fields are replaced with blanks so the padded fields can be processed.

DFSORT sets an appropriate LRECL (or reformatted record length if the INREC record is further modified) for the output records based on the build, overlay, find/replace and group operation items specified by the IFTHEN clauses. However, DFSORT does not analyze the possible results of WHEN=(logexp) conditions when determining an appropriate LRECL. When you use INREC IFTHEN clauses, you can override the INREC LRECL determined by DFSORT with the INREC IFOUTLEN parameter.

If SEQNUM is used in multiple IFTHEN clauses, the sequence number will be incremented for each record that satisfies the IFTHEN clause, that is, a separate SEQNUM counter will be kept for each IFTHEN clause. For example, if your input is:

```
RECORD A 1
RECORD B 1
RECORD B 2
RECORD C 1
RECORD A 2
RECORD C 2
RECORD B 3
RECORD D 1
```

and you specify:

```
  INREC IFTHEN=(WHEN=(8,1,CH,EQ,C'A'),OVERLAY=(15:SEQNUM,4,ZD)),
        IFTHEN=(WHEN=(8,1,CH,EQ,C'B'),OVERLAY=(16:SEQNUM,4,ZD)),
        IFTHEN=(WHEN=NONE,OVERLAY=(17:SEQNUM,4,ZD))
```

your output will be:

```
RECORD A 1    0001
RECORD B 1     0001
RECORD B 2     0002
RECORD C 1      0001
RECORD A 2    0002
RECORD C 2      0002
RECORD B 3     0003
RECORD D 1      0003
```

Separate SEQNUM counters are kept for the 'A' record, for the 'B' record, and for the NONE records.

WHEN=INIT clause

See "WHEN=INIT clause" under OUTFIL IFTHEN for details. Note that / cannot be used to create blank records or new records.

Sample Syntax:

```
  INREC IFTHEN=(WHEN=INIT,
                BUILD=(1,20,21:C'Department',31:3X,21,60)),
        IFTHEN=(WHEN=(5,2,CH,EQ,C'D1'),OVERLAY=(31:8,3)),
        IFTHEN=(WHEN=(5,2,CH,EQ,C'D2'),OVERLAY=(31:12,3))
```

WHEN=GROUP clause

See "WHEN=GROUP clause" under OUTFIL IFTHEN for details.

Sample Syntax:

```
  INREC IFTHEN=(WHEN=GROUP,
       BEGIN=(1,40,SS,EQ,C'J82',OR,1,40,SS,EQ,C'M72'),
       PUSH=(41:ID=5))
```

WHEN=(logexp) clause

See "WHEN=(logexp) clause" under OUTFIL IFTHEN for details. Note that although / can be used to create blank records and new records with OUTFIL, it cannot be used with INREC.

Sample Syntax:

```
 INREC IFTHEN=(WHEN=(1,3,CH,EQ,C'T01',AND,
          18,4,ZD,LE,+2000),OVERLAY=(42:C'Type1 <= 2000'),HIT=NEXT),
       IFTHEN=(WHEN=(1,3,CH,EQ,C'T01',AND,6,1,BI,BO,X'03'),
          BUILD=(1,21,42,13)),
       IFTHEN=(WHEN=(1,3,CH,EQ,C'T01',AND,
          18,4,ZD,GT,+2000),OVERLAY=(42:C'Type1 > 2000 '),HIT=NEXT),
       IFTHEN=(WHEN=(1,3,CH,EQ,C'T01',AND,6,1,BI,BO,X'01'),
          BUILD=(1,25,42,13))
```

WHEN=ANY clause

See "WHEN=ANY clause" under OUTFIL IFTHEN for details. Note that although / can be used to create blank records and new records with OUTFIL, it cannot be used with INREC.

Sample Syntax:

```
INREC IFTHEN=(WHEN=(1,3,SS,EQ,C'T01,T02,T03'),
              BUILD=(C'Group A',X,1,80),HIT=NEXT),
      IFTHEN=(WHEN=(1,3,SS,EQ,C'T04,T05,T06'),
              BUILD=(C'Group B',X,1,80),HIT=NEXT),
      IFTHEN=(WHEN=(1,3,SS,EQ,C'T07,T08,T09,T10'),
              BUILD=(C'Group C',X,1,80),HIT=NEXT),
      IFTHEN=(WHEN=ANY,OVERLAY=(16:C'Group Found'))
```

WHEN=NONE clause

See "WHEN=NONE clause" under OUTFIL IFTHEN for details. Note that although / can be used to create blank records and new records with OUTFIL, it cannot be used with INREC.

Sample Syntax:

```
INREC IFTHEN=(WHEN=INIT,BUILD=(1,20,21:C'Department',31:3X,21,60)),
      IFTHEN=(WHEN=(5,2,CH,EQ,C'D1'),OVERLAY=(31:8,3)),
      IFTHEN=(WHEN=(5,2,CH,EQ,C'D2'),OVERLAY=(31:12,3)),
      IFTHEN=(WHEN=NONE,OVERLAY=(31:C'***'))
```

Default for IFTHEN clauses: None; must be specified. See Appendix B, "Specification/Override of DFSORT Options".

Applicable Functions: See Appendix B, "Specification/Override of DFSORT Options".

IFOUTLEN

Overrides the INREC LRECL (or reformatted record length if the INREC record is further modified) determined by DFSORT from your INREC IFTHEN clauses. DFSORT sets an appropriate LRECL for the output records based on the build, overlay, find/replace and group operation items specified by the IFTHEN clauses. However, DFSORT does not analyze the possible results of WHEN=(logexp) conditions when determining an appropriate INREC LRECL. When you use INREC IFTHEN clauses, you can override the INREC LRECL determined by DFSORT with the INREC IFOUTLEN parameter.

Fixed-length records longer than the IFOUTLEN length are truncated to the IFOUTLEN length. Fixed-length records shorter than the IFOUTLEN are padded with blanks to the IFOUTLEN length. Variable-length records longer than the IFOUTLEN length are truncated to the IFOUTLEN length.

n

specifies the length to use for the INREC LRECL (or for the reformatted record length if the INREC record is further modified) . The value for n must be between 1 and 32767, but must not be larger than the maximum LRECL allowed for the RECFM, and must not conflict with the specified or retrieved LRECL for the fixed-length output data set.

Sample Syntax:

```
INREC IFOUTLEN=70,
  IFTHEN=(WHEN=(5,1,CH,EQ,C'1',AND,8,3,ZD,EQ,+10),
      BUILD=(1,40,C'T01-GROUP-A',65)),
  IFTHEN=(WHEN=(5,1,CH,EQ,C'2',AND,8,3,ZD,EQ,+12),
      BUILD=(1,40,C'T02-GROUP-B',65))
```

Default for IFOUTLEN: The LRECL determined from the IFTHEN clauses.

INREC statement notes

- When INREC is specified, DFSORT reformats the input records after user exit E15 or INCLUDE/OMIT statement processing is finished. Thus, references to fields by your E15 user exit and INCLUDE/OMIT statements are not affected, whereas your SORT, OUTREC, and SUM statements must refer to fields in the reformatted input records. Your E35 user exit must refer to fields in the reformatted output record.
- In general, OUTREC should be used rather than INREC so your SORT and SUM statements can refer to fields in the original input records.
- If you use locale processing for SORT, MERGE, INCLUDE, or OMIT fields, you must not use INREC. Use the OUTREC statement or the OUTFIL statement instead of INREC.
- When you specify INREC, you must be aware of the change in record size and layout of the resulting reformatted input records.
- Performance can be improved if you can significantly reduce the length of your records with INREC. INREC and OUTREC should not be used unless they are actually needed to reformat your records.
- For variable-length records, the first entry in the FIELDS, BUILD, or IFTHEN BUILD parameter must specify or include the unedited 4-byte record descriptor word (RDW), that is, the first field must be 1,4 or 1,m with m greater than 4. DFSORT sets the length of the reformatted record in the RDW.

  If the first field in the data portion of the input record is to appear unedited in the reformatted record immediately following the RDW, the entry in the FIELDS, BUILD, or IFTHEN BUILD parameter can specify both RDW and data field in one (1,m,...). Otherwise, the RDW must be specifically included in the reformatted record (for example, 1,4,1,4,HEX).
- For variable-length records, OVERLAY, IFTHEN OVERLAY or IFTHEN PUSH items must not overlay the RDW in bytes 1-4. You must ensure that 1:, 2:, 3: or 4: is not specified or defaulted for any OVERLAY or PUSH item. Note that the default for the first OVERLAY or PUSH item is 1:, so you must override it.
- If the SORTOUT LRECL is specified or available, DFSORT will use it even if it does not match the reformatted INREC record length; this can cause padding or truncation of the reformatted INREC records, or termination. If the SORTOUT LRECL is not specified or available, DFSORT can automatically use the reformatted INREC record length as the SORTOUT LRECL, when appropriate. See the discussion of the SOLRF and NOSOLRF options in “OPTION control statement” on page 173.

  For VSAM data sets, the maximum record size defined in the cluster is equivalent to the LRECL when processing fixed-length records, and is four bytes less than the LRECL when processing variable-length records. See “VSAM considerations ” on page 15 for more information.
- With FIELDS, BUILD, or IFTHEN BUILD, the variable part of the input record (that part beyond the minimum record length) can be included in the reformatted record, and if included, must be the last part. In this case, a value must be specified for p*n* that is less than or equal to the minimum record length (see L4 of the RECORD control statement) plus 1 byte; m*n* and a*n* must be omitted. For example:

  ```
  INREC FIELDS=(1,8,20C'*',9)
  ```

  With OVERLAY, the variable part of the input record must not be included in the reformatted record.
- If INREC with FIELDS or BUILD and OUTREC with FIELDS and BUILD are specified, either both must specify position-only for the last part, or neither must specify position-only for the last part. For example:

  ```
  INREC BUILD=(1,8,20C'*',9)
  OUTREC BUILD=(1,4,3Z,5)
  ```

or:

```
 INREC FIELDS=(1,40,45,5)
 OUTREC FIELDS=(1,45,C'****')
```

OVERLAY or IFTHEN, and FIELDS or BUILD, can differ with respect to position-only. For example:

```
INREC BUILD=(1,24,32:25)
OUTREC IFTHEN=(WHEN=(8,1,ZD,GT,+5),
  BUILD=(1,24,25:C'Yes',28,10)),
  IFTHEN=(WHEN=NONE,
    BUILD=(1,24,25:C'No ',28,10))
```

or:

```
INREC FIELDS=(1,18,8C'*',23)
OUTREC OVERLAY=(24:C'A')
```

- If the reformatted record includes only the RDW and the variable part of the input record, "null" records containing only an RDW could result.
-
- The input records are reformatted before processing, as specified by INREC. The output records are in the format specified by INREC, unless OUTREC is also specified.
- Fields referenced in INREC statements can overlap each other and control fields or both.
- If input is variable records, the output is also variable. This means that each record is given the correct RDW by DFSORT before output.
-
- If overflow might occur during summation, INREC can be used to create a larger SUM field in the reformatted input record (perhaps resulting in a larger record for sorting or merging) so that overflow does not occur. “Example 5” on page 437 illustrates this technique.
- DFSORT issues a message and terminates if an INREC statement is specified for a tape work data set sort or conventional merge application.
- If SZERO is in effect, -0 is treated as negative and +0 is treated as positive for edited or converted input fields, decimal constants, and the results of arithmetic expressions. If NOSZERO is in effect, -0 and +0 are treated as positive for edited or converted input fields, decimal constants, and the results of arithmetic expressions.
- If SZERO is in effect, -0 compares as less than +0 when numeric fields and constants are used. If NOSZERO is in effect, -0 compares as equal to +0 when numeric fields and constants are used.

  Note: OPTION SZERO or OPTION NOSZERO is ignored for INREC IFTHEN=(WHEN=(logexp),...) unless the OPTION statement is found in a higher source (for example, DFSPARM is a higher source than SYSIN) or before the INREC statement in the same source. For example, NOSZERO will be used in both of the following cases:

  – Case 1:

  ```
  //DFSPARM DD *
    OPTION COPY,NOSZERO
  /*
  //SYSIN DD *
    INREC IFTHEN=(WHEN=(1,2,FS,EQ,+0),OVERLAY=(22:C'Yes')),
          IFTHEN=(WHEN=NONE,OVERLAY=(22:C'No '))
  /*
  ```

  – Case 2:

  ```
  //SYSIN DD *
    OPTION COPY,NOSZERO
    INREC IFTHEN=(WHEN=(1,2,FS,EQ,+0),OVERLAY=(28:C'A')),
          IFTHEN=(WHEN=NONE,OVERLAY=(28:C'B'))
  /*
  ```

Reformatting records before processing — examples

Both INREC and OUTREC can be used to reformat records, either separately or together. The two control statements perform similar functions, although INREC reformats records before sort, merge, or copy processing, whereas OUTREC reformats records after sort, merge, or copy processing. This section shows INREC examples. See “Reformatting records after processing — examples” on page 407 for OUTREC examples.

Example 1

INREC method

```
INCLUDE COND=(5,1,GE,C'M'),FORMAT=CH
INREC FIELDS=(10,3,20,8,33,11,5,1)
SORT FIELDS=(4,8,CH,A,1,3,FI,A)
SUM FIELDS=(17,4,BI)
```

OUTREC method

```
INCLUDE COND=(5,1,GE,C'M'),FORMAT=CH
OUTREC FIELDS=(10,3,20,8,33,11,5,1)
SORT FIELDS=(20,8,CH,A,10,3,FI,A)
SUM FIELDS=(38,4,BI)
```

These examples illustrate how a fixed-length input data set is sorted and reformatted for output. Unnecessary fields are eliminated from the output records using INREC or OUTREC. The SORTIN LRECL is 80.

Records are also included or excluded by means of the INCLUDE statement, and summed by means of the SUM statement.

The reformatted input records are fixed length with a record size of 23 bytes. SOLRF (the IBM-supplied default) is in effect, so unless the SORTOUT LRECL is specified or available, it will automatically be set to the reformatted record length of 23. The reformatted records look as follows after INREC or OUTREC processing:

Position
Contents

1-3
Input positions 10 through 12

4-11
Input positions 20 through 27

12-22
Input positions 33 through 43

23
Input position 5

Identical results are achieved with INREC or OUTREC. However, use of OUTREC makes it easier to code the SORT and SUM statements. In either case, the INCLUDE COND parameters must refer to the fields of the original input records. However, with INREC, the SUM and SORT FIELDS parameters must refer to the fields of the *reformatted* input records, while with OUTREC, the SUM and SORT FIELDS parameters must refer to the fields of the *original* input records.

Example 2

```
INREC FIELDS=(1,35,2Z,36,45)
MERGE FIELDS=(20,4,CH,D,10,3,CH,D),FILES=3
SUM FIELDS=(36,4,BI,40,8,PD)
RECORD TYPE=F,LENGTH=(80,,82)
```

This example illustrates how overflow of a summary field can be prevented when three fixed-length data sets are merged and reformatted for output. The input record size is 80 bytes. To illustrate the use of the RECORD statement, assume that SORTIN and SORTOUT are not present (that is, all input/output is handled by user exits).

The reformatted input records are fixed-length with a record size of 82 bytes (an insignificant increase from the original size of 80 bytes). They look as follows:

Position
Contents

1-35
Input positions 1 through 35

36-37
Binary zeros (to prevent overflow)

38-82
Input positions 36 through 80

The MERGE and SUM statements must refer to the fields of the reformatted input records.

The reformatted output records are identical to the reformatted input records.

Thus, the 2-byte summary field at positions 36 and 37 in the original input records expands to a 4-byte summary field in positions 36 through 39 of the reformatted input/output record *before* merging. This prevents overflow of this summary field.

Restriction: If OUTREC were used instead of INREC, the records would be reformatted *after* merging, and the 2-byte summary field might overflow.

Note: This method of preventing overflow *cannot* be used for negative FI summary fields because padding with zeros rather than ones would change the sign.

Example 3

```
INREC BUILD=(20,4,12,3)
SORT FIELDS=(1,4,D,5,3,D),FORMAT=CH
OUTREC BUILD=(5X,1,4,H,19:1,2,5,3,DATE1(-),80X'FF')
```

This example illustrates how a fixed-length input data set can be sorted and reformatted for output. A more efficient sort is achieved by using INREC before sorting to reduce the input records as much as possible, and using OUTREC after sorting to add padding, the current date and repeated fields. The SORTIN LRECL is 80 bytes.

The reformatted input records are fixed-length, and have a record size of seven bytes (a significant reduction from the original size of 80 bytes). They look as follows:

Position
Contents

1-4
Input positions 20 through 23

5-7
Input positions 12 through 14

The SORT and OUTREC statements must refer to the fields of the reformatted input records.

The reformatted output records are fixed length with a record size of 113 bytes. SOLRF (the IBM-supplied default) is in effect, so unless the SORTOUT LRECL is specified or available, it will automatically be set to the reformatted record length of 113. The reformatted output records look as follows:

Position
Contents

1-5
EBCDIC blanks

6
Binary zero (for H alignment)

7-10
Input positions 20 through 23

11-18
EBCDIC blanks

19-20
Input positions 20 through 21

21-23
Input positions 12 through 14

24-33
The current date in the form C'yyyy-mm-dd'

34-113
Hexadecimal FF's

Thus, the use of INREC and OUTREC allows sorting of 7-byte records rather than 80-byte records, even though the output records are 113 bytes long.

Example 4

```
OPTION COPY,Y2PAST=1985
INREC FIELDS=(SEQNUM,4,BI,
              8,5,ZD,TO=PD,
              31,2,PD,TO=FI,LENGTH=2,
              15,6,Y2TP,
              25,3,CHANGE=(1,C'L92',X'01',C'M72',X'02',C'J42',X'03'),
              NOMATCH=(X'FF'))
```

This example illustrates how a sequence number can be generated, how values in one numeric or date format can be converted to another format, and how a lookup table can be used.

The reformatted input records will look as follows:

Position
Contents

1-4
A binary sequence number that starts at 1 and increments by 1.

5–7
A PD field containing the converted ZD field from input positions 8 through 12.

8–9
An FI field containing the converted PD field from input positions 31 through 32.

10–14
A P'yyyymmdd' date field containing the C'yymmdd' date field from input positions 15-20 transformed according to the specified century window of 1985-2084.

15
A BI field containing X'01', X'02', X'03' or X'FF' as determined by using a lookup table for the input field in positions 25-27.

The SORT statement can now refer to the "sort" field in the reformatted input records. The OUTREC statement is used to restore the records to their original format.

Example 5

```
INREC OVERLAY=(61:21,11,SFF,ADD,41,11,SFF,TO=PD,LENGTH=5)
SORT FIELDS=(61,5,PD,A)
OUTREC OVERLAY=(61:61,5,PD,EDIT=(SIII,IIT.TT),SIGNS=(+,-))
```

This example illustrates how you can use the OVERLAY parameter with INREC and OUTREC to change certain columns in your records without affecting other columns.

Positions 61-65 of the reformatted input records are overlaid with a 5-byte PD value derived from adding the sddd,ddd.dd field at positions 21-31 to the sddd,ddd.dd field at positions 41-51. The records are then sorted by this 5-byte PD field. Positions 61-71 of the reformatted output records are overlaid with an sddd,ddd.dd field derived from the 5-byte PD value. The data before positions 61-71 and after positions 61-71 are not affected

Example 6

```
  OPTION COPY
  INREC IFTHEN=(WHEN=(5,2,CH,EQ,C'GP',AND,2,1,BI,EQ,+1),
                BUILD=(1,6,16,20,C'T1',X'0003',1,7,20C'1')),
        IFTHEN=(WHEN=(5,2,CH,EQ,C'GP',AND,2,1,BI,EQ,+2),
                BUILD=(1,6,45,20,C'T2',X'0008',16,7,20C'2')),
        IFTHEN=(WHEN=(5,2,CH,EQ,C'GP',AND,2,1,BI,EQ,+3),
                BUILD=(1,6,31,20,C'T3',X'0005',25,7,20C'3')),
        IFTHEN=(WHEN=NONE,OVERLAY=(27:C'00',X'FFFF')),
     IFOUTLEN=57
```

This example illustrates how you can use IFTHEN clauses with INREC to reformat different records in different ways. IFOUTLEN=57 is used to set the reformatted record length to 57.

Records with 'GP' in positions 5-6 and X'01' in position 2 are reformatted as follows:

Position
Contents

1-6
Input positions 1-6

7-26
Input positions 16-35

27-28
'T1'

29-30
X'0003'

31-37
Input positions 1-7

38-57
20 '1's

Records with 'GP' in positions 5-6 and X'02' in position 2 are reformatted as follows:

Position
Contents

1-6
Input positions 1-6

7-26
Input positions 45-64

27-28
'T2'

29-30
X'0008'

31-37
Input positions 16-22

38-57
20 '2's

Records with 'GP' in positions 5-6 and X'03' in position 2 are reformatted as follows:

Position
Contents

1-6
Input positions 1-6

7-26
Input positions 31-50

27-28
'T3'

29-30
X'0005'

31-37
Input positions 25-31

38-57
20 '3's

Records without 'GP' in positions 5-6 or without X'01', X'02', or X'03' in position 2 are reformatted as follows:

Position
Contents

1-26
Input positions 1-26

27-28
'00'

29-30
X'FFFF'

31-57
Input positions 31-57

Example 7

```
  INREC OVERLAY=(16:1,15,JFY=(SHIFT=LEFT))
  SORT FIELDS=(16,15,CH,A)
  OUTREC BUILD=(1,15)
```

This example illustrates how you can left-justify characters in an input field so they can be sorted without regard to the leading blanks.

The 15-byte input records might look like this:

```
   CARRIE
       VICKY
          FRANK
      SAM
DAVID
    MARTIN
```

Note that if we sort these records using just this control statement:

```
   SORT FIELDS=(1,15,CH,A)
```

The 15-byte output records are as follows:

```
          FRANK
       VICKY
      SAM
    MARTIN
   CARRIE
DAVID
```

Because of the different number of leading blanks in the input records, we don't get what we want. To fix that, while keeping the leading blanks in the original records, we use the JFY function of INREC to make a left-justified copy of the 15-byte input field at positions 16-30, SORT on it and use OUTREC to remove the left-justified field. With the INREC, SORT and OUTREC control statements shown previously, the output records are:

```
   CARRIE
DAVID
          FRANK
    MARTIN
      SAM
       VICKY
```

If we wanted the output to contain the sorted left-justified fields, we could use these control statements:

```
  INREC BUILD=(1,15,JFY=(SHIFT=LEFT))
  SORT FIELDS=(1,15,CH,A)
```

The output records would then be:

```
CARRIE
DAVID
FRANK
MARTIN
SAM
VICKY
```

Example 8

```
  INREC PARSE=(%00=(ENDBEFR=C',',FIXLEN=11),
               %01=(ENDBEFR=C',',FIXLEN=5),
               %02=(FIXLEN=6)),
        OVERLAY=(31:%00,42:%01,47:%02)
  SORT FIELDS=(31,11,CH,A,42,5,UFF,A,47,6,SFF,D)
  OUTREC BUILD=(1,30)
```

This example illustrates how you can sort FB input records with variable position/length fields, such as comma separated values.

The 30-byte input records might look like this:

```
Marketing,96218,+27365
Development,3807,+1275
Research,7283,+5001
Development,1700,-5316
Research,978,+13562
Development,3807,-158
Research,7283,+5002
Marketing,52,-8736
Development,5781,+2736
Marketing,52,+1603
Research,16072,-2022
```

We want to sort the first field as character ascending, the second field as unsigned numeric ascending and the third field as signed numeric descending.

Note that each record has three variable fields in comma separated value format. The fields do not start and end in the same position in every record and do not have the same length in every record. The first and second fields end with a comma and the third field ends with a blank.

In order to sort variable fields like these, we use the PARSE and OVERLAY functions of INREC to create a fixed parsed copy of each variable field. We use %00 to create an 11-byte fixed parsed field into which we extract the value before the first comma. We use %01 to create a 5-byte fixed parsed field into which we extract the value after the first comma and before the second comma. We use %02 to create a 6-byte fixed parsed field into which we extract the value after the second comma. Then we SORT on the fixed parsed fields. Finally, we use OUTREC to remove the fixed parsed fields.

After the INREC statement is processed, the records look like this:

```
Marketing,96218,+27365        Marketing  96218+27365
Development,3807,+1275        Development3807 +1275
Research,7283,+5001           Research   7283 +5001
Development,1700,-5316        Development1700 -5316
Research,978,+13562           Research   978  +13562
Development,3807,-158         Development3807 -158
Research,7283,+5002           Research   7283 +5002
Marketing,52,-8736            Marketing  52   -8736
Development,5781,+2736        Development5781 +2736
Marketing,52,+1603            Marketing  52   +1603
Research,16072,-2022          Research   16072-2022
```

Since the second fixed parsed field is unsigned and left-justified, we sort it with the UFF format. Since the third fixed parsed field is signed and left-justified, we sort it with the SFF format.

After the SORT and OUTREC statements are processed, the output records look like this:

```
Development,1700,-5316
Development,3807,+1275
Development,3807,-158
Development,5781,+2736
Marketing,52,+1603
Marketing,52,-8736
Marketing,96218,+27365
Research,978,+13562
Research,7283,+5002
Research,7283,+5001
Research,16072,-2022
```

Example 9

```
  INREC PARSE=(%00=(ENDBEFR=C',',FIXLEN=11),
               %01=(ENDBEFR=C',',FIXLEN=5),
               %02=(FIXLEN=6)),
        BUILD=(1,4,5:%00,16:%01,21:%02,27:5)
  SORT FIELDS=(5,11,CH,A,16,5,UFF,A,21,6,SFF,D)
  OUTREC BUILD=(1,4,27)
```

This example illustrates how you can sort VB input records with variable position/length fields, such as comma separated values. This example is very similar to the previous example for FB records, except that with VB records we need to copy the fixed parsed fields after the 4-byte RDW rather than at the end of the records.

The VB input records might look like this:

```
Length|Data
    26|Marketing,96218,+27365
    26|Development,3807,+1275
    23|Research,7283,+5001
    26|Development,1700,-5316
    23|Research,978,+13562
    25|Development,3807,-158
    23|Research,7283,+5002
    22|Marketing,52,-8736
    26|Development,5781,+2736
    22|Marketing,52,+1603
    24|Research,16072,-2022
```

After the INREC statement is processed, the records look like this:

```
Length|Data
    48|Marketing  96218+27365Marketing,96218,+27365
    48|Development3807 +1275 Development,3807,+1275
    45|Research   7283 +5001 Research,7283,+5001
    48|Development1700 -5316 Development,1700,-5316
    45|Research   978  +13562Research,978,+13562
    47|Development3807 -158  Development,3807,-158
    45|Research   7283 +5002 Research,7283,+5002
    44|Marketing  52   -8736 Marketing,52,-8736
    48|Development5781 +2736 Development,5781,+2736
    44|Marketing  52   +1603 Marketing,52,+1603
    46|Research   16072-2022 Research,16072,-2022
```

After the SORT and OUTREC statements are processed, the output records look like this:

```
Length|Data
    26|Development,1700,-5316
    26|Development,3807,+1275
    25|Development,3807,-158
    26|Development,5781,+2736
    22|Marketing,52,+1603
    22|Marketing,52,-8736
    26|Marketing,96218,+27365
    23|Research,978,+13562
    23|Research,7283,+5002
    23|Research,7283,+5001
    24|Research,16072,-2022
```

Example 10

```
  OPTION COPY
  INREC FINDREP=(IN=(X'00',X'FF'),OUT=C'')
```

This example illustrates how you can remove characters from FB or VB records.

The VB input records might look like this in hexadecimal:

```
RDW----|Data
000F0000D1E4D5C5**00**C1D7D9C9D3**FF**
00100000C2C5E3E3E8**00**C4C1C9E2E8**FF**
```

We want to remove each X'00' and X'FF' character. We use IN=(X'00',X'FF') to indicate we want to find each X'00' and X'FF' character, and OUT=C'' (null) to indicate we want to remove it.

The output records look like this:

```
RDW----|Data
000D0000D1E4D5C5C1D7D9C9D3
000E0000C2C5E3E3E8C4C1C9E2E8
```

Note that the X'00' and X'FF' characters have been removed and the RDW length decreased accordingly. For VB input records, FINDREP processing automatically starts at position 5 after the RDW so the X'00' characters in the RDW are not affected.

Example 11

```
  OPTION COPY
  INREC FINDREP=(IN=C'BALANCE',
                 OUT=C'BALANCE 1000',SHIFT=NO,DO=1)
```

This example illustrates how you can find a value in FB or VB records and overlay it with a larger value without shifting other bytes in the records.

The FB input records might look like this:

```
CUSTOMER1 10100
MNTHLY STMT BALANCE 2000
CUSTOMER2 11100
MNTHLY STMT REQUIRES NO MODIFICATION ACCT BALANCE 5000
CUSTOMER3 11111
YOUR INFO ENCLOSED
MNTHLY STMT REQUIRES MODIFICATION ACCT BALANCE 7000
```

We want to replace every instance of 'BALANCE dddd' with 'BALANCE 1000' where dddd can be any value. We use IN=C'BALANCE' to indicate we want to find each instance of 'BALANCE'. We use OUT='BALANCE 1000' to indicate we want to replace it with 'BALANCE 1000'. We use SHIFT=NO to do an overlay, overriding the default of shifting bytes. Since we only have at most one instance of 'BALANCE dddd' in a record, we can use DO=1 to stop processing a record after one replacement of 'BALANCE dddd' with 'BALANCE 1000'. In this case, DO=1 is more efficient than the default of continuing to look for more

instances of 'BALANCE dddd' after the first instance. We would get the same result without DO=1, just less efficiently

The output records look like this:

```
CUSTOMER1 10100
MNTHLY STMT BALANCE 1000
CUSTOMER2 11100
MNTHLY STMT REQUIRES NO MODIFICATION ACCT BALANCE 1000
CUSTOMER3 11111
YOUR INFO ENCLOSED
MNTHLY STMT REQUIRES MODIFICATION ACCT BALANCE 1000
```

Example 12

```
  INREC IFTHEN=(WHEN=GROUP,BEGIN=(2,4,CH,EQ,C'RPT.'),
    PUSH=(31:6,8))
  OPTION EQUALS
  SORT FIELDS=(31,8,CH,A)
  OUTFIL INCLUDE=(31,8,CH,EQ,C'FRANK',OR,
     31,8,CH,EQ,C'SRIHARI'),BUILD=(1,30)
```

This example illustrates how you can SORT and INCLUDE groups of FB records depending on a value in the first record of each group. We propagate the value in the first record of the group to every record of the group, SORT and INCLUDE on the value, and then remove it.

The 30-byte FBA input records might look like this:

```
1RPT.SRIHARI
 LINE  1 FOR REPORT 1
 LINE  2 FOR REPORT 1
 ...
1RPT.VICKY
 LINE  1 FOR REPORT 2
 LINE  2 FOR REPORT 2
 ...
1RPT.FRANK
 LINE  1 FOR REPORT 3
 LINE  2 FOR REPORT 3
 ...
1RPT.DAVID
 LINE  1 FOR REPORT 4
 LINE  2 FOR REPORT 4
 ...
```

Each report starts with 'RPT.reptname' in positions 2-13. In the output data set we only want to include records for reports with specific reptname values, and the reptname values we want can change from run to run. We also want to sort by the reptname values in ascending order. For this example, let's say we just want the SRIHARI and FRANK reports.

We use an IFTHEN WHEN=GROUP clause to propagate the reptname value to each record of the group. BEGIN indicates a group starts with 'RPT.' in positions 2-5. PUSH overlays the reptname value from the first record of the group (the 'RPT.reptname' record) at positions 31-38 (after the end of the record) in each record of the group including the first. After the IFTHEN GROUP clause is executed, the intermediate records look like this:

```
1RPT.SRIHARI                 SRIHARI
 LINE  1 FOR REPORT 1        SRIHARI
 LINE  2 FOR REPORT 1        SRIHARI
 ...                         SRIHARI
1RPT.VICKY                   VICKY
 LINE  1 FOR REPORT 2        VICKY
 LINE  2 FOR REPORT 2        VICKY
 ...                         VICKY
1RPT.FRANK                   FRANK
 LINE  1 FOR REPORT 3        FRANK
 LINE  2 FOR REPORT 3        FRANK
 ...                         FRANK
1RPT.DAVID                   DAVID
 LINE  1 FOR REPORT 4        DAVID
```

```
 LINE  2 FOR REPORT 4         DAVID
 ...                          DAVID
```

Note that the records of each group have the reptname value from the first record of that group in positions 31-38.

We use a SORT statement to sort ascending on the reptname in positions 31-38. We use the EQUALS option to ensure that records in the same group (that is, with the same reptname value) are kept in their original order. After the SORT statement is executed, the intermediate records look like this:

```
1RPT.DAVID                    DAVID
 LINE  1 FOR REPORT 4         DAVID
 LINE  2 FOR REPORT 4         DAVID
 ...                          DAVID
1RPT.FRANK                    FRANK
 LINE  1 FOR REPORT 3         FRANK
 LINE  2 FOR REPORT 3         FRANK
 ...                          FRANK
1RPT.SRIHARI                  SRIHARI
 LINE  1 FOR REPORT 1         SRIHARI
 LINE  2 FOR REPORT 1         SRIHARI
 ...                          SRIHARI
1RPT.VICKY                    VICKY
 LINE  1 FOR REPORT 2         VICKY
 LINE  2 FOR REPORT 2         VICKY
 ...                          VICKY
```

We use an OUTFIL statement to only INCLUDE the records with a a reptname of FRANK or SRIHARI in positions 31-38, and to remove the reptname from positions 31-38 so the included output records will be identical to the input records. After the OUTFIL statement is executed, the final output records look like this:

```
1RPT.FRANK
 LINE  1 FOR REPORT 3
 LINE  2 FOR REPORT 3
 ...
1RPT.SRIHARI
 LINE  1 FOR REPORT 1
 LINE  2 FOR REPORT 1
 ...
```

Example 13

```
  OPTION COPY,Y2PAST=1996
  INREC BUILD=(1,6,Y2W,TOJUL=Y4T,X,
     1,6,Y2W,WEEKDAY=CHAR3,X,
     9,7,Y4T,TOGREG=Y4T(/),X,
     9,7,Y4T,WEEKDAY=DIGIT1)
```

This example illustrates how to convert an mmddyy date to a ccyyddd date and a 3-character weekday string, and how to convert a ccyyddd date to a ccyy/mm/dd date and 1-digit weekday string.

The input records might be as follows:

```
120409  1999014
051895  2003235
999999  0000000
013099  1992343
```

The output records would be as follows:

```
2009338 FRI 1999/01/14 5
2095138 WED 2003/08/23 7
9999999 999 0000/00/00 0
1999030 SAT 1992/12/08 3
```

The Y2PAST=1996 option sets the century window to 1996-2095. The century window is used to transform yy in the Y2W field to ccyy.

Note that date conversion is not performed for the special indicators (all 9s and all 0s); the special indicator is just used appropriately for the output date field.

Example 14

```
  OPTION COPY
  INREC OVERLAY=(11:1,8,Y4T,ADDDAYS,+50,TOGREG=Y4T,
                21:1,8,Y4T,SUBMONS,+7,TOGREG=Y4T,
                31:1,8,Y4T,ADDYEARS,+2,TOGREG=Y4T)
  OUTFIL REMOVECC,
   HEADER1=('Input     +50 days  -7 months +2 years')
```

This example illustrates how you can add and subtract days, months and years from date fields.

The SORTIN data set has these input records with a C'ccyymmdd' date field in positions 1-8:

```
20070305
20071213
20080219
20080901
20091122
20090115
20100915
20100630
99999999
```

The INREC statement performs three different date field arithmetic operations on the input date field. It adds 50 days, subtracts 7 months and adds 2 years. We use 1,8,Y4T for the input field to match the C'ccyymmdd' input date, and we use TOGREG=Y4T to give us a C'ccyymmdd' output date. The OUTFIL statement adds headings.

SORTOUT will have these records:

```
Input     +50 days  -7 months +2 years
20070305  20070424  20060805  20090305
20071213  20080201  20070513  20091213
20080219  20080409  20070719  20100219
20080901  20081021  20080201  20100901
20091122  20100111  20090422  20111122
20090115  20090306  20080615  20110115
20100915  20101104  20100215  20120915
20100630  20100819  20091130  20120630
99999999  99999999  99999999  99999999
```

Note that the '99999999' input value is treated as a special indicator for output.

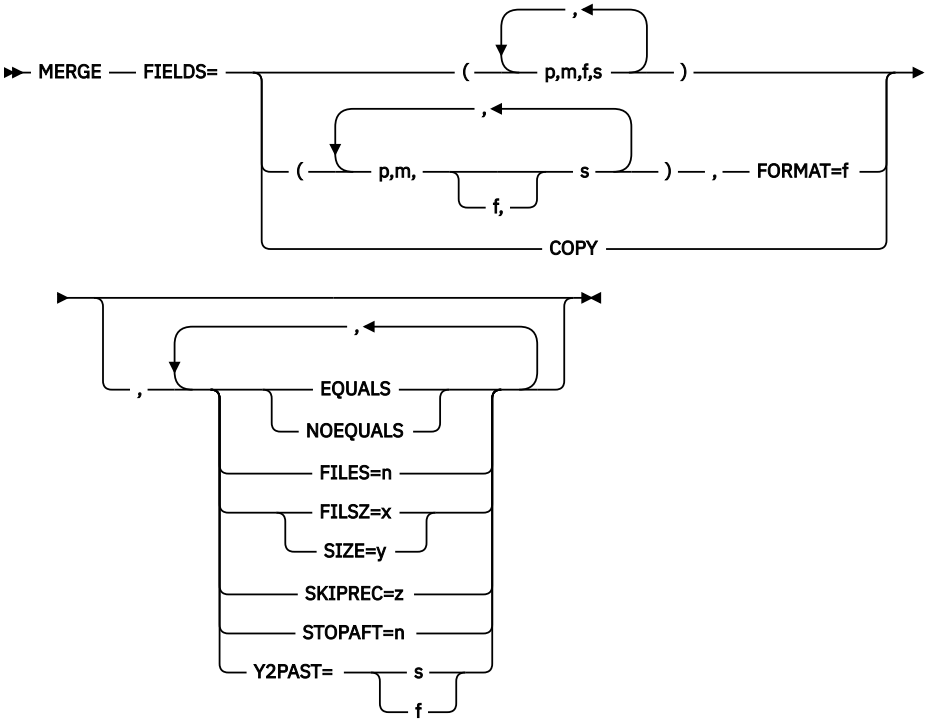
JOINKEYS control statement

Used only for a JOINKEYS application. See Chapter 4, “Using a JOINKEYS application for joining two files,” on page 439 for details.

JOIN control statement

Used only for a JOINKEYS application. See Chapter 4, “Using a JOINKEYS application for joining two files,” on page 439for details.

MERGE control statement



The MERGE control statement must be used when a merge operation is to be performed; this statement describes the control fields in the input records on which the input data sets have previously been sorted.

A MERGE statement can also be used to specify a copy application. User labels will not be copied to the output data sets.

You can merge up to 100 data sets with Blockset merge or up to 16 data sets with Conventional merge. If Blockset merge is not selected, you can use a SORTDIAG DD statement to force message ICE800I, which gives a code indicating why Blockset could not be used.

The way in which DFSORT processes short MERGE control fields depends on the setting for VLSHRT/NOVLSHRT. A short field is one where the variable-length record is too short to contain the entire field, that is, the field extends beyond the record. For details about merging short records, see the discussion of the VLSHRT and NOVLSHRT options in “OPTION control statement” on page 173.

The options available on the MERGE statement can be specified in other sources as well. A table showing all possible sources for these options and the order of override are given in Appendix B, “Specification/override of DFSORT options,” on page 805. When an option can be specified on either the MERGE or OPTION statement, it is preferable to specify it on the OPTION statement.

DFSORT accepts but does not process the following MERGE operands: WORK=value and ORDER=value.

DFSORT's collating behavior can be modified according to your cultural environment. The cultural environment is established by selecting the active locale. The active locale's collating rules affect MERGE processing as follows:

- DFSORT produces merged records for output according to the collating rules defined in the active locale. This provides merging for single- or multi-byte character data, based on defined collating rules that retain the cultural and local characteristics of a language.

If locale processing is to be used, the active locale will only be used to process character (CH) control fields.

For more information on locale processing, see “Cultural environment considerations ” on page 6 or LOCALE in “OPTION control statement” on page 173.

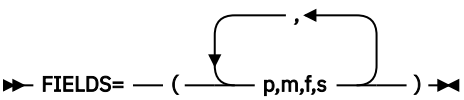
Note: For a merge application, records deleted during an E35 exit routine are not sequence checked. If you use an E35 exit routine without an output data set, sequence checking is not performed at the time the records are passed to the E35 user exit; therefore, you must ensure that input records are in correct sequence.

DFSORT's Unicode collation behavior can be modified according to your cultural environment. The cultural environment is established by selecting the active collation version. The active collation version rules affects MERGE processing as follows:

- DFSORT produces merged records for output according to the collating version rules defined in the active collation version. This provides sorting for single-or multi-byte character data based on defined collating version rules that retain the cultural and local characteristics of a language.

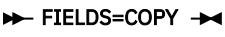
See "Unicode Environment Considerations" on page 6 or COLLKEY in "OPTION control statement" on page 173.

FIELDS



Is written exactly the same way for a merge as it is for a sort. The meanings of p, m, f, and s are described in the discussion of the SORT statement. The defaults for this and the following parameters are also given there. See "SORT control statement" on page 423.

FIELDS=COPY

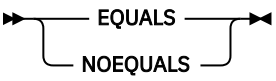


See the discussion of the COPY option on the OPTION statement, in "OPTION control statement" on page 173.

FORMAT=f

See the discussion of the FORMAT option in "SORT control statement" on page 423. Used the same way for a merge as for a sort.

EQUALS or NOEQUALS



See the discussion of these options on the OPTION statement, in "OPTION control statement" on page 173.

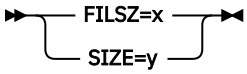
FILES=n

Specifies the number of input files for a merge when input is supplied through the E32 exit.

Default: None; must be specified when an E32 exit is used.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

FILSZ or SIZE



See the discussion of these options on the OPTION statement, in "OPTION control statement" on page 173.

SKIPREC

See the discussion of this option on the OPTION statement, in "OPTION control statement" on page 173.

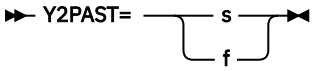
Note: SKIPREC is used for a copy or sort application, but is not used for a merge application.

STOPAFT

See the discussion of this option on the OPTION statement, in "OPTION control statement" on page 173.

Note: STOPAFT is used for a copy or sort application, but is not used for a merge application.

Y2PAST



See the discussion of this option on the OPTION statement, in "OPTION control statement" on page 173.

Note: CENTURY=value and CENTWIN=value can be used instead of Y2PAST=value.

Specifying a MERGE or COPY—examples

Example 1

```
MERGE FIELDS=(2,5,CH,A),FILSZ=29483
```

FIELDS
The control field begins on byte 2 of each record in the input data sets. The field is 5 bytes long and contains character (EBCDIC) data that has been presorted in ascending order.

FILSZ
The input data sets contain exactly 29483 records.

Example 2

```
MERGE FIELDS=(3,8,ZD,E,40,6,CH,D)
```

FIELDS
The major control field begins on byte 3 of each record, is 8 bytes long, and contains zoned decimal data that is modified by your routine before the merge examines it.

The second control field begins on byte 40, is 6 bytes long, and contains character data in descending order.

Example 3

```
MERGE FIELDS=(25,4,A,48,8,A),FORMAT=ZD
```

FIELDS
The major control field begins on byte 25 of each record, is 4 bytes long, and contains zoned decimal data that has been placed in ascending sequence.

The second control field begins on byte 48, is 8 bytes long, is also in zoned decimal format, and is also in ascending sequence. The FORMAT parameter is used to indicate that both fields have ZD format.

Example 4

```
MERGE FIELDS=COPY
```

FIELDS
The input data set is copied to output. No merge takes place.

Example 5: Merging UTF8 data

```
MERGE FIELDS=(28,12,UTF8,A)
```

FIELDS
The control field begins on the 28th byte of each record in the input data set, is 12 bytes long, and contains 8 bit encoding Unicode Transformation Format (UTF8) data. It is to be merged in ascending order.

Example 6: Merging UTF16 data

```
MERGE FIELDS=(25,400,A,600,100,D),FORMAT=UTF16
```

FIELDS
The first three values describe the major control field. It begins on byte 25 of each record, is 400 bytes long, and contains a 16 bit encoding Unicode Transformation Format (UTF16) data, and is to be merged in ascending order.

The next three values describe the second control field. It begins on byte 600, is 100 bytes long, contains a 16 bit encoding Unicode Transformation Format (UTF16) data, and is to be merged in descending order.

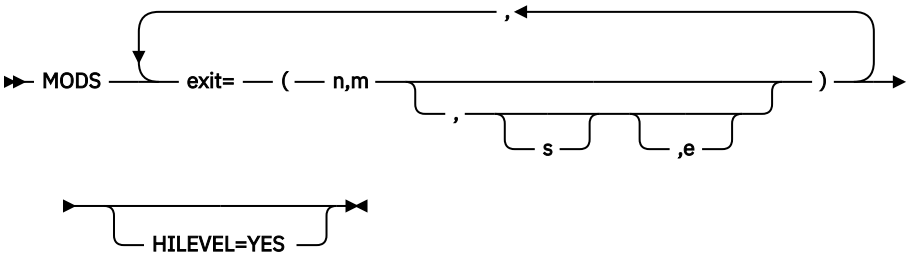
FORMAT
FORMAT=UTF16 is used to supply UTF16 format for the p,m,s fields and is equivalent to specifying p,m,UTF16,s for these fields.

Example 7: Merging UTF32 data

```
MERGE FIELDS=(75,44,UTF32,A)
```

FIELDS
The control field begins on byte 75 of each record in the input data set, is 44 bytes long, and contains 32 bit encoding Unicode Transformation Format (UTF32) data. It is to be merged in ascending order.

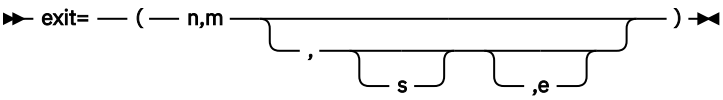
MODS control statement



The MODS statement is needed only when DFSORT passes control to your routines at user exits. The MODS statement associates user routines with specific DFSORT exits and provides DFSORT with descriptions of these routines. For details about DFSORT user exits and how user routines can be used, see Chapter 5, “Using your own user exit routines,” on page 467.

To use one of the user exits, you substitute its three-character name (for example, E31) for the word *exit* in the MODS statement format shown previously in this section. You can specify any valid user exit, except E32. (E32 can be used only in a merge operation invoked from a program; its address must be passed in a parameter list.)

exit



The values that follow the exit parameter describe the user routine. These values are:

n

specifies the name of your routine (member name if your routine is in a library). You can use any valid operating system name for your routine. This allows you to keep several alternative routines with different names in the same library.

m

specifies the number of bytes of main storage your routine uses.Include storage obtained (via GETMAIN) by your routine (or, for example, by OPEN) and the storage required to load the COBOL library subroutines.

s

specifies either the name of the DD statement in your DFSORT job step that defines the library in which your routine is located or SYSIN if your routine is in the input stream. SYSIN is not valid for copy processing.

If a value is not specified for s, DFSORT uses the following search order to find the library in which your routine is located:

1. The libraries identified by the STEPLIB DD statement
2. The libraries identified by the JOBLIB DD statement (if there is no STEPLIB DD statement)
3. The link library.

e

specifies the linkage editor requirements of your routine or indicates that your routine is written in COBOL. The following values are allowed:

N

specifies that your routine has already been bound or link-edited and can be used in the DFSORT run without further binding or link-editing. This is the default for e. N (specified or defaulted) can

be overridden by the EXEC PARM parameters 'E15=COB' and 'E35=COB' or by the HILEVEL=YES parameter.

N64
specifies that your routine will use a 64-bit exit parameter list, has already been bound or link-edited and can be used in the DFSORT run without further binding or link-editing.

Note: N64 is only allowed when DFSORT is invoked from a program with the 64-bit invocation parameter list.

C
specifies that your E15 or E35 routine is written in COBOL. If you code C for any other exit, it is ignored, and N is assumed. Your COBOL-written routine must already have been link-edited. The COBEXIT option of the OPTION statement specifies the library for the COBOL exits.

T
specifies that your routine must be bound or link-edited together with other routines to be used in the same phase (for example, E1n routines) of DFSORT. See "Dynamically binding or link-editing user exit routines" on page 476 for additional information. This value is not valid for copy processing.

S
specifies that your routine requires binding or link-editing but that it must be bound or link-edited separately from the other routines (for example, E3n routines) to be used in a particular phase of DFSORT. E11 and E31 exit routines are the only routines eligible for separate binding or link-editing. See "Dynamically binding or link-editing user exit routines" on page 476 for additional information. This value is not valid for copy processing.

If you do not specify a value for e, N is assumed.

HILEVEL=YES

specifies that:

- if an E15 routine is identified on the MODS statement, it is written in COBOL
- if an E35 routine is identified on the MODS statement, it is written in COBOL.

If you identify an E15 routine and an E35 routine on the MODS statement, specify HILEVEL=YES only if both routines are written in COBOL. If you do not identify an E15 or E35 routine on the MODS statement, HILEVEL=YES is ignored.

Note: COBOL=YES can be used instead of HILEVEL=YES.

Note:

1. The s parameter must be the same or omitted for each routine with N, N64 or C for the e parameter (library concatenation is allowed). These routines cannot be placed in SYSIN. Each such routine must be a load module.
2. Each routine for which T or S is specified for the e parameter can be placed in any library or in SYSIN; they do not all have to be in the same library or SYSIN (but can be). Some routines can even be in different libraries (or the same library) and the rest can be in SYSIN. Each such routine, if in a library, can be either an object deck or a load module; if in SYSIN, it must be an object deck.
3. If the same routine is used in both input (that is, E1n routines) and output (that is, E3n routines) DFSORT program phases, a separate copy of the routine must be provided for each exit.
4. HILEVEL=YES can be used instead of C as the fourth parameter, to indicate that an E15 or E35 routine is written in COBOL. In this case, if T is specified as the fourth parameter for E15 or E35, DFSORT terminates. If you identify an E15 routine and an E35 routine on the MODS statement, specify HILEVEL=YES only if both routines are written in COBOL.

5. EXEC PARM parameter E15=COB can be used instead of C as the fourth parameter, to indicate that an E15 is written in COBOL. In this case, if T is specified as the fourth parameter for E15, DFSORT terminates.
6. EXEC PARM parameter E35=COB can be used instead of C as the fourth parameter, to indicate that an E35 is written in COBOL. In this case, if T is specified as the fourth parameter for E35, DFSORT terminates.
7. If HILEVEL=YES, E15=COB, or E35=COB is used instead of C as the fourth parameter, to indicate that an exit is written in COBOL, the fourth parameter for that exit must be specified as N or not specified.
8. If a COBOL E15 or E35 is specified for a conventional merge or tape work data set sort, DFSORT terminates.
9. exit=(n,m) can be used to omit both the s and e parameters.
10. exit=(n,m,,e) can be used to omit the s parameter, but not the parameter.
11. The s parameter must be specified for a conventional merge or tape work data set sort, or when S or T is specified for the e parameter.

Default: None; must be specified if you use exit routines. N is the default for the fourth parameter.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

For information on user exit routines in SYSIN, see “System DD statements” on page 63.

For details on how to design your routines, refer to “Summary of rules for user exit routines” on page 474.

When you are preparing your MODS statement,remember that DFSORT must know the amount of main storage your routine needs so that it can allocate main storage properly for its own use. If you do not know the exact number of bytes your program requires (including requirements for system services), make a slightly high estimate. The value of m in the MODS statement is written the same way whether it is an exact figure or an estimate: you do not precede the value by E for an estimate.

Identifying user exit routines—examples

Example 1

```
MODS E15=(ADDREC,552,MODLIB),E35=(ALTREC,11032,MODLIB)
```

E15
At exit E15, DFSORT transfers control to your own routine. Your routine is in the library defined by a job control statement with the ddname MODLIB. Its member name is ADDREC and uses 552 bytes.

E35
At exit E35, DFSORT transfers control to your routine. Your routine is in the library defined by the job control statement with the ddname MODLIB. Its member name is ALTREC and will use 11032 bytes.

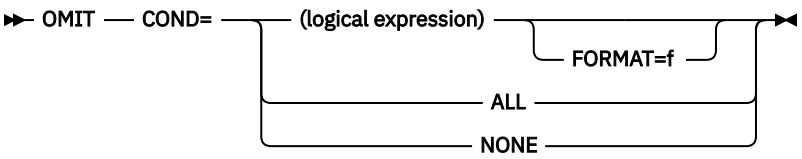
Example 2

```
MODS   E15=(COBOLE15,7000,,C),
       E35=(COBOLE35,7000,EXITC,C)
```

E15
At exit E15, DFSORT transfers control to your own routine. Your routine is written in COBOL and is in the STEPLIB/JOBLIB or link libraries. Its member name is COBOLE15 and it uses 7000 bytes.

E35
At exit E35, DFSORT transfers control to your routine. Your routine is written in COBOL and is in the library defined by the job control statement with the ddname EXITC. Its member name is COBOLE35 and it uses 7000 bytes.

OMIT control statement

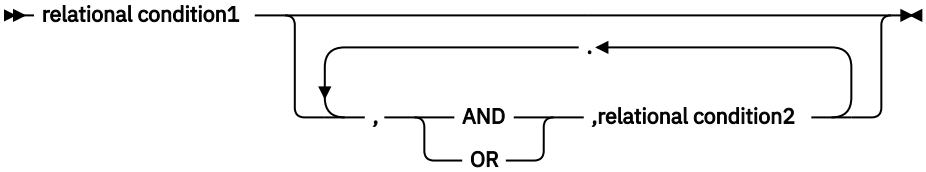


Use an OMIT statement if you do not want all of the input records to appear in the output data sets. The OMIT statement selects the records you do *not* want to include.

You can specify either an INCLUDE statement or an OMIT statement in the same DFSORT run, but not both.

The way in which DFSORT processes short INCLUDE/OMIT compare fields depends on the settings for VLSCMP/NOVLSCMP and VLSHRT/NOVLSHRT. A short field is one where the variable-length record is too short to contain the entire field, that is, the field extends beyond the record. For details about including or omitting short records, see the discussion of the VLSCMP and NOVLSCMP options in "OPTION control statement" on page 173.

A logical expression is one or more relational conditions logically combined, based on fields in the input record, and can be represented at a high level as follows:



If the logical expression is true for a given record, the record is omitted from the output data set.

Five types of relational conditions can be used as follows:

1.
2. **Comparisons:**

   Compare two compare fields or a compare field and a decimal, hexadecimal, character or current, future, or past date constant.

   For example, you can compare the first 6 bytes of each record with its last 6 bytes, and omit those records in which those fields are identical. Or you can compare a date field with today's date, yesterday's date, or tomorrow's date and omit those records accordingly.
3. **Substring Comparison Tests:**

   Search for a constant within a field value or a field value within a constant.

   For example, you can search the value in a 6-byte field for the character constant C'OK', and omit those records for which C'OK' is found somewhere in the field. Or you can search the character constant C'J69,L92,J82' for the value in a 3-byte field, and omit those records for which C'J69', C'L92', or C'J82' appears in the field.
4. **Bit Logic Tests:**

   Test the state (on or off) of selected bits in a binary field using a bit or hexadecimal mask or a bit constant.

   For example, you can omit those records which have bits 0 and 2 on in a 1-byte field. Or you can omit those records which have bits 3 and 12 on and bits 6 and 8 off in a 2-byte field.
5. **Date Comparisons:**

   Compare a two-digit year date field to a two-digit year date constant, the current two-digit year date or another two-digit year date field, using the century window in effect.

For example, you can omit only those records for which a Z'yymm' date field is between January 1996 and March 2005. Or you can omit only those records for which a P'dddyy' field is less than another P'dddyy' field.

6. **Numeric Tests:**

   Test a field for numerics or non-numerics in character, zoned decimal, or packed decimal format.

   For example, you can omit only those records in which a 5-byte field contains only '0'-'9' characters (that is, numerics). Or you can omit only those records in which a 9-byte field contains invalid ZD numeric data (that is, non-numerics). Or you can omit only those records in which a 12-byte field contains valid PD numeric data (that is, numerics).

7. **Alphanumeric Tests:**

   Test a field for alphanumerics or non-alphanumerics in character format. Various combinations of uppercase characters (A-Z), lowercase characters (a-z) and numeric characters (0-9) can be used.

   For example, you can omit only those records in which a 10-byte field contains only 'A'-'Z' characters (that is, uppercase characters) or '0'-'9' characters (that is, numeric characters). Or you can omit only those records in which a 20-byte field contains characters other than 'a'-'z' (that is, lowercase characters).

For complete details on the parameters of the OMIT control statement, see "INCLUDE control statement" on page 91.

The OMIT control statement differs from the OMIT parameter of the OUTFIL statement in the following ways:

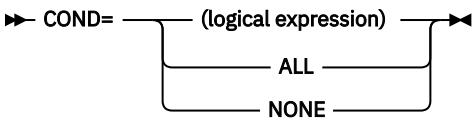
- The OMIT statement applies to all input records; the OMIT parameter applies only to the OUTFIL input records for its OUTFIL group.
- FORMAT=f can be specified with the OMIT statement but not with the OMIT parameter. Thus, you can use FORMAT=f and p,m or p,m,f fields with the OMIT statement, but you must only use p,m,f fields with the OMIT parameter. For example:

```
OMIT FORMAT=BI,
  COND=(5,4,LT,11,4,OR,21,4,EQ,31,4,OR,
     61,20,SS,EQ,C'FLY')

OUTFIL OMIT=(5,4,BI,LT,11,4,BI,OR,21,4,BI,EQ,31,4,BI,OR,
       61,20,SS,EQ,C'FLY')
```

- D2 format can be specified with the OMIT statement but not with the OMIT parameter.

See "OUTFIL control statements" on page 221 for more details on the OUTFIL OMIT parameter.

COND



logical expression
specifies one or more relational conditions logically combined, based on fields in the input record. If the logical expression is true for a given record, the record is omitted from the output data sets.

ALL
specifies that all of the input records are to be omitted from the output data sets.

NONE
specifies that none of the input records are to be omitted from the output data sets.

Default: NONE. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

FORMAT

For details on this parameter, see “INCLUDE control statement” on page 91.

Omitting records from the output data set—example

Example

```
OMIT COND=(27,1,CH,EQ,C'D',&,
           (22,2,BI,SOME,X'C008',|,
            28,1,BI,EQ,B'.1....01'))
```

This statement omits records in which:

- Byte 27 contains D

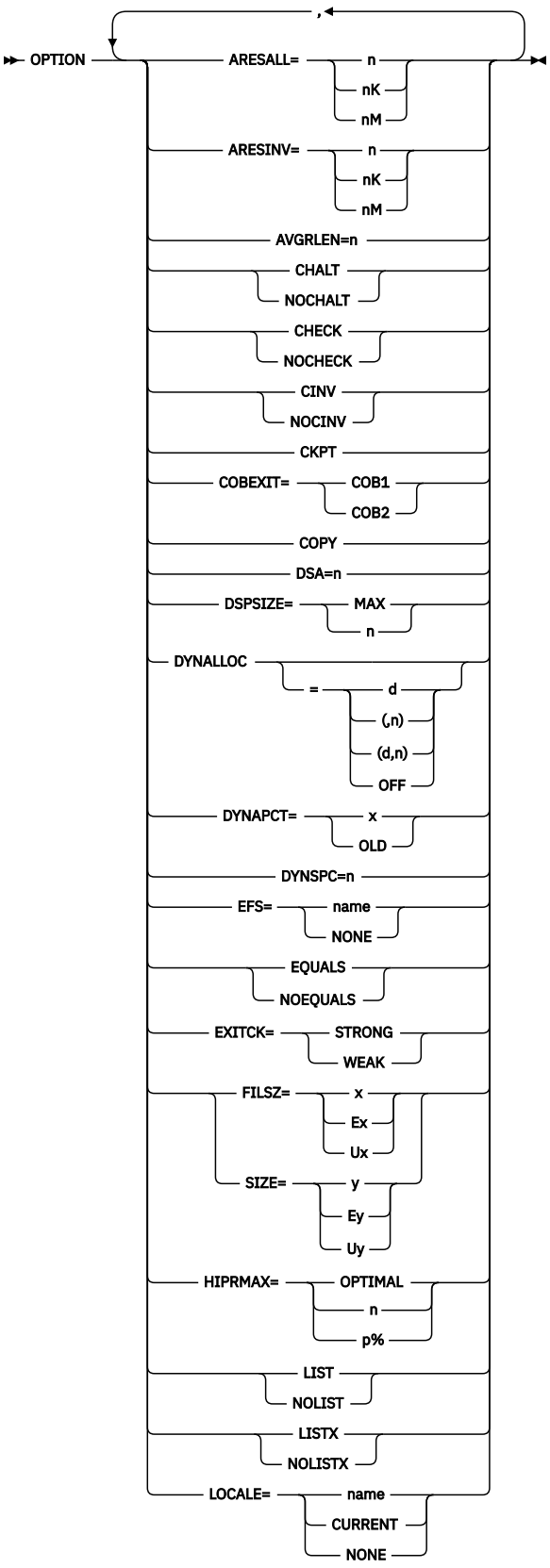
  AND

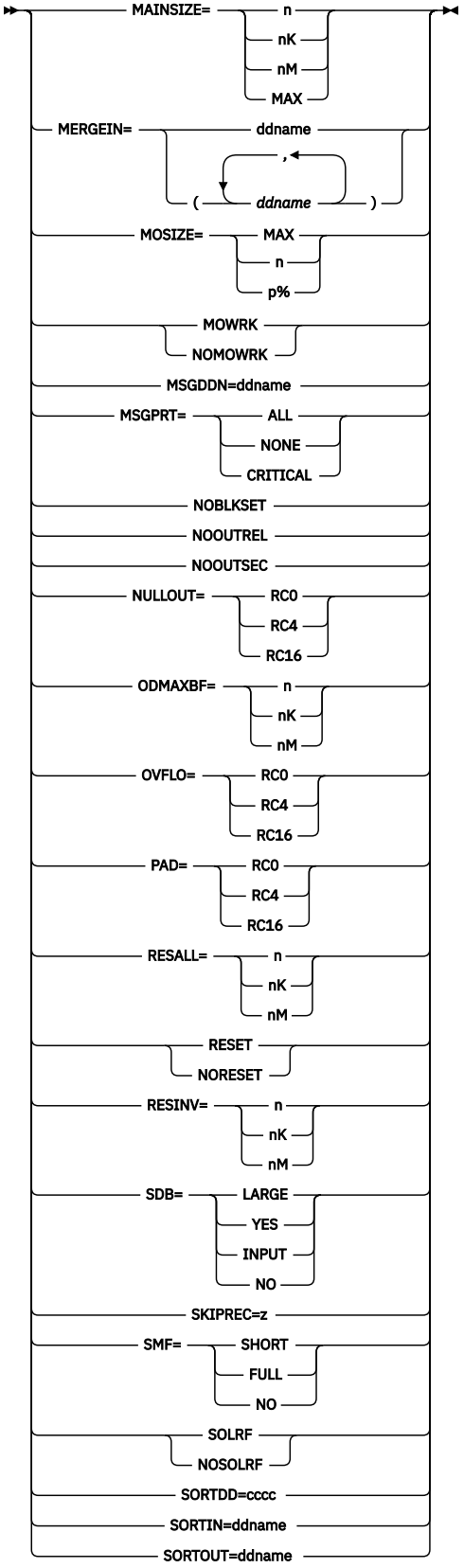
- Bytes 22 through 23 have some, but not all of bits 0, 1 and 12 on OR byte 28 is equal to the specified pattern of bit 1 on, bit 6 off and bit 7 on.

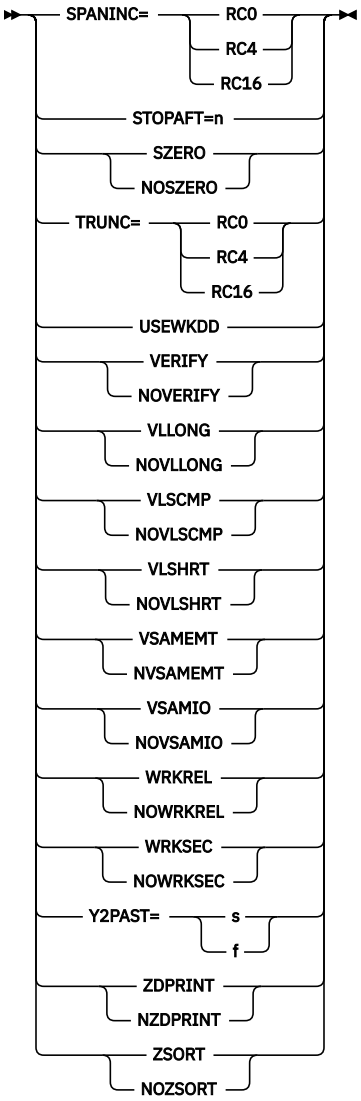
Note that the AND and OR operators can be written with the AND and OR signs, and that parentheses are used to change the order in which AND and OR are evaluated.

For additional examples of logical expressions, see “INCLUDE control statement” on page 91.

OPTION control statement




MAINSIZE=
n
nK
nM
MAX
MERGEIN=
ddname
,
ddname
MOSIZE=
MAX
n
p%
MOWRK
NOMOWRK
MSGDDN=ddname
MSGPRT=
ALL
NONE
CRITICAL
NOBLKSET
NOOUTREL
NOOUTSEC
NULLOUT=
RC0
RC4
RC16
ODMAXBF=
n
nK
nM
OVFLO=
RC0
RC4
RC16
PAD=
RC0
RC4
RC16
RESALL=
n
nK
nM
RESET
NORESET
RESINV=
n
nK
nM
SDB=
LARGE
YES
INPUT
NO
SKIPREC=z
SMF=
SHORT
FULL
NO
SOLRF
NOSOLRF
SORTDD=cccc
SORTIN=ddname
SORTOUT=ddname



Note for Syntax Diagram: The keywords EFS, LIST, NOLIST, LISTX, NOLISTX, MSGDDN, MSGPRT, SMF, SORTDD, SORTIN, SORTOUT, and USEWKDD are used only when they are specified on the OPTION control statement passed by an extended parameter list or when specified in the DFSPARM data set. If they are specified on an OPTION statement read from the SYSIN or SORTCNTL data set, the keyword is recognized, but the parameters are ignored.

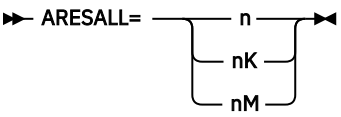
The OPTION control statement allows you to override some of the options available at installation time (such as EQUALS and CHECK) and to supply other optional information (such as DYNALLOC, COPY, and SKIPREC).

Some of the options available on the OPTION statement are also available on the SORT or MERGE statement (such as FILSZ and SIZE). It is preferable to specify these options on the OPTION statement. For override rules, see Appendix B, “Specification/override of DFSORT options,” on page 805.

Details of aliases for OPTION statement options are given under the description of individual options. Table 42 on page 216 summarizes the available aliases.

DFSORT accepts but does not process the following OPTION operands: ALGQ, APP, APPEND, BIAS=value, BLKSET, CASCADE, DIAG, ERASE, EXCPVR=value, MAXPFIX=value, NEW, NEWFILE, NODIAG, NOERASE, NOINC, NOSTIMER, NOSWAP, OPT=value, REP, REPLACE, ROUTE=value, WRKADR=value, WRKDEV=value, and WRKSIZ=value.

ARESALL



Temporarily overrides the ARESALL installation option, which specifies the number of bytes to be reserved above virtual for system use.

ARESALL applies only to the amount of main storage above virtual. This option is normally not needed because of the large amount of storage available above 16MB virtual (the default for ARESALL is 0 bytes). The RESALL option applies to the amount of main storage below 16MB virtual.

n
specifies that n bytes of storage are to be reserved.

Limit: 8 digits.

nK
specifies that n times 1024 bytes of storage are to be reserved.

Limit: 5 digits.

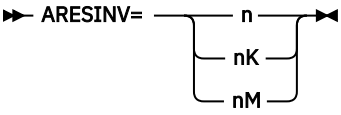
nM
specifies that n times 1048576 bytes of storage are to be reserved.

Limit: 2 digits.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

ARESINV



Temporarily overrides the ARESINV installation option, which specifies the number of bytes to be reserved for an invoking program's user exits that reside in or use space above 16MB virtual. The reserved space is not meant to be used for the invoking program's executable code. ARESINV is used only when DFSORT is dynamically invoked.

ARESINV applies only to the amount of main storage above 16MB virtual. The RESINV option applies to the amount of main storage below 16MB virtual.

n
specifies that n bytes of storage are to be reserved.

Limit: 8 digits.

nK
specifies that n times 1024 bytes of storage are to be reserved.

Limit: 5 digits.

nM
specifies that n times 1048576 bytes of storage are to be reserved.

Limit: 2 digits.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

AVGRLEN

Specifies the average input record length in bytes for variable-length record sort applications. This value is used when necessary to determine the input file size. The resulting value is important for sort applications, because it is used for several internal optimizations as well as for dynamic work data set allocation (see OPTION DYNALLOC). See "Specify input/output data set characteristics accurately" on page 749 and "Allocation of work data sets" on page 799 for more information on file size considerations.

n
specifies the average input record length. n must be between 4 and 32767 and must include the 4-byte record descriptor word (RDW).

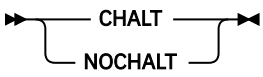
Note:

1. AVGRLEN=n on the OPTION statement overrides the L5 value on the RECORD statement (LENGTH operand) if both are specified. The L5 value on the RECORD statement is ignored for Blockset.
2. L5=n can be used instead of AVGRLEN=n.

Default: If AVGRLEN=n is not specified, DFSORT uses one-half of the maximum record length as the average record length. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

CHALT or NOCHALT



Temporarily overrides the CHALT installation option, which specifies whether format CH fields are translated by the alternate collating sequence as well as format AQ or just the latter.

CHALT
specifies that DFSORT translates character control fields and compare fields with formats CH and AQ using the alternate collating sequence.

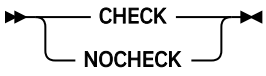
NOCHALT
specifies that format CH fields are not translated.

Note: If you use locale processing for SORT, MERGE, INCLUDE, or OMIT fields, you must not use CHALT. If you need alternate sequence processing for a particular field, use format AQ.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

CHECK or NOCHECK



Temporarily overrides the CHECK installation option, which specifies whether the record count should be checked for applications that use the E35 user exit routine without an output data set.

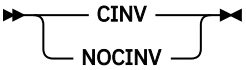
CHECK
specifies that the record count should be checked.

NOCHECK
specifies that the record count should not be checked.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

CINV or NOCINV



Temporarily overrides the CINV installation option, which specifies whether DFSORT can use control interval access for VSAM data sets. The Blockset technique uses control interval access for VSAM input data sets, when possible, to improve performance.

CINV
specifies that DFSORT should use control interval access when possible for VSAM data sets.

NOCINV
specifies that DFSORT should not use control interval access.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

CKPT

Activates the Checkpoint/Restart facility for sorts that use the Peerage or Vale techniques.

Because CKPT is only supported in the Peerage and Vale techniques, the Blockset technique must be bypassed for the Checkpoint/Restart facility to be used. Installation option IGNCKPT=NO causes Blockset to be bypassed when CKPT is specified at run-time. The NOBLKSET option can also be used to bypass Blockset at run-time.

A SORTCKPT DD statement must be coded when you use the Checkpoint/Restart facility (see “SORTCKPT DD statement” on page 73).

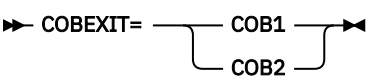
Note:

1. CHKPT can be used instead of CKPT.
2. Functions such as OUTFIL processing, which are supported only by the Blockset technique, cannot be used if the Checkpoint/Restart facility is used.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

COBEXIT



Temporarily overrides the COBEXIT installation option, which specifies the library for COBOL E15 and E35 routines.

COB1
specifies that COBOL E15 and E35 routines are run with the OS/VS COBOL run-time library or, in some cases, with no COBOL run-time library.

COBEXIT=COB1 is **obsolete**, but is still available for compatibility reasons.

Note that Language Environment is the only run-time library for COBOL supported by IBM service.

COB2
specifies that COBOL E15 and E35 routines are run with either the VS COBOL II run-time library or the Language Environment run-time library.

Note that Language Environment is the only run-time library for COBOL supported by IBM service.

Note: The DFSORT documents only discuss the Language Environment run-time library, and assume that COBEXIT=COB2 is in effect. Although it is possible to run with older run-time libraries, and with COBEXIT=COB1, these are not recommended or discussed, and are not supported by IBM service.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

COPY

Causes DFSORT to copy a SORTIN data set or inserted records to the output data sets unless all records are disposed of by an E35 exit routine. Records can be edited by E15 and E35 exit routines; INCLUDE/OMIT, INREC, OUTREC, and OUTFIL statements; and SKIPREC and STOPAFT parameters. E35 is entered after each SORTIN or E15 record is copied.

The following must not be used in copy applications:

- BDAM data sets
- Dynamic binding or link-editing.

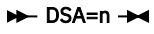
See message ICE160A in *z/OS DFSORT Messages, Codes and Diagnosis Guide* for additional restrictions that apply to copy applications.

Note: User labels will not be copied to the output data sets.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

DSA



Temporarily overrides the DSA installation option, which specifies the maximum amount of storage available to DFSORT for dynamic storage adjustment of a Blockset sort application when SIZE/MAINSIZE=MAX is in effect. If you specify a DSA value greater than the TMAXLIM value in effect, you allow DFSORT to use more storage than the TMAXLIM value if doing so should improve performance. The amount of storage DFSORT uses is subject to the DSA value as well as system limits such as region size. However, whereas DFSORT always tries to obtain as much storage as it can up to the TMAXLIM value, DFSORT only tries to obtain as much storage as needed to improve performance up to the DSA value.

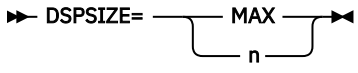
The performance improvement from dynamic storage adjustment usually provides a good tradeoff against the increased storage used by DFSORT. On storage constrained systems, however, the DSA value should be set low enough to prevent unacceptable paging.

n
specifies that DFSORT can dynamically adjust storage to improve performance, subject to a limit of n MB. n must be a value between 0 and 2000. If n is less than or equal to the TMAXLIM value in effect, n is set to 0 to indicate that storage will not be dynamically adjusted.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

DSPSIZE



Temporarily overrides the DSPSIZE installation option, which specifies the maximum amount of data space to be used with dataspace sorting. A data space is an area of contiguous virtual storage that is backed by real and auxiliary storage, whichever is necessary as determined by the system. Because DFSORT is able to sort large pieces of data using data space, CPU time and elapsed time are reduced.

Several factors can limit the amount of data space an application uses:

- The IEFUSI exit can limit the total amount of Hiperspace, memory objects and data space available to an application.
- DSPSIZE can limit the amount of data space available to an application, as detailed later in this section.
- Sufficient available storage must be present to back DFSORT's data spaces. "Available" storage is the storage used to back new data space data and consists of the following two types:

  1. Free storage. This is storage not being used by any application.
  2. Old storage. This is storage used by another application whose data has been unreferenced for a sufficiently long time so that the system considers it eligible to be paged out to auxiliary storage to make room for new data space data.

  The amount of available storage constantly changes, depending on current system activity. Consequently, DFSORT checks the amount of available central storage throughout a data space sorting run and switches from using a data space to using disk work data sets if the available central storage is too low.
- Other concurrent Hipersorting, memory sorting and dataspace sorting applications further limit the amount of available storage. A dataspace sorting application knows the storage needs of every other Hipersorting, memory object sorting and dataspace sorting application on the system. A dataspace sorting application does not try to back its data space data with storage needed by another Hipersorting, memory object sorting or dataspace sorting application. This prevents overcommitment of storage resources if multiple large concurrent DFSORT applications start at similar times on the same system.
- The installation options EXPMAX, EXPOLD, and EXPRES can also be used to further limit the amount of storage available to dataspace sorting applications. EXPMAX limits the total amount of available storage that can be used at any one time to back DFSORT Hiperspaces, memory objects and data spaces. EXPOLD limits the total amount of old storage that can be used at any one time to back DFSORT Hiperspaces, memory objects and data spaces. EXPRES sets aside a specified amount of available storage for use by non-Hipersorting, non-memory object sorting and non-dataspace sorting applications.

Some of these limits depend on system and other DFSORT activity throughout the time a dataspace sorting application runs. Consequently, the amount of data space a dataspace sorting application uses can vary from run to run.

If the amount of data space DFSORT decides to use is sufficient, DFSORT sorts your data in main storage and does not require additional temporary work space. If the amount of data space is not sufficient, DFSORT uses disk as temporary work space. Installation option DYNAUTO=NO is changed to DYNAUTO=YES whenever dataspace sorting is possible. Hiperspace is not used when dataspace sorting is used.

MAX
specifies that DFSORT dynamically determines the maximum amount of data space to be used for dataspace sorting. In this case, DFSORT bases its data space usage on the size of the file being sorted and the paging activity of the system.

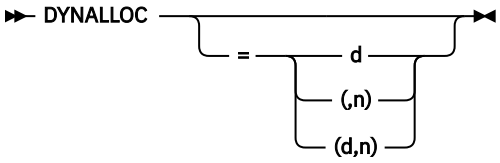
n
specifies the maximum amount, in MB, of data space to be used for dataspace sorting. n must be a value between 0 and 9999. The actual amount of data space used does not exceed n, but may be less than n depending on the size of the file being sorted and the paging activity of the system.

If n is zero, dataspace sorting is not used.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

DYNALLOC



Assigns DFSORT the task of dynamically allocating needed work space. You do not need to calculate and use JCL to specify the amount of work space needed by the program. DFSORT uses the dynamic allocation facility of the operating system to allocate work space for you.

Refer to Appendix A, “Using work space,” on page 797 for guidelines on the use of DYNALLOC.

d
specifies the device name. You can specify any IBM disk or tape device supported by your operating system in the same way you would specify it in the JCL UNIT parameter. You can also specify a group name, such as DISK or SYSDA.

n
specifies the maximum number of requested work data sets. If you specify more than 255, a maximum of 255 data sets is used. If you specify 1 and the Blockset technique is selected, a maximum of 2 data sets is used. If you specify more than 32 and the Blockset technique is not selected, a maximum of 32 data sets is used.

Tip: For optimum allocation of resources such as virtual storage, avoid specifying a large number of work data sets unnecessarily.

For tape work data sets, the number of volumes specified (explicitly or by default) is allocated to the program. The program requests standard label tapes.

DYNALLOC is not used if SORTWKdd DD statements are provided unless installation option DYNAUTO=IGNWKDD is specified and OPTION USEWKDD is not in effect.

`If VIO=NO is in effect:`

- Work space can be allocated on nontemporary data sets (DSNAME parameter specified).

- If the device (d) you specify is a virtual device and reallocation to a real device fails, DFSORT will ignore VIO=NO and use the virtual device.

Note: Message ICE165I gives information about work data set allocation/use.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

- DYNALLOC can automatically be activated by using the DYNAUTO installation option.
- If DYNALLOC is specified without d, the default for d is that specified (or defaulted) by the DYNALOC installation option
- If DYNALLOC is specified without n, the default for n is that specified (or defaulted) by the DYNALOC installation option.

You can specify DYNALLOC without n, without d, or without both. If DYNALLOC is specified without n, and the IBM–supplied default for the n value of the DYNALOC installation option is chosen, then:

- If one of the Blockset techniques is chosen, four work data sets will be requested.
- If a technique other than Blockset is chosen, three work data sets will be requested.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

DYNALLOC=OFF

Directs DFSORT *not* to allocate work space dynamically, overriding that function of installation option DYNAUTO=YES, or DYNAUTO=IGNWKDD, or the run-time option DYNALLOC (without OFF). Use this option when you know that an in-core sort can be performed, and you want to suppress dynamic allocation of work space.

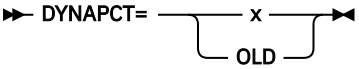
OFF
directs DFSORT not to allocate work space dynamically.

Note: When Hipersorting or dataspace sorting is in effect, DFSORT uses dynamic allocation when necessary, even if DYNALLOC=OFF has been specified.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

DYNAPCT



specifies additional work data sets to be dynamically allocated with zero space. DFSORT only extends these data sets when necessary to complete a sort application. The availability of additional work data sets can help avoid out of space ABENDs.

x
specifies the number of additional work data sets (y) as a percentage of the maximum number of dynamically allocated work data sets (DYNALLOC/DYNALOC n value) in effect. y will be set to n * x%. The total number of dynamically allocated work data sets will be n + y. For example, if DYNALLOC=(SYSDA,20) and DYNAPCT=20 are in effect, 4 additional work data sets will be allocated for a total of 24.

The value x must be between 0 and 254. The minimum value for y is 1 and the maximum value for y is 254. The maximum value for n + y is 255; if x results in a value for n + y greater than 255, y will be set to 255-n.

OLD
specifies additional work data sets should only be allocated when DFSORT cannot determine the file size. When DFSORT is able to determine the file size, additional work data sets will not be allocated (y=0), and the total number of work data sets will be n.

Note: When message ICE118I is issued indicating that DFSORT cannot determine the file size, y is set as follows:

- For DYNAPCT=OLD, y is set to n * 50%
- For DYNAPCT=x with x <= 50, y is set to n * 50%
- For DYNAPCT=x with x > 50, y is set to n * x%

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

DYNSPC

DYNSPC=ntemporarily overrides the DYNSPC installation option, which specifies the total default primary space allocation for all of the dynamically allocated work data sets when the input file size is unknown. That is, when DFSORT cannot determine the input file size for a sort application and the number of records is not supplied by a FILSZ or SIZE value.

Generally, DFSORT can automatically determine the input file size. However, in a few cases, such as when an E15 supplies all of the input records, when information about a tape data set is not available from a tape management system, or when Blockset is not selected, DFSORT cannot determine an accurate file size. In these cases, if the number of records is not supplied by the FILSZ or SIZE run-time option, and dynamic allocation of work data sets is used, DFSORT uses the DYNSPC value in effect as the approximate amount of primary space. DFSORT uses 20% of the primary space as secondary space. Although the primary space is always allocated, secondary space (up to 15 extents) is only allocated as needed.

You may want to use DYNSPC to override the installation default with a larger or smaller value depending on the amount of disk space available for DFSORT work data sets, and the amount of data to be sorted for this application. As a guideline, Table 38 on page 183 shows the approximate primary space in cylinders that is allocated on a 3390 when Blockset sorts an unknown number of 6000-byte records.

Table 38. Example of DYNSPC Primary Space

DYNSPC value (megabytes)	Primary space (cylinders)
32	48
64	93
128	183
256	366
512	732

The larger your DYNSPC value, the more data DFSORT can sort when the file size is unknown. For example, in a test using just dynamically allocated work space (no Hiperspace or data space) with the primary space shown in Table 38 on page 183, and all of the corresponding secondary space, Blockset is able to sort approximately 150 megabytes with DYNSPC=32 and approximately 1200 megabytes with DYNSPC=256. If Hiperspace or data space can be used along with dynamically allocated work space, the amount of data DFSORT can sort will increase according to the amount of Hiperspace or data space available.

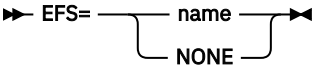
n
specifies the *total* default primary space, in megabytes, to be allocated for *all* dynamically allocated work data sets (n is *not* the primary space for each data set). n must be a value between 1 and 65535.

Do not specify a value which exceeds the available disk space, because this causes dynamic allocation to fail for sort applications that use this value.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

EFS



Temporarily overrides the EFS installation option, which specifies whether DFSORT is to pass control to an Extended Function Support (EFS) program. See Chapter 9, "Using extended function support," on page 717 for more information.

name
specifies the name of the EFS program that will be called to interface with DFSORT.

NONE
specifies no call will be made to the EFS program.

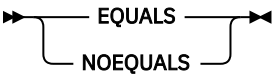
Note:

1. EFS is processed only if it is passed on the OPTION control statement in an extended parameter list or in DFSPARM.
2.
3. If you use locale processing for SORT, MERGE, INCLUDE, or OMIT fields, you must not use an EFS program. DFSORT's locale processing may eliminate the need for an EFS program. See the LOCALE option later in this section for information related to locale processing.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

EQUALS or NOEQUALS



Temporarily overrides the EQUALS installation option, which specifies whether the original sequence of records that collate identically for a sort or a merge should be preserved from input to output.

EQUALS
specifies that the original sequence must be preserved.

NOEQUALS
specifies that the original sequence need not be preserved.

For sort applications, the sequence of the output records depends upon the order of:

- The records from the SORTIN file
- The records inserted by an E15 user exit routine
- The E15 records inserted within input from SORTIN.

For merge applications, the sequence of the output records depends upon the order of:

- The records from a SORTINnn file. Records that collate identically are output in the order of their file increments. For example, records from SORTIN01 are output before any records that collate identically from SORTIN02.
- The records from an E32 user exit routine for the same file increment number. Records that collate identically from E32 are output in the order of their file increments. For example, records from the file with increment 0 are output before any records that collate identically from the file with increment 4.

Note:

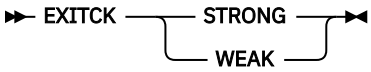
1. When EQUALS is in effect, the total number of bytes occupied by all control fields must not exceed 4088.
2. Using EQUALS can degrade performance.
3. When EQUALS is in effect with SUM, the first record of summed records is kept. When NOEQUALS is in effect with SUM, the record to be kept is unpredictable.

   If a technique other than Blockset is selected, NOEQUALS is forced if SUM is specified.
4. Do not specify EQUALS if variable-length records are sorted using tape work files and the RDW is part of the control field.
5. The number of records to be sorted cannot exceed 4294967295 (4 gigarecords minus 1); if the number of records exceeds this number, message ICE121A is issued and DFSORT terminates.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

EXITCK



Temporarily overrides the EXITCK installation option, which specifies whether DFSORT terminates or continues when it receives certain invalid return codes from E15 or E35 user exit routines. For full details of the return codes affected by this parameter, see “E15/E35 return codes and EXITCK” on page 512.

STRONG
specifies that DFSORT issues an error message and terminates when it receives an invalid return code from an E15 or E35 user exit routine.

WEAK
specifies that DFSORT interprets certain invalid return codes from E15 and E35 user exit routines as valid and continues processing. Use of EXITCK=WEAK can make it difficult to detect errors in the logic of E15 and E35 user exit routines.

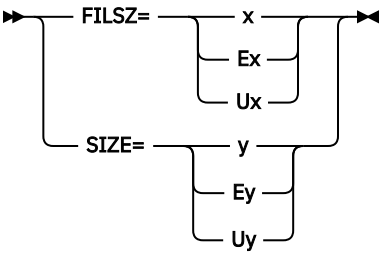
Note: EXITCK=WEAK is treated like EXITCK=STRONG when:

- Tape work data sets are specified for a sort application.
- The Blockset technique is not selected for a merge application.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

FILSZ or SIZE



The FILSZ parameter specifies either the exact number of records to be sorted or merged, or an estimate of the number of records to be sorted. The SIZE parameter specifies either the exact number of records in the input data sets, or an estimate of the number of records in the input data sets. The supplied record count is used by DFSORT for two purposes:

1. To check that the actual number of records sorted or merged or the number of records in the input data sets is equal to the exact number of records expected. FILSZ=x or SIZE=y causes this check to be performed and results in termination with message ICE047A if the check fails.
2. To determine the input file size for a sort application. DFSORT performs calculations based on the user supplied record count and other parameters (such as AVGRLEN) to estimate the total number of bytes to be sorted. This value is important for sort runs, because it is used for several internal optimizations as well as for dynamic work data set allocation (see OPTION DYNALLOC). If no input record count (or only an estimate) is supplied for the sort run, DFSORT attempts to automatically compute the file size to be used for the optimizations and allocations.

The type of FILSZ or SIZE value specified (x/y, Ux/Uy, Ex/Ey, or none) controls the way DFSORT performs the previous two functions, and can have a significant effect on performance and work data set allocation. See Chapter 10, “Improving efficiency,” on page 747 and “File size and dynamic allocation” on page 800 for more information on file size considerations.

x or y
specifies the exact number of records to be sorted or merged (x) or the exact number of records in the input data sets (y). This value is always used for both the record check and the file size calculations. FILSZ=x or SIZE=y can be used to force DFSORT to perform file size calculations based on x or y, and to cause DFSORT to terminate the sort or merge application if x or y is not exact.

If installation option FSZEST=NO is in effect and either FILSZ=x or SIZE=y is specified, DFSORT terminates if the actual number of records is different from the specified exact value (x or y). In this case, the actual number of records is placed in the IN field of message ICE047A (or message ICE054I in some cases) before termination. However, if installation option FSZEST=YES is in effect, DFSORT treats FILSZ=x or SIZE=y like FILSZ=Ex or SIZE=Ey, respectively; it does not terminate when the actual number of records does not equal x or y.

FILSZ=0 causes Hipersorting, dataspace sorting, and dynamic allocation of work space not to be used, and results in termination with message ICE047A unless the number of records sorted or merged is 0. If no E15 user exit is present, SIZE=0 has the same effect in terms of Hipersorting and dynamic allocation of work space, and results in termination with message ICE047A unless the number of records in the input data sets is 0.

x
specifies the exact number of records to be sorted or merged; it must take into account the number of records in the input data sets, records to be inserted or deleted by E15 or E32, and records to be deleted by the INCLUDE/OMIT statement, SKIPREC, and STOPAFT. x must be changed whenever the number of records to be sorted or merged changes in any way.

y
specifies the exact number of records in the input data sets; it must take into account the number of records to be deleted by STOPAFT. y must be changed whenever the number of records in the input data sets changes in any way.

Limit: 28 digits (15 significant digits)

Ex or Ey
specifies an estimate of the number of records to be sorted (x) or an estimate of the number of records in the input data sets (y). This value is not used for the record check. It is used for the file size calculations, but only if DFSORT could not reasonably estimate the input file size itself. In all other cases, this value is ignored by DFSORT. See "Dynamic allocation of work data sets" on page 800 for details on exactly when an estimated record count is used and when it is ignored by DFSORT.

FILSZ=E0 or SIZE=E0 is always ignored.

x
specifies an estimate of the number of records to be sorted; it should take into account the number of records in the input data sets, records to be inserted or deleted by E15, and records to be deleted by the INCLUDE/OMIT statement, SKIPREC, and STOPAFT. x should be changed whenever the number of records to be sorted changes significantly.

y
specifies an estimate of the number of records in the input data sets; it should take into account the number of records to be deleted by STOPAFT. y should be changed whenever the number of records in the input data sets changes significantly.

Limit: 28 digits (15 significant digits)

Ux or Uy
specifies the number of records to be sorted (x) or the number of records in the input data sets (y). This value is not used for the record check, but is always used for the file size calculations. FILSZ=Ux or SIZE=Uy can be used to force DFSORT to perform file size calculations based on x or y, while avoiding termination if x or y is not exact.

The FSZEST installation option has no effect on FILSZ=Ux or SIZE=Uy processing.

FILSZ=U0 causes Hipersorting, dataspace sorting, and dynamic allocation of work space not to be used, and may cause degraded performance or termination with message ICE046A, if the actual number of records to be sorted is significantly larger than 0. If no E15 user exit is present, SIZE=U0 has the same effect in terms of Hipersorting, dataspace sorting,and dynamic allocation of work space, and may cause degraded performance or termination with message ICE046A, if the actual number of records in the input data sets is significantly larger than 0.

x
specifies the number of records to be sorted; it should take into account the number of records in the input data sets, records to be inserted or deleted by E15, and records to be deleted by the INCLUDE/OMIT statement, SKIPREC, and STOPAFT. x should be changed whenever the number of records to be sorted changes significantly.

y
specifies the number of records in the input data sets; it should take into account the number of records to be deleted by STOPAFT. y should be changed whenever the number of records in the input data sets changes significantly.

Limit: 28 digits (15 significant digits)

Table 39 on page 187 summarizes the differences for the three FILSZ variations:

Table 39. FILSZ Variations Summary

FILSZ=n is equivalent to FILSZ=En if installation option FSZEST=YES is specified.Conditions	**FILSZ=n**	**FILSZ=Un**	**FILSZ=En**
Number of records	Exact	Estimate	Estimate
Applications	Sort, merge	Sort	Sort
Terminate if n wrong?	Yes	No	No

Table 39. FILSZ Variations Summary (continued)

FILSZ=n is equivalent to FILSZ=En if installation option FSZEST=YES is specified.Conditions	FILSZ=n	FILSZ=Un	FILSZ=En
Use for file size calculation?	Yes	Yes	When DFSORT cannot compute file size
n includes records:			
In input data sets	Yes	Yes	Yes
Inserted/deleted by E15	Yes	Yes	Yes
Inserted by E32	Yes	No	No
Deleted by INCLUDE/ OMIT	Yes	Yes	Yes
Deleted by SKIPREC	Yes	Yes	Yes
Deleted by STOPAFT	Yes	Yes	Yes
Update n when number of records changes:	In any way	Significantly	Significantly
Effects of n=0	Hipersorting and DYNALLOC not used	Hipersorting and DYNALLOC not used	None

Table 40 on page 188 summarizes the differences for the three SIZE variations:

Table 40. SIZE Variations Summary

SIZE=n is equivalent to SIZE=En if installation option FSZEST=YES is specified.Conditions	SIZE=n	SIZE=Un	SIZE=En
Number of records	Exact	Estimate	Estimate
Applications	Sort, merge	Sort	Sort
Terminate if n wrong?	Yes	No	No
Use for file size calculation?	Yes	Yes	When DFSORT cannot compute file size
n includes records:			
In input data sets	Yes	Yes	Yes
Inserted/deleted by E15	No	No	No
Inserted by E32	No	No	No
Deleted by INCLUDE/ OMIT	No	No	No
Deleted by SKIPREC	No	No	No
Deleted by STOPAFT	Yes	Yes	Yes
Update n when number of records changes:	In any way	Significantly	Significantly

Table 40. SIZE Variations Summary (continued)

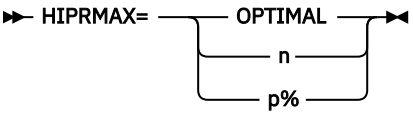
SIZE=n is equivalent to SIZE=En if installation option FSZEST=YES is specified.Conditions	SIZE=n	SIZE=Un	SIZE=En
Effects of n=0	Hipersorting and DYNALLOC not used	Hipersorting and DYNALLOC not used	None

Attention: Using the SIZE or FILSZ parameter to supply inaccurate information to DFSORT can negatively affect DFSORT performance, and, when work space is dynamically allocated, can result in wasted disk space or termination with message ICE083A or ICE046A. Therefore, it is important to update the record count value whenever the number of records to be sorted changes significantly.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

HIPRMAX



Temporarily overrides the HIPRMAX installation option, which specifies the maximum amount of Hiperspace to be used for Hipersorting. Hiperspace is a high-performance data space that resides in central storage and is backed by auxiliary storage (if necessary). Because I/O processing is reduced for Hipersorting, elapsed time, EXCP counts, and channel usage are also reduced.

Several factors can limit the amount of Hiperspace an application uses:

- The IEFUSI exit can limit the total amount of Hiperspace, memory objects and data space available to an application.
- HIPRMAX can limit the amount of Hiperspace available to an application, as detailed later in this section.
- Sufficient available storage must be present to back DFSORT's Hiperspaces. "Available" storage is the storage used to back new Hiperspace data and consists of the following two types:
  1. Free storage. This is storage not being used by any application.
  2. Old storage. This is storage used by another application whose data has been unreferenced for a sufficiently long time so that the system considers it eligible to be paged out to auxiliary storage to make room for new Hiperspace data.

The amount of available storage constantly changes, depending upon current system activity. Consequently, DFSORT checks the available storage level throughout a Hipersorting application and switches from Hiperspace to work data sets if the available storage level gets too low.

- Other concurrent Hipersorting, memory sorting and dataspace sorting applications further limit the amount of available storage. A Hipersorting application knows the storage needs of every other Hipersorting, memory object sorting and dataspace sorting application on the system. A Hipersorting application does not try to back its Hiperspace data with storage needed by another Hipersorting, memory object sorting, or dataspace sorting application. This prevents overcommitment of storage resources if multiple large concurrent DFSORT applications start at similar times on the same system.
- The installation options EXPMAX, EXPOLD, and EXPRES can also be used to further limit the amount of storage available to Hipersorting applications. EXPMAX limits the total amount of available

storage that can be used at any one time to back DFSORT Hiperspaces, memory objects and data spaces. EXPOLD limits the total amount of old storage that can be used at any one time to back DFSORT Hiperspaces, memory objects and data spaces. EXPRES sets aside a specified amount of available storage for use by non-Hipersorting, non-memory object sorting, and non-dataspace sorting applications.

Some of these limits depend on system and other DFSORT activity throughout the time a Hipersorting application runs. Consequently, the amount of Hiperspace a Hipersorting application uses can vary from run to run.

HIPRMAX=n specifies a fixed value for HIPRMAX. HIPRMAX=p% specifies a value for HIPRMAX that varies as a percentage of an appropriate portion of central storage. If the storage on a system changes, HIPRMAX=p% will cause a corresponding change in the HIPRMAX value selected by DFSORT, whereas HIPRMAX=n will not. When sharing DFSORT installation options between systems, such as in a sysplex, HIPRMAX=p% can be used to tailor the HIPRMAX value to the system selected for the application, providing a more dynamic HIPRMAX value than HIPRMAX=n.

If the amount of Hiperspace available for Hipersorting is insufficient for temporary storage of the records, intermediate disk storage is used along with Hiperspace. If the amount of Hiperspace is too small to improve performance, Hipersorting is not used. DYNAUTO=NO is changed to DYNAUTO=YES for Hipersorting.

Hipersorting can cause a small CPU time increase. When CPU optimization is a concern, you can use HIPRMAX=0 to suppress Hipersorting.

Note: HIPRLIM=OPTIMAL can be used instead of HIPRMAX=OPTIMAL. HIPRLIM=m can be used instead of HIPRMAX=n. HIPRLIM=m specifies a Hiperspace limit of m times 4096 bytes rounded up to the nearest megabyte. m must be a value between 0 and 2559744. If m is 0, Hipersorting is not used.

OPTIMAL
specifies that DFSORT determines dynamically the maximum amount of Hiperspace to be used for Hipersorting.

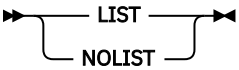
n
specifies that DFSORT determines dynamically the maximum amount of Hiperspace to be used for Hipersorting, subject to a limit of nMB. n must be a value between 0 and 32767. If n is 0, Hipersorting is not used.

p%
specifies that DFSORT determines dynamically the maximum amount of hiperspace to be used for Hipersorting, subject to a limit of p percent of an appropriate portion of central storage. p must be a value between 0 and 100. If p is 0, Hipersorting is not used. The value calculated for p% is limited to 32767MB, and is rounded down to the nearest MB.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

LIST or NOLIST



Temporarily overrides the LIST installation option, which specifies whether DFSORT program control statements should be written to the message data set. See *z/OS DFSORT Messages, Codes and Diagnosis Guide* for details on use of the message data set.

LIST
specifies that DFSORT control statements are printed to the message data set.

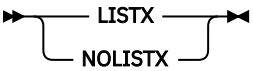
NOLIST
specifies that DFSORT control statements are not printed to the message data set.

Note: LIST or NOLIST are processed only if they are passed on the OPTION control statement in an extended parameter list or in DFSPARM.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

LISTX or NOLISTX



Temporarily overrides the LISTX installation option, which specifies whether DFSORT writes to the message data set program control statements that are returned by an EFS program. See *z/OS DFSORT Messages, Codes and Diagnosis Guide* for details on use of the message data set.

LISTX
specifies that control statements returned by an EFS program are printed to the message data set.

NOLISTX
specifies that control statements returned by an EFS program are not printed to the message data set.

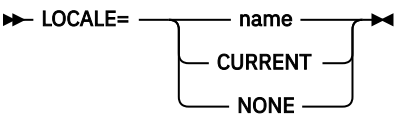
Note:

1. LISTX or NOLISTX are processed only if they are passed on the OPTION control statement in an extended parameter list or in DFSPARM.
2. If EFS=NONE is in effect after final override rules have been applied, NOLISTX is in effect.
3. LISTX and NOLISTX can be used independently of LIST and NOLIST.
4. For more information on printing EFS control statements, see *z/OS DFSORT Messages, Codes and Diagnosis Guide*

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

LOCALE



Temporarily overrides the LOCALE installation option, which specifies whether locale processing is to be used and, if so, designates the active locale.

DFSORT's collating behavior can be modified according to your cultural environment. Your cultural environment is defined to DFSORT using the X/Open locale model. A locale is a collection of data grouped into categories that describes the information about your cultural environment.

The collate category of a locale is a collection of sequence declarations that defines the relative order between collating elements (single character and multi-character collating elements). The sequence declarations define the collating rules.

If locale processing is to be used, the active locale will affect the behavior of DFSORT's SORT, MERGE, INCLUDE, and OMIT functions. For SORT and MERGE, the active locale will only be used to process character (CH) control fields. For INCLUDE and OMIT, the active locale will only be used to process character (CH) compare fields, and character and hexadecimal constants compared to character (CH) compare fields.

Note: Locale processing is not used for IFTRAIL TRLID or IFTHEN WHEN, BEGIN or END constants or compare fields.

name
specifies that locale processing is to be used and designates the name of the locale to be made active during DFSORT processing.

The locales are designated using a descriptive name. For example, to set the active locale to represent the French language and the cultural conventions of Canada, specify LOCALE=FR_CA. You can specify up to 32 characters for the descriptive locale name. The locale names themselves are not case-sensitive. See *Using Locales* for complete locale naming conventions.

You can use IBM-supplied and user-defined locales.

The state of the active locale prior to DFSORT being entered will be restored on DFSORT's completion.

CURRENT
specifies that locale processing is to be used, and the current locale active when DFSORT is entered will remain the active locale during DFSORT processing.

NONE
specifies that locale processing is not to be used. DFSORT will use the binary encoding of the code page defined for your data for collating and comparing.

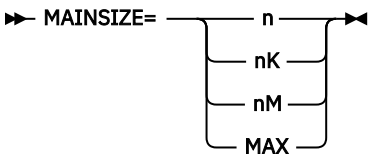
Note:

1. LOCALE is processed only if it is passed on the OPTION control statement in an extended parameter list or in DFSPARM.
2. To use an IBM-supplied locale, DFSORT must have access to the Language Environment run-time library. For example, this library might be called SYS1.SCEERUN. If you are unsure of the name of this library at your location, contact your system administrator. To use a user-defined locale, DFSORT must have access to the load library containing it.
3. If you use locale processing for SORT, MERGE, INCLUDE, or OMIT fields:
   - VLSHRT is not used for SORT or MERGE. DFSORT pads the record to allow the job to continue even if all records do not have sufficient length to have data at the fields being referred to, but a padded record is not treated the same as an original record when it comes to sorting with EQUALS on those fields that do not exist. As the short records are treated differently from the records that contain ALL the sort fields, your SORT order results on short records when using EQUALS, are unpredictable.
   - INREC or an E61 user exit must not be used for SORT or MERGE.
   - CHALT or an EFS program must not be used.
4. Locale processing is not used for IFTRAIL TRLID or IFTHEN WHEN, BEGIN or END constants or compare fields.
5. Locale processing for DFSORT's SORT, MERGE, INCLUDE, and OMIT functions can improve performance relative to applications that must perform pre-processing or post-processing of data to produce the desired collating results. However, locale processing should be used only when required, because it can show degraded performance relative to collating, using character encoding values.
6. DFSORT locale processing may require an additional amount of storage that depends on the environment supporting the locale as well as the locale itself. It may be necessary to specify a REGION of several MB or more for DFSORT applications that use locale processing.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions:; See Appendix B, “Specification/override of DFSORT options,” on page 805.

MAINSIZE



Temporarily overrides the SIZE installation option, which specifies the amount of main storage available to DFSORT.

MAINSIZE applies to the total amount of main storage above and below 16MB virtual. DFSORT determines how much storage to allocate above and below 16MB virtual, but the total amount of storage cannot exceed MAINSIZE.

Storage used for OUTFIL processing will be adjusted automatically, depending upon several factors, including:

- Total available storage
- Non-OUTFIL processing storage requirements
- Number of OUTFIL data sets and their attributes (for example, block size).

OUTFIL processing is subject to the ODMAXBF limit and your system storage limits (for example, IEFUSI) but not to DFSORT storage limits, that is, SIZE/MAINSIZE, MAXLIM, and TMAXLIM. DFSORT attempts to use storage above 16MB virtual for OUTFIL processing whenever possible.

For details on main storage allocation, see "Tuning main storage" on page 755.

n
specifies that n bytes of storage are to be allocated. If you specify more that 2097152000, 2097152000 is used.

Limit: 10 digits

nK
specifies that n times 1024 bytes of storage are to be allocated. If you specify more than 2048000K, 2048000K is used.

Limit: 7 digits

nM
specifies that n times 1048576 bytes of storage are to be allocated. If you specify more than 2000M, 2000M is used.

Limit: 4 digits.

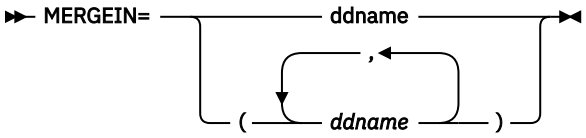
MAX
instructs DFSORT to calculate the amount of virtual storage available and allocate an amount of storage up to the TMAXLIM or DSA value when Blockset is selected, or up to the MAXLIM value when Blockset is not selected.

Note: CORE=value can be used instead of MAINSIZE=value.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

MERGEIN



Specifies up to 100 ddnames to be be used for a MERGE application instead of the SORTINnn ddnames. This allows you to use any valid alternate ddnames for the MERGE data sets.

If EQUALS is in effect, records that collate identically are output in the order of their ddnames in the MERGEIN list.

Each ddname can be 1 through 8 characters. If a ddname is specified more than once in the MERGEIN operand, it will only be used once. If more than 100 unique ddnames are specified in the MERGEIN operand, only the first 100 will be used. Do not use ddnames reserved for use by DFSORT, such as SYSOUT, ccccOUT, ccccWKd, ccccWKdd, ccccDKd, or ccccDKdd , where cccc is the specified or defaulted value for the SORTDD operand and d is any character.

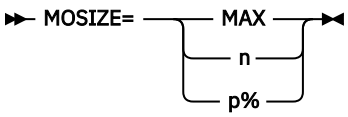
Note:

1. MERGEIN is processed only if it is passed on the OPTION control statement in an extended parameter list, or in DFSPARM.
2. If both MERGEIN=(ddname1,ddname2,...) and SORTDD=cccc are specified, ddname1, ddname2 and so on are used for the input files. The same ddname cannot be specified for MERGEIN and SORTOUT, or for MERGEIN and OUTFIL.

Default: SORTINnn, unless SORTDD=cccc is specified in an extended parameter list or in DFSPARM, in which case ccccINnn is the default.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

MOSIZE



Temporarily overrides the MOSIZE installation option, which specifies the maximum amount of memory object storage to be used for memory object sorting in 64-bit virtual storage. A memory object is a data area in virtual storage that is allocated above the bar and backed by central storage. Because I/O processing is reduced for memory object sorting, elapsed time, EXCP counts, and channel usage are also reduced.

Note: The "bar" refers to the 2-gigabyte address within the 64-bit address space. The bar separates storage below the 2-gigabyte address called "below the bar", from storage above the 2-gigabyte address called "above the bar".

There are several factors that can limit the total memory object storage used by an application:

1. The MEMLIMIT parameter on the JOB or EXEC JCL statement can limit the total number of usable virtual pages above the bar in a single address space.
2. In addition to limiting the total amount of Hiperspace and data space available to an application, the IEFUSI exit can also limit the total number of usable virtual pages above the bar in a single address space.
3. MOSIZE can limit the total amount of memory object storage available to an application, as detailed later in this section.
4. Sufficient available central storage must be present to back DFSORT's memory object storage. The amount of available central storage changes constantly, depending on current system activity. Consequently, DFSORT checks the amount of available central storage throughout a memory object sorting run and switches from using a memory object to using disk work data sets if the available central storage is too low.
5. Other concurrent Hipersorting, memory object sorting, and dataspace sorting applications further limit the amount of available storage. A memory object sorting application keeps track of the storage needs of all other Hipersorting, memory object sorting, and dataspace sorting applications

on the system, and does not attempt to back its memory object storage with storage needed by another Hipersorting, memory object sorting, or dataspace sorting application. This prevents overcommitment of storage resources in the event of multiple large concurrent Hipersorting, memory object sorting, and dataspace sorting applications starting at similar times on the same system.

6. The installation options EXPMAX, EXPOLD, and EXPRES can also be used to further limit the amount of storage available to memory object sorting applications. EXPMAX limits the total amount of available storage that can be used at any one time to back DFSORT Hiperspaces, memory objects, and data spaces. EXPOLD limits the total amount of old storage that can be used at any one time to back DFSORT Hiperspaces, memory objects, and data spaces. EXPRES sets aside a specified amount of available storage for use by non-Hipersorting, non-memory object sorting, and non-dataspace sorting applications.

Some of these limits depend on system and other DFSORT activity during the time that a memory object sorting application runs. Consequently, the total amount of memory object storage that a memory object sorting application uses can vary from run to run.

MOSIZE=n specifies a fixed value for MOSIZE. MOSIZE=p% specifies a value for MOSIZE that varies as a percentage of the available central storage on the system at run-time. If the available central storage on a system changes, MOSIZE=p% will cause a corresponding change in the MOSIZE value selected by DFSORT, whereas MOSIZE=n will not. When sharing DFSORT installation options between systems, such as in a sysplex, MOSIZE=p% can be used to tailor the MOSIZE value to the system selected for the application, providing a more dynamic MOSIZE value than MOSIZE=n.

If the total memory object storage available for memory object sorting is insufficient for temporary storage of the records, intermediate disk storage can be used along with the memory object. When memory object sorting is enabled, DFSORT changes DYNAUTO=NO to DYNAUTO=YES in some cases.

MAX
specifies that DFSORT determines dynamically the maximum size of a memory object to be used for memory object sorting. In this case, DFSORT bases its memory object usage on the size of the file being sorted and the central storage usage activity.

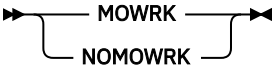
n
specifies that DFSORT determines dynamically the maximum size of a memory object to be used for memory object sorting, subject to a limit of n MB. n must be a value between 0 and 2147483646. The actual size used does not exceed n, but may be less, depending on the size of the file being sorted and the central storage usage activity on the system. If n is 0, memory object sorting is not used

p%
specifies that DFSORT determines dynamically the maximum size of a memory object to be used for memory object sorting, subject to a limit of p percent of the available central storage. p must be a value between 0 and 100. If p is 0, memory object sorting is not used. The value calculated for p% is limited to 2147483646 MB, and is rounded down to the nearest MB.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

MOWRK or NOMOWRK



Temporarily overrides the MOWRK installation option, which specifies whether the memory object storage available to DFSORT for memory object sorting can be used as intermediate work space. DFSORT has the capability of using memory object storage as intermediate work space (similar to the way Hiperspace is used but more efficient), or as an extension of main storage. Using memory object

storage as intermediate work space is the preferred and recommended choice, but can be disabled, if appropriate.

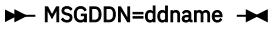
MOWRK
specifies that memory object storage can be used as intermediate work space, or as an extension of main storage, as appropriate.

NOMOWRK
specifies that memory object storage can only be used as an extension of main storage.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

MSGDDN



Temporarily overrides the MSGDDN installation option, which specifies an alternate ddname for the message data set. MSGDDN must be in effect if:

- A program that invokes DFSORT uses SYSOUT (for instance, COBOL uses SYSOUT) and you do not want DFSORT messages intermixed with the program messages.
- Your E15 and E35 routines are written in COBOL and you do not want DFSORT messages intermixed with the program messages.
- A program invokes DFSORT more than once and you want separate messages for each invocation of DFSORT.

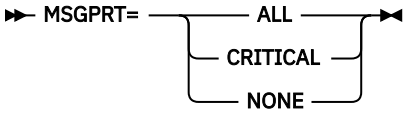
The ddname can be any 1- through 8- character name but must be unique within the job step; do not use a name that is used by DFSORT (for example, SORTIN). If the ddname specified is not available at run-time, SYSOUT is used instead. For details on use of the message data set, see *z/OS DFSORT Messages, Codes and Diagnosis Guide*

Note: MSGDDN is processed only if it is passed on the OPTION control statement in an extended parameter list or in DFSPARM.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

MSGPRT



Temporarily overrides the MSGPRT installation option, which specifies the class of messages to be written to the message data set. For details on use of the message data set, see *z/OS DFSORT Messages, Codes and Diagnosis Guide*.

ALL
specifies that all messages except diagnostic messages (ICE800I to ICE999I) are to be printed. Control statements print only if LIST is in effect.

CRITICAL
specifies that only critical messages will be printed. Control statements print only if LIST is in effect.

NONE
specifies that no messages and control statements will be printed.

Note:

1. MSGPRT is processed only if it is passed on the OPTION control statement in an extended parameter list or in DFSPARM.
2. PRINT=value can be used instead of MSGPRT=value.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

NOBLKSET

Causes DFSORT to bypass the Blockset technique normally used for a sort or merge application. Using this option generally results in degraded performance.

Note: Functions such as OUTFIL processing, which are supported only by the Blockset technique, cause the NOBLKSET option to be ignored.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

NOOUTREL

Temporarily overrides the OUTREL installation option, which specifies whether unused temporary output data set space is released. NOOUTREL means that unused temporary output data set space is *not* released.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

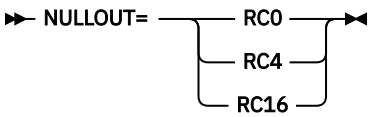
NOOUTSEC

Temporarily overrides the OUTSEC installation option, which specifies whether automatic secondary allocation is used for temporary or new output data sets. NOOUTSEC means that automatic secondary allocation for output data sets is not used.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

NULLOUT



Temporarily overrides the NULLOUT installation option, which specifies the action to be taken by DFSORT when there are no records for the SORTOUT data set, as indicated by an OUT count of 0 in message ICE054I

RC0
specifies that DFSORT should issue message ICE173I, set a return code of 0, and continue processing when there are no records for the SORTOUT data set.

RC4
specifies that DFSORT should issue message ICE173I, set a return code of 4, and continue processing when there are no records for the SORTOUT data set.

RC16
specifies that DFSORT should issue message ICE206A, terminate, and give a return code of 16 when there are no records for the SORTOUT data set.

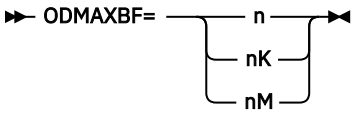
Note:

1. The return code of 0 or 4 set when there are no records for the SORTOUT data set can be overridden by a higher return code set for some other reason.
2. NULLOUT does not apply when SORTOUT is not present, when tape work data sets are specified for a sort application, or when the Blockset technique is not selected for a merge application. DFSORT does not check if there are no records for the SORTOUT data set in these cases.
3. NULLOUT applies to the SORTOUT data set. NULLOFL on the OUTFIL statement applies to OUTFIL data sets.
4. For an ICEGENER application, NULLOUT applies to the SYSUT2 data set if DFSORT copy is used. Note that ICEGENER passes back return code 12 instead of return code 16.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

ODMAXBF



Temporarily overrides the ODMAXBF installation option, which specifies the maximum buffer space DFSORT can use for each OUTFIL data set. The actual amount of buffer space used for a particular OUTFIL data set will not exceed the ODMAXBF limit, but can be less than the limit. OUTFIL processing is supported by the Blockset technique for sort, copy, and merge applications.

The storage used for OUTFIL processing is adjusted automatically according to the total storage available, the storage needed for non-OUTFIL processing, and the number of OUTFIL data sets and their attributes (for example, block size). OUTFIL processing is subject to the ODMAXBF limit in effect and the system storage limits (for example, IEFUSI), but not to the DFSORT storage limits (that is, SIZE, MAXLIM, and TMAXLIM). DFSORT attempts to use storage above 16MB virtual for OUTFIL processing whenever possible.

Lowering ODMAXBF below 2M can cause performance degradation for the application, but may be necessary if you consider the amount of storage used for OUTFIL processing to be a problem. Raising ODMAXBF can improve EXCPs for the application but can also increase the amount of storage needed.

n
specifies that a maximum of n bytes of buffer space is to be used for each OUTFIL data set. If you specify less than 262144, 262144 is used. If you specify more than 16777216, 16777216 is used.

Limit: 8 digits

nK
specifies that a maximum of n times 1024 bytes of buffer space is to be used for each OUTFIL data set. If you specify less than 256K, 256K is used. If you specify more than 16384K, 16384K is used.

Limit: 5 digits

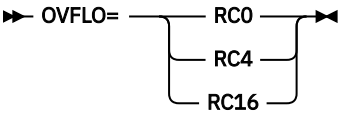
nM
specifies that a maximum of n times 1048576 bytes of buffer space is to be used for each OUTFIL data set. If you specify 0M, 256K is used. If you specify more than 16M, 16M is used.

Limit: 2 digits

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

OVFLO



Temporarily overrides the OVFLO installation option, which specifies the action to be taken by DFSORT when BI, FI, PD or ZD summary fields overflow.

RC0
specifies that DFSORT should issue message ICE152I (once), set a return code of 0 and continue processing when summary fields overflow. The pair of records involved in a summary overflow is left unsummed and neither record is deleted. Summary overflow does not prevent further summation.

RC4
specifies that DFSORT should issue message ICE152I (once), set a return code of 4 and continue processing when summary fields overflow. The pair of records involved in a summary overflow is left unsummed and neither record is deleted. Summary overflow does not prevent further summation.

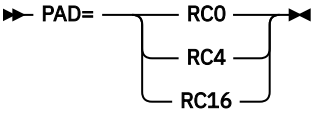
RC16
specifies that DFSORT should issue message ICE195A, terminate and give a return code of 16 when summary fields overflow.

Note: The return code of 0 or 4 set for summary overflow can be overridden by a higher return code set for some other reason.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

PAD



Temporarily overrides the PAD installation option, which specifies the action to be taken by DFSORT when the SORTOUT LRECL is larger than the SORTIN/SORTINnn LRECL, for the cases where DFSORT allows LRECL padding.

RC0
specifies that DFSORT should issue message ICE171I, set a return code of 0 and continue processing when the SORTOUT LRECL is larger than the SORTIN/SORTINnn LRECL.

RC4
specifies that DFSORT should issue message ICE171I, set a return code of 4 and continue processing when the SORTOUT LRECL is larger than the SORTIN/SORTINnn LRECL.

RC16
specifies that DFSORT should issue message ICE196A, terminate and give a return code of 16 when the SORTOUT LRECL is larger than the SORTIN/SORTINnn LRECL.

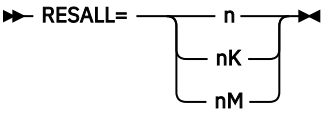
Note:

1. The return code of 0 or 4 set for LRECL padding can be overridden by a higher return code set for some other reason.
2. For an ICEGENER application, the GNPAD value is used and the PAD value is ignored.
3. For some LRECL padding situations (for example, a tape work data set sort), DFSORT issues ICE043A and terminates with a return code of 16. The PAD value has no effect in these cases.
4. DFSORT does not check for LRECL padding if:
   a. A SORTIN DD (sort/copy), SORTINnn DD (merge) or SORTOUT DD is not present
   b. A SORTIN DD (sort/copy), SORTINnn DD (merge) or SORTOUT DD specifies a VSAM data set.
5. DFSORT does not check OUTFIL data sets for LRECL padding.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

RESALL



Temporarily overrides the RESALL installation option, which specifies the number of bytes to be reserved in a REGION for system use. Usually, only 4K bytes (the standard default) of main storage must be available in a region for system use. However, in some cases, this may not be enough; for example, if your installation does not have BSAM/QSAM modules resident, you have user exits that open data sets, or you have COBOL exits. RESALL is used only when MAINSIZE/SIZE=MAX is in effect.

RESALL applies only to the amount of main storage below 16MB virtual. The ARESALL option applies to the amount of main storage above 16MB virtual.

n
specifies that n bytes of storage are to be reserved. If you specify less than 4096, 4096 is used.

Limit: 8 digits.

nK
specifies that n times 1024 bytes of storage are to be reserved. If you specify less than 4K, 4K is used.

Limit: 5 digits.

nM
specifies that n times 1048576 bytes of storage are to be reserved. If you specify 0M, 4K is used.

Limit: 2 digits.

Tip: A better way to reserve the required storage for user exits activated by the MODS statement is to use the **m** parameter of the MODS statement.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

RESET or NORESET

Temporarily overrides the RESET installation option, which specifies whether DFSORT should process a VSAM output data set defined with REUSE as a NEW or MOD data set.

RESET
specifies that DFSORT processes a VSAM output data set defined with REUSE as a NEW data set. The high-used RBA is reset to zero and the output data set is effectively treated as an initially empty cluster.

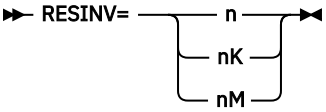
NORESET
specifies that DFSORT processes a VSAM output data set defined with REUSE as a MOD data set. The high-used RBA is not reset and the output data set is effectively treated as an initially non-empty cluster.

Note: A VSAM output data set defined without REUSE is processed as a MOD data set.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

RESINV



Temporarily overrides the RESINV installation option, which specifies the number of bytes to be reserved in a REGION for the invoking program. RESINV is used only when DFSORT is dynamically invoked and MAINSIZE/SIZE=MAX is in effect.

RESINV applies only to the amount of main storage below 16MB virtual. The ARESINV option applies to the amount of main storage above 16MB virtual.

This extra space is usually required for data handling by the invoking program or user exits while DFSORT is running (as is the case with some PL/I- and COBOL- invoked sort applications). Therefore, if your invoking program's user exits do not perform data set handling, you do not need to specify this parameter. The reserved space is not meant to be used for the invoking program's executable code.

The amount of space required depends upon what routines you have, how the data is stored, and which access method you use.

n
specifies that n bytes of storage are to be reserved.

Limit: 8 digits

nK
specifies that n times 1024 bytes of storage are to be reserved.

Limit: 5 digits

nM
specifies n times 1048576 bytes of main storage are to be reserved.

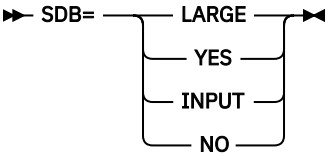
Limit: 2 digits.

Tip: A better way to reserve the required storage for user exits activated by the MODS statement is to use the *m* parameter of the MODS statement.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

SDB



Temporarily overrides the SDB installation option, which specifies whether DFSORT should use the system-determined optimum block size for output data sets when the block size is specified as zero or defaulted to zero. System-determined block size applies to both SMS-managed and non-SMS-managed data sets and results in the most efficient use of space for the device on which the output data set resides.

DFSORT can select system-determined optimum block sizes greater than 32760 bytes for tape output data sets.

If you want DFSORT to use system-determined block sizes for disk and tape output data sets, specify one of the following values:

- SDB=LARGE if you want to allow DFSORT to select tape output block sizes greater than 32760 bytes.
- SDB=YES (or its alias SDB=SMALL) if you want DFSORT to select tape output block sizes less than or equal to 32760 bytes.
- SDB=INPUT if you want to allow DFSORT to select tape output block sizes greater than 32760 bytes only when tape input data sets with block sizes greater than 32760 bytes are used.

DFSORT will not select a tape output block size greater than the BLKSZLIM in effect. In particular, if a default BLKSZLIM of 32760 is in effect, DFSORT will not select a tape output block size greater than 32760 bytes. Therefore, in order to allow DFSORT to select tape output block sizes greater than 32760 bytes for particular jobs, you may need to ensure that your JCL or data class supplies appropriately large BLKSZLIM values (for example, 1GB) for those jobs.

If you don't want DFSORT to use system-determined block sizes, specify SDB=NO (not recommended as an installation option).

LARGE
specifies that DFSORT is to use the system-determined optimum block size for an output data set when its block size is zero. SDB=LARGE allows DFSORT to select a block size greater than 32760 bytes for a tape output data set, when appropriate. A larger tape block size can improve elapsed time and tape utilization, but you must ensure that applications which subsequently use the resulting tape data set can handle larger block sizes.

DFSORT selects the system-determined optimum block size as follows:

- For a disk output data set, the optimum block size for the device used is selected based on the obtained or derived RECFM and LRECL for the output data set. The maximum block size for disk output data sets is 32760 bytes.
- For a tape output data set, the optimum block size is selected based on the obtained or derived RECFM and LRECL for the output data set, as shown in Table 41 on page 203.

Table 41. SDB=LARGE Block Sizes for Tape Output Data Sets

RECFM	BLKSIZE is set to:
F or FS	LRECL
FB or FBS	Highest possible multiple of LRECL that is less than or equal to the optimum block size for the device, subject to the BLKSZLIM in effect.
V, D, VS, or DS	LRECL + 4
VB, DB, VBS, or DBS	Optimum block size for the device, subject to the BLKSZLIM in effect.

DFSORT uses the system-determined optimum block size for the output data set in most cases when the block size is zero. However, the following conditions prevent DFSORT from using the system-determined block size:

- Output data set block size is available (that is, non-zero) in the JFCB (disk or tape) or format 1 DSCB (disk) or tape label (only for DISP=MOD with AL, SL, or NSL label, when appropriate)
- Output is a spool, dummy, VSAM, or unmovable data set, or a z/OS UNIX file.
- The output data set is on tape with a label type of AL
- DFSORT's Blockset technique is **not** selected.

In the previous cases, DFSORT uses the specified block size, or determines an appropriate (though not necessarily optimum) block size for the output data set. The selected block size is limited to 32760 bytes.

YES
specifies that DFSORT is to use the system-determined optimum block size for an output data set when its block size is zero, but is to limit the selected block size to a maximum of 32760 bytes. See the discussion of SDB=LARGE for more information; the only difference between SDB=LARGE and SDB=YES is that SDB=LARGE allows block sizes greater than 32760 bytes for tape output data sets, whereas SDB=YES does not.

INPUT
specifies that DFSORT is to use the system-determined optimum block size for an output data set when its block size is zero, but is to limit the selected block size to a maximum of 32760 bytes if the input block size is less than or equal to 32760 bytes. Thus, SDB=INPUT works like SDB=LARGE if the input block size is greater than 32760 bytes (only possible for tape input data sets) and works like SDB=YES if the input block size is less than or equal to 32760 bytes. See the discussions of SDB=LARGE and SDB=YES for more information.

NO
specifies that DFSORT is not to use the system-determined optimum block size. When the output data set block size is zero, DFSORT selects an appropriate (though not necessarily optimum) block size for the output data set based on the obtained or derived output or input attributes. SDB=NO limits the selected block sizes to a maximum of 32760 bytes.

SDB=NO works like SDB=YES if the input block size is greater than 32760 bytes (only possible for tape input data sets). See the discussion of SDB=YES for more information.

Note:

1. SDB=NO does not prevent the use of system-determined block size for the output data set at allocation or in other cases where the output data set block size is set before DFSORT gets control.
2. When DFSORT uses system-determined block size, the selected output data set block size may be different from the block size selected previously. Applications that require a specific output data set block size should be changed to specify that block size explicitly.
3. SDB and SDB=SMALL can be used instead of SDB=YES. NOSDB can be used instead of SDB=NO.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

SIZE

See FILSZ.

SKIPREC

Specifies the number of records *z* you want to skip (delete) before starting to sort or copy the input data set. SKIPREC is usually used if, on a preceding DFSORT run, you have processed only part of the input data set.

An application with an input data set that exceeds intermediate storage capacity usually terminates unsuccessfully. However, for a tape work data set sort, you can use a routine at E16 (as described in Chapter 5, “Using your own user exit routines,” on page 467) to instruct the program to sort only those records already read in. It then prints a message giving the number of records sorted. You can use SKIPREC in a subsequent sort run to bypass the previously-sorted records, sort only the remaining records, and then merge the output from different runs to complete the application.

z
specifies the number of records to be skipped.

Limit: 28 digits (15 significant digits)

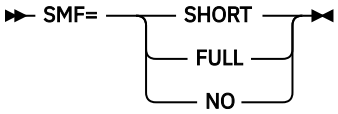
Note:

1. SKIPREC applies only to records read from SORTIN (not from E15 routines). (See Figure 2 on page 10.)
2. If SKIPREC=0 is in effect, SKIPREC is not used.
3. You may want to consider using the STARTREC parameter of the OUTFIL statement as an alternative to using SKIPREC.

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

SMF



Temporarily overrides the SMF installation option, which specifies whether a DFSORT SMF record is to be produced as described in *z/OS DFSORT Installation and Customization*.

SHORT
specifies that DFSORT is to produce a short SMF type-16 record for a successful run. The short SMF record does not contain record-length distribution statistics or data set sections.

FULL
specifies that DFSORT is to produce a full SMF type-16 record for a successful run. The full SMF record contains the same information as the short record, as well as record-length distribution and data set sections, as appropriate.

NO
specifies that DFSORT is not to produce an SMF type-16 record for this run.

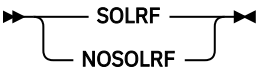
Note:

1. SMF is processed only if it is passed on the OPTION control statement in an extended parameter list or in DFSPARM.
2. SMF=FULL can degrade performance for a variable-length record application.
3. The DFSORT SVC is called to write SMF type-16 records. If SMF=SHORT or SMF=FULL is in effect, the correct DFSORT SVC for this release must be loaded in LPA or MLPA.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

SOLRF or NOSOLRF



Temporarily overrides the SOLRF installation option, which specifies whether DFSORT should set the SORTOUT LRECL to the reformatted record length when the SORTOUT LRECL is unknown.

SOLRF
specifies that DFSORT should use the reformatted record length for the SORTOUT LRECL when the SORTOUT LRECL is not specified or available. DFSORT will use one of the following for the SORTOUT LRECL, in the order listed:

1. The SORTOUT LRECL if available from the JFCB, format 1 DSCB, DFSMSrmm, ICETPEX, or tape label
2. The L3 length if specified in the RECORD statement
3. The OUTREC length if the OUTREC statement is specified
4. The INREC length if the INREC statement is specified
5. The L2 length if specified in the RECORD statement providing an E15 user exit is present
6. The SORTIN or SORTINnn LRECL if available from the JFCB, format 1 DSCB, DFSMSrmm, ICETPEX, or tape label
7. The L1 length in the RECORD statement

NOSOLRF
specifies that DFSORT should not use the reformatted record length for the SORTOUT LRECL. DFSORT will use one of the following for the SORTOUT LRECL, in the order listed:

1. The SORTOUT LRECL if available from the JFCB, format 1 DSCB, DFSMSrmm, ICETPEX, or tape label
2. The L3 length if specified in the RECORD statement providing an E35 exit, OUTREC statement or INREC statement is present
3. The L2 length if specified in the RECORD statement providing an E15 user exit is present
4. The SORTIN or SORTINnn LRECL if available from the JFCB, format 1 DSCB, DFSMSrmm, ICETPEX, or tape label
5. The L1 length in the RECORD statement

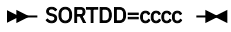
Note:

1. With SOLRF in effect (the IBM-supplied default), DFSORT sets the SORTOUT LRECL to the INREC or OUTREC record length when appropriate, which is usually what you want when you use INREC or OUTREC. If you want DFSORT to use the input length for the SORTOUT LRECL even when INREC or OUTREC is present, you can use NOSOLRF, but be aware that this can cause padding or truncation of the reformatted records, or termination.
2. CAOUTREC can be used instead of SOLRF.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

SORTDD



Specifies a four-character prefix for the ddnames to be used when you dynamically invoke DFSORT more than once in a program step. The four characters replace "SORT" in the following ddnames: SORTIN, SORTOUT, SORTINn, SORTINnn, SORTOFd, SORTOFdd, SORTWKd, SORTWKdd, SORTJNF1, SORTJNF2 and SORTCNTL. This allows you to use a different set of ddnames for each call to DFSORT.

cccc
Specifies a four-character prefix. The four characters must all be alphanumeric or national ($, #, or @). The first character must be alphabetic. The first three characters must not be SYS.

For example, if you use ABC# as replacement characters, DFSORT uses DD statements ABC#IN, ABC#CNTL, ABC#WKdd, and ABC#OUT instead of SORTIN, SORTCNTL, SORTWKdd, and SORTOUT.

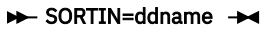
Note:

1. SORTDD is processed only if it is passed on the OPTION control statement in an extended parameter list, or in DFSPARM.
2. If both SORTIN=ddname and SORTDD=cccc are specified, ddname is used for DFSORT input.
3. If both SORTOUT=ddname and SORTDD=cccc are specified, ddname is used for DFSORT output.
4. If both MERGEIN=(ddname1,ddname2,...) and SORTDD=cccc are specified, ddname1, ddname2, and so on are used for DFSORT input.

Default: SORT. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

SORTIN



Specifies a ddname to be associated with the SORTIN data set. This allows you to dynamically invoke DFSORT more than once in a program step, passing a different ddname for each input data set.

The ddname can be 1 through 8 characters, but must be unique within the job step. Do *not* use ddnames reserved for use by DFSORT, such as **cccc**WK**d**, **cccc**WK**dd**, **cccc**DK**d**, or **cccc**DK**dd** , where **cccc** is the specified or defaulted value for the SORTDD operand and **d** is any character.

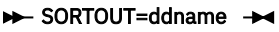
Note:

1. SORTIN is processed only if it is passed on the OPTION control statement in an extended parameter list, or in DFSPARM.
2. If both SORTIN=ddname and SORTDD=cccc are specified, ddname is used for the input file. The same ddname cannot be specified for SORTIN and SORTOUT, or for MERGEIN and SORTOUT.
3. If SORTIN is used for a tape work data set sort, DFSORT terminates.

Default: SORTIN, unless SORTDD=cccc is specified in which case ccccIN is the default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

SORTOUT



Specifies a ddname to be associated with the SORTOUT data set or with an OUTFIL data set for which FNAMES and FILES are not specified. This allows you to dynamically invoke DFSORT more than once in a program step, passing a different ddname for each output data set.

The ddname can be 1 through 8 characters, but must be unique within the job step. Do *not* use ddnames reserved for use by DFSORT, such as **cccc**WK**d**, **cccc**WK**dd**, **cccc**DK**d**, or **cccc**DK**dd** , where **cccc** is the specified or defaulted value for the SORTDD operand and **d** is any character.

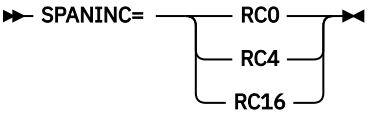
Note:

1. SORTOUT is processed only if it is passed on the OPTION control statement in an extended parameter list or in DFSPARM.
2. If both SORTOUT=ddname and SORTDD=cccc are specified, ddname is used for the output file. The same ddname cannot be specified for SORTIN and SORTOUT, or for MERGEIN and SORTOUT.
3. If SORTOUT is specified for a conventional merge or for a tape work data set sort, DFSORT terminates.

Default: SORTOUT, unless SORTDD=cccc is specified, in which case ccccOUT is the default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

SPANINC



Temporarily overrides the SPANINC installation option, which specifies the action to be taken by DFSORT when one or more incomplete spanned records are detected in a variable spanned input data set.

RC0
specifies that DFSORT should issue message ICE197I (once), set a return code of 0 and eliminate all incomplete spanned records it detects. Valid records will be recovered.

RC4
specifies that DFSORT should issue message ICE197I (once), set a return code of 4 and eliminate all incomplete spanned records it detects. Valid records will be recovered.

RC16
specifies that DFSORT should issue message ICE204A, terminate and give a return code of 16 when an incomplete spanned record is detected.

Note:

1. The return code of 0 or 4 set for incomplete spanned records can be overridden by a higher return code set for some other reason.
2. In cases where a spanned record cannot be properly assembled (for example, it has a segment length less than 4 bytes), DFSORT issues ICE141A and terminates with a return code of 16. The SPANINC value has no effect in these cases.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

STOPAFT

Specifies the maximum number of records (n) you want accepted for sorting or copying (that is, read from SORTIN or inserted by E15 and not deleted by SKIPREC, E15, or the INCLUDE/OMIT statement). When n records have been accepted, no more records are read from SORTIN; E15 continues to be entered as if EOF were encountered until a return code of 8 is sent, but no more records are inserted. If end-of-file is encountered before n records are accepted, only those records accepted up to that point are sorted or copied.

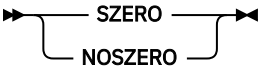
n
specifies the maximum number of records to be accepted.

Limit: 28 digits (15 significant digits)

Default: None; optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

SZERO or NOSZERO



Temporarily overrides the SZERO installation option, which specifies whether DFSORT should treat numeric -0 and +0 values as signed (that is, different) or unsigned (that is, the same) for collation, comparisons, editing and conversions, minimums and maximums. The following DFSORT control statements are affected by this option: INCLUDE, INREC, MERGE, OMIT, OUTFIL, OUTREC and SORT.

SZERO
specifies that DFSORT should treat numeric zero values as signed. -0 and +0 are treated as **different** values, that is, -0 is treated as a negative value and +0 is treated as a positive value. SZERO affects DFSORT processing of numeric values as follows:

- For collation of SORT and MERGE fields, -0 collates before +0 in ascending order and after +0 in descending order.
- For comparisons of INCLUDE, OMIT, and OUTFIL compare fields and constants, -0 compares as less than +0.
- For editing and conversions of INREC, OUTREC, and OUTFIL reformatting fields, decimal constants, and the results of arithmetic expressions, -0 is treated as negative and +0 is treated as positive.
- For minimums and maximums of OUTFIL TRAILERx fields, -0 is treated as negative and +0 is treated as positive.

NOSZERO
specifies that DFSORT should treat numeric zero values as unsigned. -0 and +0 are treated as **the same** value, that is, -0 and +0 are both treated as positive values. NOSZERO affects DFSORT processing of numeric values as follows:

- For collation of SORT and MERGE fields, -0 collates equally with +0.
- For comparisons of INCLUDE, OMIT and OUTFIL compare fields and constants, -0 compares as equal to +0.
- For editing and conversions of INREC, OUTREC, and OUTFIL reformatting fields, decimal constants, and the results of arithmetic expressions, -0 and +0 are treated as positive.
- For minimums and maximums of OUTFIL TRAILERx fields, -0 and +0 are treated as positive.

Note: OPTION SZERO or OPTION NOSZERO is ignored for OUTFIL INCLUDE and OMIT, and INREC, OUTREC, and OUTFIL WHEN, BEGIN and END, if the OPTION statement is "found" **after** the INREC, OUTREC, or OUTFIL statement. To avoid this, specify SZERO or NOSZERO as an EXEC/

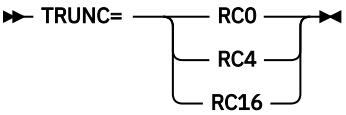
DFSPARM PARM option, or in an OPTION statement **before** the INREC, OUTREC, or OUTFIL statement in the same source, for example:

```
//SYIN DD *
 OPTION NOSZERO,COPY
 OUTFIL INCLUDE=(...)
/*
```

Default: Usually, the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

TRUNC



Temporarily overrides the TRUNC installation option, which specifies the action to be taken by DFSORT when the SORTOUT LRECL is smaller than the SORTIN/SORTINnn LRECL, for the cases where DFSORT allows LRECL truncation.

RC0
specifies that DFSORT should issue message ICE171I, set a return code of 0 and continue processing when the SORTOUT LRECL is smaller than the SORTIN/SORTINnn LRECL.

RC4
specifies that DFSORT should issue message ICE171I, set a return code of 4 and continue processing when the SORTOUT LRECL is smaller than the SORTIN/SORTINnn LRECL.

RC16
specifies that DFSORT should issue message ICE196A, terminate and give a return code of 16 when the SORTOUT LRECL is smaller than the SORTIN/SORTINnn LRECL.

Note:

1. The return code of 0 or 4 set for LRECL truncation can be overridden by a higher return code set for some other reason.
2. For an ICEGENER application, the GNTRUNC value is used and the TRUNC value is ignored.
3. For some LRECL truncation situations (for example, a tape work data set sort), DFSORT issues ICE043A and terminates with a return code of 16. The TRUNC value has no effect in these cases.
4. DFSORT does not check for LRECL truncation if:
   a. A SORTIN DD (sort/copy), SORTINnn DD (merge) or SORTOUT DD is not present
   b. A SORTIN DD (sort/copy), SORTINnn DD (merge) or SORTOUT DD specifies a VSAM data set.
5. DFSORT does not check OUTFIL data sets for LRECL truncation.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

USEWKDD

Temporarily overrides the DYNAUTO=IGNWKDD option, which specifies that dynamic work data sets are used even if SORTWKdd DD statements are present. This option allows JCL SORTWKdd data sets to be used rather than deallocated.

Note: USEWKDD is processed only if it is passed on the OPTION control statement in an extended parameter list or in DFSPARM.

Default: None, optional. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Function: See Appendix B, “Specification/override of DFSORT options,” on page 805.

VERIFY or NOVERIFY

Temporarily overrides the VERIFY installation option, which specifies whether sequence checking of the final output records must be performed.

VERIFY
specifies that sequence checking is performed.

NOVERIFY
specifies that sequence checking is not performed.

Note:

1. Using VERIFY can degrade performance.
2. SEQ=YES can be used instead of VERIFY, SEQ=NO can be used instead of NOVERIFY.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

VLLONG or NOVLLONG

Temporarily overrides the VLLONG installation option, which specifies whether DFSORT is to truncate "long" variable-length output records. A long output record is one whose length is greater than the LRECL of the SORTOUT or OUTFIL data set it is to be written to.

VLLONG is not meaningful for fixed-length output record processing.

VLLONG
specifies that DFSORT truncates long variable-length output records to the LRECL of the SORTOUT or OUTFIL data set.

NOVLLONG
specifies that DFSORT terminates if a long variable-length output record is found.

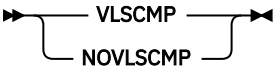
Note:

1. VLLONG should not be used unless you want the data at the end of long variable-length output records to be truncated for your DFSORT application; inappropriate use of VLLONG can result in unwanted loss of data.
2. VLLONG can be used to truncate long OUTFIL data records, but has no effect on long OUTFIL header or trailer records.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

VLSCMP or NOVLSCMP



Temporarily overrides the VLSCMP installation option, which specifies whether DFSORT is to pad "short" variable-length INCLUDE/OMIT compare fields with binary zeros. A short field is one where the variable-length record is too short to contain the entire field, that is, the field extends beyond the record. VLSCMP and NOVLSCMP apply to the INCLUDE and OMIT statements and to the INCLUDE and OMIT parameters of the OUTFIL statement.

The compare fields are only padded temporarily for testing; they are not actually changed for output.

VLSCMP is not meaningful for fixed-length record processing.

The settings for VLSCMP/NOVLSCMP and VLSHRT/NOVLSHRT provide three levels of processing for short INCLUDE/OMIT fields in the following hierarchy:

1. VLSCMP allows all of the INCLUDE/OMIT comparisons to be performed even if some fields are short. Because short fields are padded with binary zeros, comparisons involving short fields are false (unless a test against binary zero is relevant, as discussed later in this section). Comparisons involving non-short fields can be true or false.
2. NOVLSCMP and VLSHRT treat the entire INCLUDE/OMIT logical expression as false if any field is short. Thus comparisons involving non-short fields are ignored if any comparison involves a short field.
3. NOVLSCMP and NOVLSHRT result in termination if any field is short.

To illustrate how this works, suppose the following INCLUDE statement is used:

```
INCLUDE COND=(6,1,CH,EQ,C'1',OR,70,2,CH,EQ,C'T1')
```

If a variable-length input record has a length less than 71 bytes, the field at bytes 70-71 is short and the following occurs:

- With VLSCMP, the record is included if byte 6 of the input record is C'1' or omitted if byte 6 is not C'1'. The comparison of bytes 70-71 equal to C'T1' is false because bytes 70-71 contain either X'hh00' (for a record length of 70 bytes) or X'0000' (for a record length of less than 70 bytes). The comparison involving the non-short field is performed even though a short field is present.
- With NOVLSCMP and VLSHRT, the record is omitted because any short field makes the entire logical expression false. The comparison involving the non-short field is not performed because a short field is present.
- With NOVLSCMP and NOVLSHRT, DFSORT terminates because any short field results in termination.

In general, comparisons involving short fields are false with VLSCMP. However, if a binary zero value is relevant to the comparison, the use of binary zeros for padding might make the comparison true. For example, suppose the following INCLUDE statement is used:

```
INCLUDE COND=(21,2,CH,EQ,C'JX',OR,
              (55,2,CH,EQ,58,2,CH,AND,
               70,1,BI,LT,X'08'))
```

If a variable-length input record has a length less than 70 bytes, the field at byte 70 is short and is padded to X'00'. This makes the comparison of byte 70 less than X'08' true even though byte 70 is a short field and so probably irrelevant.

Likewise, if a variable-length record has a length less than 55 bytes, the fields at bytes 55-56 and 58-59 are short and are each padded to X'0000', and the field at byte 70 is short and is padded to X'00'. This makes the comparison of bytes 55-56 equal to 58-59 true and the comparison of byte 70 less than X'08' true even though all three fields are short and probably irrelevant.

In such cases where padding of short fields with binary zeros may result in unwanted true comparisons, you can get the result you want by adding an appropriate check of the record length to the INCLUDE/OMIT logical expression, such as:

```
INCLUDE COND=(21,2,CH,EQ,C'JX',OR,
              (1,2,BI,GE,X'0046',AND,
               55,2,CH,EQ,58,2,CH,AND,
               70,1,BI,LT,X'08'))
```

Now the comparisons involving bytes 55-56, 58-59 and 70 can only be true for records that are 70 bytes (X'0046') or longer. Thus, the irrelevant comparisons involving short fields are eliminated.

Keep in mind that short compare fields are padded with zeros when VLSCMP is in effect and code your INCLUDE/OMIT logical expressions to allow for that or even take advantage of it.

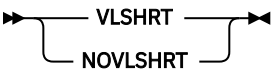
VLSCMP
specifies that short variable-length compare fields are padded with binary zeros.

NOVLSCMP
specifies that short variable-length compare fields are not padded.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

VLSHRT or NOVLSHRT



Temporarily overrides the VLSHRT installation option, which specifies whether DFSORT is to continue processing if a "short" variable-length SORT/MERGE control field, INCLUDE/OMIT compare field, or SUM summary field is found. A short field is one where the variable-length record is too short to contain the entire field, meaning that the field extends beyond the record. VLSHRT applies to the SORT, MERGE, INCLUDE, OMIT and SUM statements, and to the INCLUDE and OMIT parameters of the OUTFIL statement.

VLSHRT processing is not meaningful for fixed-length record processing.

The way in which DFSORT processes short INCLUDE/OMIT compare fields depends on the settings for VLSCMP/NOVLSCMP and VLSHRT/NOVLSHRT. For details, see the discussion of the VLSCMP and NOVLSCMP options.

VLSHRT
specifies that DFSORT continues processing if a short control field, compare field or summary field is found.

NOVLSHRT
specifies that DFSORT terminates if a short control field, compare field or summary field is found.

Note:

1. VLSHRT is not used if an INREC or OUTREC statement is specified, if you have an EFS01 or EFS02 routine, or if locale processing is used for SORT or MERGE fields. Note that none of these situations prevents the use of VLSCMP.
2. Unlike the OUTREC statement, the OUTREC , BUILD, OVERLAY, FINDREP or IFTHEN parameter of the OUTFIL statement does not force NOVLSHRT. Thus, you can use VLSHRT with OUTFIL to eliminate records with the INCLUDE or OMIT parameter and reformat the remaining records with the OUTREC, , BUILD, OVERLAY, FINDREP or IFTHEN parameter. If a short OUTFIL OUTREC or BUILD field is found, DFSORT terminates (even if VLSHRT is in effect) unless the VLFILL=byte parameter of OUTFIL is specified. If a short OUTFIL OVERLAY, FINDREP or IFTHEN field is found, DFSORT pads the missing bytes with blanks so it can be processed.

3. If VLSHRT is in effect and Blockset is selected:
   - DFSORT pads short SORT or MERGE control fields with binary zeros, thus making the order predictable for records with equal control fields of different lengths. The control fields are only padded temporarily for collation; they are not actually changed for output. Padding may increase the amount of work space required.
   - Records with short SUM summary fields are excluded from summation; that is, if either one of a pair of records being summed has a short SUM field, the records are left unsummed and neither record is deleted.
4. If VLSHRT is in effect and Blockset is not selected:
   - DFSORT terminates if the first byte of the first (major) SORT or MERGE control field is not included in the record.
   - DFSORT does not pad short SORT or MERGE control fields, thus making the order unpredictable for records with equal control fields of different lengths.
   - In certain cases, VLSHRT is not used because of the number and position of the SORT or MERGE control fields.
   - EQUALS is not used.

   Tip: You can use a SORTDIAG DD statement to force message ICE800I, which gives a code indicating why Blockset could not be used.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

VSAMEMT or NVSAMEMT

Temporarily overrides the VSAMEMT installation option, which specifies whether DFSORT should accept an empty VSAM input data set.

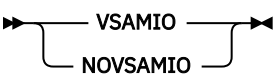
VSAMEMT
specifies that DFSORT accepts an empty VSAM input data set and processes it as having zero records.

NVSAMEMT
specifies that DFSORT terminates if an empty VSAM input data set is found.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

VSAMIO or NOVSAMIO



Temporarily overrides the VSAMIO installation option, which specifies whether DFSORT should allow a VSAM data set defined with REUSE to be sorted in-place.

VSAMIO
specifies that DFSORT can use the same VSAM data set for input and output when all of the following conditions are met:

- The application is a sort.

- RESET is in effect.
- The VSAM data set was defined with REUSE.

These conditions ensure that the VSAM data set is processed as NEW for output and will contain the sorted input records, that is, it will be sorted in-place.

DFSORT terminates if the same VSAM data set is specified for input and output and any of the previous conditions are not met.

NOVSAMIO
specifies that DFSORT terminates if the same VSAM data set is used for input and output.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

WRKREL or NOWRKREL

Temporarily overrides the WRKREL installation option, which specifies whether unused temporary SORTWKdd data set space will be released.

WRKREL
specifies that unused space is released.

NOWRKREL
specifies that unused space is not released.

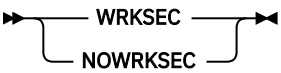
Note:

1. If you have dedicated certain volumes for SORTWKdd data sets, and you do not want unused temporary space to be released, you should specify NOWRKREL.
2. If WRKREL is in effect, DFSORT releases space for the SORTWKdd data sets just prior to termination. Space is released only for those SORTWKdd data sets that were used for the sort application.
3. RLS=0 can be used instead of NOWRKREL. RLS=n (n greater than 0) can be used instead of WRKREL.

Default: Usually the installation default. See Appendix B, "Specification/override of DFSORT options," on page 805 for full override details.

Applicable Functions: See Appendix B, "Specification/override of DFSORT options," on page 805.

WRKSEC or NOWRKSEC



Temporarily overrides the WRKSEC installation option, which specifies whether DFSORT uses automatic secondary allocation for temporary JCL SORTWKdd data sets.

WRKSEC
specifies that automatic secondary allocation for temporary JCL SORTWKdd data sets is used and that 25 percent of the primary allocation will be used as the secondary allocation.

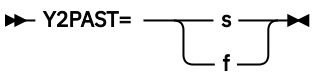
NOWRKSEC
specifies that automatic secondary allocation for temporary JCL SORTWKdd data sets is not used.

Note: SEC=0 can be used instead of NOWRKSEC. SEC=n (n greater than 0) can be used instead of WRKSEC.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

Y2PAST



Temporarily overrides the Y2PAST installation option, which specifies the sliding (s) or fixed (f) century window. The century window is used with DFSORT's Y2 formats to correctly interpret two-digit year data values as four-digit year data values.

s
specifies the number of years DFSORT is to subtract from the current year to set the beginning of the sliding century window. Because the Y2PAST value is subtracted from the current year, the century window slides as the current year changes. For example, Y2PAST=81 would set a century window of 1925-2024 in 2006 and 1926-2025 in 2007. s must be a value between 0 and 100.

f
specifies the beginning of the fixed century window. For example, Y2PAST=1962 would set a century window of 1962-2061. f must be a value between 1000 and 3000.

Note: CENTURY=value and CENTWIN=value can be used instead of Y2PAST=value.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

ZDPRINT or NZDPRINT

Temporarily overrides the ZDPRINT installation option, which specifies whether positive zoned-decimal (ZD) fields resulting from summing must be converted to printable numbers (that is, whether the zone of the last digit should be changed from a hexadecimal C to a hexadecimal F). See “SUM control statement” on page 433 for further details on the use of ZDPRINT and NZDPRINT.

ZDPRINT
means convert positive ZD summation results to printable numbers. For example, change hexadecimal F3F2C5 (prints as 32E) to F3F2F5 (prints as 325).

NZDPRINT
means do not convert positive ZD summation results to printable numbers.

Note: ZDPRINT=YES can be used instead of ZDPRINT. ZDPRINT=NO can be used instead of NZDPRINT.

Default: Usually the installation default. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Function: See Appendix B, “Specification/override of DFSORT options,” on page 805.

ZSORT or NOZSORT

Temporarily overrides the ZSORT installation option, which specifies IBM Integrated Accelerator for Z Sort can be used for sorting.

ZSORT
means IBM Integrated Accelerator for Z Sort is requested and eligible sort workloads will be taking advantage of sort accelerator.

NOZSORT
means IBM Integrated Accelerator for Z Sort is disabled and the sorting is done using the traditional techniques.

Default: NOZSORT.

Aliases for OPTION statement options

For compatibility reasons, the following OPTION statement options can be specified by using the aliases listed in Table 42 on page 216. See the indicated OPTION statement options for complete details.

Table 42. Aliases for OPTION Statement Options

Aliases for OPTION Statement OptionsAlias	OPTION Statement Option
CAOUTREC	SOLRF
CENTURY=value	Y2PAST=value
CENTWIN=value	Y2PAST=value
CHKPT	CKPT
CORE=value	MAINSIZE=value
HIPRLIM=value	HIPRMAX=value
L5=value	AVGRLEN=value
NOSDB	SDB=NO
PRINT=value	MSGPRT=value
RLS=n	WRKREL
RLS=0	NOWRKREL
SDB	SDB=YES
SDB=SMALL	SDB=YES
SEC=n	WRKSEC
SEC=0	NOWRKSEC
SEQ=YES	VERIFY
SEQ=NO	NOVERIFY
ZDPRINT=YES	ZDPRINT
ZDPRINT=NO	NZDPRINT

Specifying DFSORT options or COPY—examples

Example 1

```
SORT FIELDS=(1,20,CH,A)
OPTION SIZE=50000,SKIPREC=5,EQUALS,DYNALLOC
```

FIELDS
The control field begins on the first byte of each record in the input data set, is 20 bytes long, contains character data, and is to be sorted in ascending order.

SIZE
The data set to be sorted contains 50000 records.

SKIPREC
Five records are skipped (deleted) before starting to process the input data set.

EQUALS
The sequence of records that collate identically is preserved from input to output.

DYNALLOC
Two data sets (by default) are allocated on SYSDA (by default). The space on the data set is calculated using the SIZE value in effect.

Example 2

```
SORT   FIELDS=(1,2,CH,A),CKPT
OPTION EQUALS,NOCHALT,NOVERIFY,CHECK
```

FIELDS
The control field begins on the first byte of each record in the input data set, is 2 bytes long, contains character data, and is to be sorted in ascending order.

CKPT
DFSORT takes checkpoints during this run.

Note: CKPT is ignored if the Blockset technique is used. If checkpoints are required, you must bypass the Blockset technique by specifying the NOBLKSET option, or by specifying installation option IGNCKPT=NO. However, functions such as OUTFIL, which are supported only by the Blockset technique, cannot be used if the Checkpoint/Restart facility is used.

EQUALS
The sequence of records that collate identically is preserved from input to output.

NOCHALT
Only AQ fields are translated through the ALTSEQ translate table. If installation option CHALT=YES was specified, then NOCHALT temporarily overrides it.

NOVERIFY
No sequence check is performed on the final output records.

CHECK
The record count is checked at the end of program processing.

Example 3

```
OPTION FILSZ=50,SKIPREC=5,DYNALLOC=3390
SORT  FIELDS=(1,2,CH,A),SKIPREC=1,SIZE=200,DYNALLOC=(3380,5)
```

This example shows how parameters specified on the OPTION control statement override those specified on the SORT control statement, regardless of the order of the two statements.

FILSZ
DFSORT expects 50 records on the input data set. (Note that there is a difference in meaning between FILSZ and SIZE and that the OPTION specification of FILSZ is used in place of SIZE.)

SKIPREC
DFSORT causes five records from the beginning of the input file to be skipped. (SKIPREC=1 on the SORT statement is ignored.)

DYNALLOC
DFSORT allocates two work data sets (by default) on an IBM 3390.

FIELDS
The control field begins on the first byte of each record in the input data set, is 2 bytes long, contains character data, and is to be sorted in ascending order.

Example 4

```
OPTION NOBLKSET
```

NOBLKSET
DFSORT does not use the Blockset technique for a sort or merge.

Example 5

```
OPTION STOPAFT=100
```

STOPAFT
DFSORT accepts 100 records before sorting or copying.

Example 6

```
OPTION RESINV=32000,MSGPRT=NONE,
       MSGDDN=SORTMSGS,SORTDD=ABCD,SORTIN=MYINPUT,
       SORTOUT=MYOUTPUT,NOLIST
```

This example illustrates the parameters RESINV, MSGPRT, MSGDDN, SORTDD, SORTIN, SORTOUT, and NOLIST, and the actions taken when these parameters are supplied on an OPTION statement read from the SYSIN data set or the SORTCNTL data set. The parameters are recognized, but not used.

RESINV
32000 bytes of storage are reserved for the user.

MSGPRT=NONE
The keyword is ignored, and messages are printed according to the installation default.

MSGDDN=SORTMSGS
The keyword is ignored, and all messages are written to the SYSOUT data set.

SORTDD=ABCD
The keyword is ignored, and the standard prefix SORT is used.

SORTIN=MYINPUT
The keyword is ignored, and the ddname SORTIN is used to reference the input data set.

SORTOUT=MYOUTPUT
The keyword is ignored, and the ddname SORTOUT is used to reference the output data set.

NOLIST
The keyword is ignored, and control statements are printed according to the installation defaults.

Example 7

```
OPTION RESINV=32000,MSGPRT=CRITICAL
       MSGDDN=SORTMSGS,SORTDD=ABCD,SORTIN=MYINPUT,
       SORTOUT=MYOUTPUT,NOLIST
```

This example illustrates keywords RESINV, MSGPRT, MSGDDN, SORTDD, SORTIN, SORTOUT, and NOLIST and the actions taken when these keywords are supplied on the OPTION control statement passed by DFSPARM. These options can also be passed in an extended parameter list, but must be coded as one contiguous statement without continuation lines.

RESINV
32000 bytes of storage are reserved for the user.

MSGPRT=CRITICAL
Only critical messages are printed on the message data set.

MSGDDN=SORTMSGS
Messages are written to the SORTMSGS data set.

SORTDD=ABCD
SORT uses ABCD as a prefix for all sort names.

SORTIN=MYINPUT
The ddname MYINPUT is used to reference the input data set.

SORTOUT=MYOUTPUT
The ddname MYOUTPUT is used to reference the output data set.

NOLIST
Control statements are not printed.

Example 8

```
SORT   FIELDS=(3,4,CH,A)
OPTION COPY,SKIPREC=10,CKPT
MODS E15=(E15,1024,MODLIB),E35=(E35,1024,MODLIB)
```

SORT
The sort statement is ignored because the COPY option has been specified.

COPY
The copy processing is always done on a record-by-record basis. Each record is therefore read from SORTIN, passed to the E15 exit, passed to the E35 exit, and written to SORTOUT. (Contrast this with a sort, where all the records are read from SORTIN and passed to the E15 exit before any records are passed to the E35 exit and written to SORTOUT.)

SKIPREC
Ten records are skipped before copying starts.

CKPT
The checkpoint option is not used for copy applications.

Example 9

```
SORT   FIELDS=(5,4,CH,A)
SUM FIELDS=(12,5,ZD,25,6,ZD)
OPTION ZDPRINT
```

ZDPRINT
The positive summed ZD values are printable because DFSORT uses an F sign for the last digit.

Example 10

```
//S1 EXEC  PGM=SORT
//SYSOUT    DD  SYSOUT=*
//SARA DD *
AAA FROM SARA
CCC FROM SARA
DDD FROM SARA
//MOLLY DD *
AAA FROM MOLLY
BBB FROM MOLLY
DDD FROM MOLLY
//NORA DD *
AAA FROM NORA
BBB FROM NORA
CCC FROM NORA
//SORTOUT DD SYSOUT=*
//DFSPARM  DD    *
 OPTION EQUALS,MERGEIN=(NORA,SARA,MOLLY)
 MERGE FIELDS=(1,3,CH,A)
/*
```

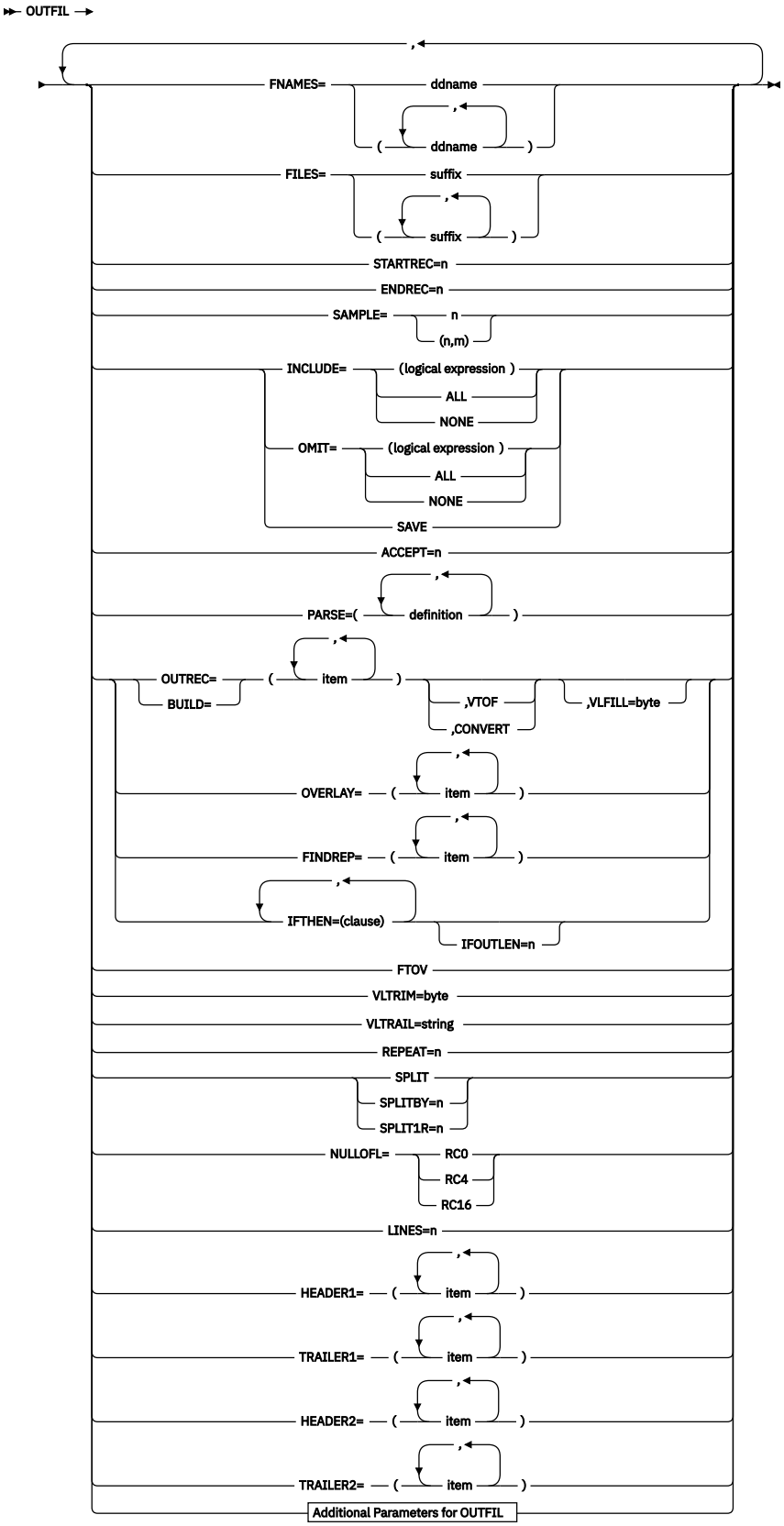
This example illustrates the use of the alternate ddnames NORA, SARA and MOLLY for a MERGE application instead of SORTINnn ddnames. Since EQUALS is specified, equally collating records will be from NORA, then SARA, then MOLLY, that is, in the order specified in the MERGEIN list. Thus, SORTOUT contains these records:

```
AAA FROM NORA
AAA FROM SARA
AAA FROM MOLLY
BBB FROM NORA
BBB FROM MOLLY
CCC FROM NORA
CCC FROM SARA
DDD FROM SARA
DDD FROM MOLLY
```

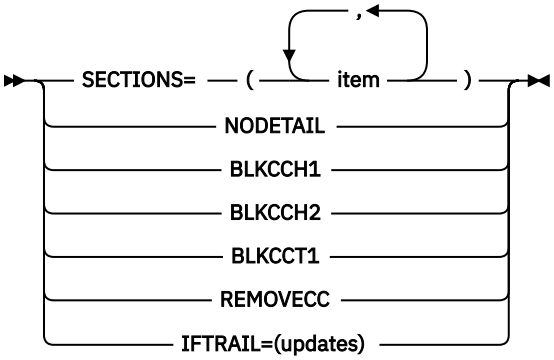
If MERGEIN=(SARA,MOLLY,NORA) was specified in the MERGEIN list in DFSPARM, SORTOUT would contain these records:

```
AAA FROM SARA
AAA FROM MOLLY
AAA FROM NORA
BBB FROM MOLLY
BBB FROM NORA
CCC FROM SARA
CCC FROM NORA
DDD FROM SARA
DDD FROM MOLLY
```

OUTFIL control statements



Additional Parameters for OUTFIL



OUTFIL control statements allow you to create one or more output data sets for a sort, copy, or merge application from a single pass over one or more input data sets. You can use multiple OUTFIL statements, with each statement specifying the OUTFIL processing to be performed for one or more output data sets. OUTFIL processing begins after all other processing ends (that is, after processing for exits, options, and other control statements).

OUTFILE can be used as an alias for OUTFIL.

OUTFIL statements support a wide variety of output data set tasks, including:

- Creation of multiple output data sets containing unedited or edited records from a single pass over one or more input data sets.
- Creation of multiple output data sets containing different ranges or subsets of records from a single pass over one or more input data sets. In addition, records that are not selected for any subset can be saved in a separate output data set.
- Conversion of variable-length record data sets to fixed-length record data sets.
- Conversion of fixed-length record data sets to variable-length record data sets.
- A wide variety of parsing, editing, and reformatting tasks including:
  - The use of fixed position/length fields or variable position/length fields. For fixed fields, you specify the starting position and length of the field directly. For variable fields, such as delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings (and many other types), you define rules that allow DFSORT to extract the relevant data into fixed parsed fields, and then use the parsed fields as you would use fixed fields.
  - Insertion of blanks, zeros, strings, current date, future date, past date, current time, sequence numbers, decimal constants, and the results of arithmetic expressions before, between, and after the input fields in the reformatted records.
  - Sophisticated conversion capabilities, such as find and replace, hexadecimal display, bit display, translation of EBCDIC letters from lowercase to uppercase or uppercase to lowercase, translation of characters from EBCDIC to ASCII and from ASCII to EBCDIC, translation of characters using the ALTSEQ translation table, conversion of numeric values from one format to another, left justify or left-squeeze (remove leading blanks or all blanks and shift left), and right-justify or right-squeeze (remove trailing blanks or all blanks and shift right).
  - Sophisticated editing capabilities, such as control of the way numeric fields are presented with respect to length, leading or suppressed zeros, thousands separators, decimal points, leading and trailing positive and negative signs, and so on.
  - Twenty-seven pre-defined editing masks are available for commonly used numeric editing patterns, encompassing many of the numeric notations used throughout the world. In addition, a virtually unlimited number of numeric editing patterns are available via user-defined editing masks.
  -
  - Transformation of SMF, TOD, and ETOD date and time values to more usable forms.

  - Conversion of input date fields of one type (CH, ZD, PD, 2-digit year, 4-digit year, Julian, Gregorian) to corresponding output date fields of another type or to a corresponding day of the week.
  - Various types of arithmetic operations for input date fields.
  - Selection of a character constant, hexadecimal constant, or input field from a lookup table for output, based on a character, hexadecimal, or bit string as input (that is, lookup and change).
- Creation of the reformatted records in one of the following ways:
  - By building the entire record one item at a time.
  - By only overlaying specific columns.
  - By performing find and replace operations.
  - By using sophisticated conditional logic or group operations to choose how different records are reformatted.
- Highly detailed three-level (report, page, and section) reports containing a variety of report elements you can specify (for example, current date, current time, edited or converted page numbers, character strings, and blank lines) or derive from the input records (for example, character fields; unedited, edited or converted numeric input fields; edited or converted record counts; and edited or converted totals, maximums, minimums, and averages for numeric input fields).
- Creation of multiple output records from each input record, with or without intervening blank output records.
- Repetition and sampling of data records.
- Splitting of data records in rotation among a set of output data sets.
- Updating counts and totals in an existing trailer (last) record based on the current data records.

The parameters of OUTFIL are grouped by primary purpose as follows:

-
- **FNAMES** and **FILES** specify the ddnames of the **OUTFIL data sets** to be created. Each OUTFIL data set to be created must be specifically identified using FNAMES or FILES in an OUTFIL statement. By contrast, the **SORTOUT data set** is created by default if a DD statement for it is present. The term "SORTOUT data set" denotes the single non-OUTFIL output data set, but in fact, the SORTOUT ddname can be used for an OUTFIL data set either explicitly or by default.

  If SORTOUT is identified as an OUTFIL ddname, either explicitly (for example, via FILES=OUT) or by default (OUTFIL without FILES or FNAMES), the data set associated with the SORTOUT ddname will be processed as an OUTFIL data set rather than as the SORTOUT data set.

  OUTFIL data sets have characteristics and requirements similar to those for the SORTOUT data set, but there are differences in the way each is processed. The major differences are that an E39 exit routine is not entered for OUTFIL data sets, and that OUTFIL processing does not permit the use of the LRECL value to pad fixed-format OUTFIL records. (DFSORT will automatically determine and set an appropriate RECFM, LRECL, and BLKSIZE for each OUTFIL data set for which these attributes are not specified or available.)

  For a single DFSORT application, OUTFIL data sets can be intermixed with respect to VSAM and non-VSAM, tape and disk, and so on. All of the data sets specified for a particular OUTFIL statement are processed in a similar way and thus are referred to as an **OUTFIL group**. (That is, you group OUTFIL data sets that use the same operands by specifying them on a single OUTFIL statement.) For example, the first OUTFIL statement might have an INCLUDE operand that applies to an OUTFIL group of one non-VSAM data set on disk and another on tape; a second OUTFIL statement might have OMIT and OUTREC operands that apply to an OUTFIL group of one non-VSAM data set on disk and two VSAM data sets.

  Records are processed for OUTFIL as they are for SORTOUT, after all other DFSORT processing is complete. Conceptually, you can think of an **OUTFIL input record** as being intercepted at the point between being passed from an E35 exit and written to SORTOUT, although neither an E35 exit nor

SORTOUT need actually be specified with OUTFIL processing. With that in mind, see Figure 2 on page 10 for details on the processing that occurs prior to processing the OUTFIL input record. In particular:

- Records deleted by an E15 or E35 exit, an INCLUDE, OMIT or SUM statement, or the SKIPREC or STOPAFT parameter are not available for OUTFIL processing
- If records are reformatted by an E15 exit, an INREC or OUTREC statement, or an E35 exit, the resulting reformatted record is the OUTFIL input record to which OUTFIL fields must refer.

- **STARTREC** starts processing for an OUTFIL group at a specific OUTFIL input record. **ENDREC** ends processing for an OUTFIL group at a specific OUTFIL input record. **SAMPLE** selects a sample of OUTFIL input records for an OUTFIL group using a specific interval and number of records in that interval. Separately or together, STARTREC, ENDREC, and SAMPLE select a range of records to which subsequent OUTFIL processing will apply.
- **INCLUDE, OMIT**, and **SAVE** select the records to be included in the data sets of an OUTFIL group. INCLUDE and OMIT operate against the specified fields of each OUTFIL input record to select the output records for their OUTFIL group (all records are selected by default). SAVE selects the records that are not selected for any other OUTFIL group.

  Whereas the INCLUDE and OMIT statements apply to all input records, the INCLUDE and OMIT parameters apply only to the OUTFIL input records for their OUTFIL group. The INCLUDE and OMIT parameters have all of the logical expression capabilities of the INCLUDE and OMIT statements.
- **ACCEPT** limits the number of OUTFIL input records accepted for processing for an OUTFIL group. A record is accepted if it is not deleted by STARTREC, ENDREC, SAMPLE, INCLUDE or OMIT processing. ACCEPT limits the number of records to which subsequent OUTFIL processing will apply.
-
- **PARSE, OUTREC, BUILD, OVERLAY, FINDREP, or IFTHEN** reformat the output records for an OUTFIL group. These parameters allow you to rearrange, edit, replace, remove and change fixed position/length fields or variable position/length fields from the OUTFIL input records and to insert blanks, zeros, strings, current date, future date, past date, current time, sequence numbers, decimal constants, and the results of arithmetic expressions.

  OVERLAY allows you to change specific existing columns without affecting the entire record.

  FINDREP allows you to do various find and replace operations.

  IFTHEN clauses allow you to reformat different records in different ways according to the criteria you specify. IFOUTLEN can be used with IFTHEN clauses to set the output LRECL.

  OUTREC or BUILD gives you complete control over the items in your reformatted records and the order in which they appear, and also allows you to produce multiple reformatted output records from each input record, with or without intervening blank output records.

  VTOF or **CONVERT** can be used with OUTREC or BUILD to convert variable-length input records to fixed-length output records.

  VLFILL can be used to allow processing of variable-length input records which are too short to contain all specified OUTREC or BUILD fields.

  Whereas the FIELDS, BUILD, OVERLAY, FINDREP, and IFTHEN parameters of the OUTREC statement apply to all input records, the OUTREC, BUILD, OVERLAY, FINDREP, and IFTHEN parameters of the OUTFIL statement apply only to the OUTFIL input records for its OUTFIL group. In addition, the OUTREC and BUILD parameters of the OUTFIL statement support the forward slash (/) separator for creating blank records and new records, whereas the FIELDS and BUILD parameters of the OUTREC statement do not.

-

- **FTOV** can be used to convert fixed-length input records to variable-length output records. FTOV can be used with or without OUTREC , BUILD, OVERLAY, FINDREP, or IFTHEN.
- **VLTRIM** can be used to remove the trailing bytes with a specified value, such as blanks, binary zeros or asterisks, from variable-length records. VLTRIM can be used with or without FTOV.
- **VLTRAIL** can be used to insert a character string or hexadecimal string at the end of variable-length records. VLTRAIL can be used with or without FTOV or VLTRIM.
- **REPEAT** can be used to repeat each output record a specified number of times.
- **SPLIT, SPLITBY, or SPLIT1R** splits the output records in rotation among the data sets of an OUTFIL group.

  With SPLIT, the first output record is written to the first OUTFIL data set in the group, the second output record is written to the second data set, and so on. When each OUTFIL data set has one record, the rotation starts again with the first OUTFIL data set.

  SPLITBY can be used to rotate by a specified number of records rather than by one record, for example, records 1-10 to the first OUTFIL data set, records 11-20 to the second OUTFIL data set, and so on. When each OUTFIL data set has n records, the rotation starts again with the first OUTFIL data set.

  SPLIT1R can be used to rotate once by a specified number of records. SPLIT1R continues with the last OUTFIL data set instead of rotating back to the first OUTFIL data set. When each OUTFIL data set has n records, the last OUTFIL data set will have the remainder of the records even if that is more than n.
-
- **LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS**, and **NODETAIL** indicate that a report is to be produced for an OUTFIL group, and specify the details of the **report records** to be produced for the report. Reports can contain report records for a report header (first page), report trailer (last page), page header and page trailer (at the top and bottom of each page, respectively), and section headers and trailers (before and after each section, respectively).

  Data records for the report result from the inclusion of OUTFIL input records. All of the capabilities of the OUTREC, BUILD, OVERLAY, FINDREP, or IFTHEN parameters are available to create reformatted data records from the OUTFIL input records. Each set of sequential OUTFIL input records, with the same binary value for a specified field, results in a corresponding set of data records that is treated as a section in the report.

  The length for the data records must be equal to or greater than the maximum report record length. OUTFIL data sets used for reports must have or will be given ANSI control character format ('A' as in, for example, RECFM=FBA or RECFM=VBA), and must allow an extra byte in the LRECL for the carriage control character that DFSORT will add to each report and data record. DFSORT uses these carriage control characters to control page ejects and the placement of the lines in your report according to your specifications. DFSORT uses appropriate carriage controls (for example, C'-' for triple space) in header and trailer records when possible, to reduce the number of report records written. DFSORT always uses the single space carriage control (C' ') in data records. Although these carriage control characters may not be shown when you view an OUTFIL data set (depending upon how you view it), they will be used if you print the report.
- **BLKCCH1, BLKCCT1 or BLKCCH2** can be used avoid forcing a page eject for the report header, report trailer or first page header, respectively, by using a blank instead of a '1' for the ANSI control carriage character of its first line.
- **REMOVECC** can be used to remove the ANSI control characters from a report. In this case, an 'A' is not added to or required for the RECFM and an extra byte is not added to or required for the LRECL.
- **IFTRAIL** can be used to update count and total values in an existing trailer (last) record based on the current data records.
- Figure 7 on page 226 illustrates the order in which OUTFIL records and parameters are processed.

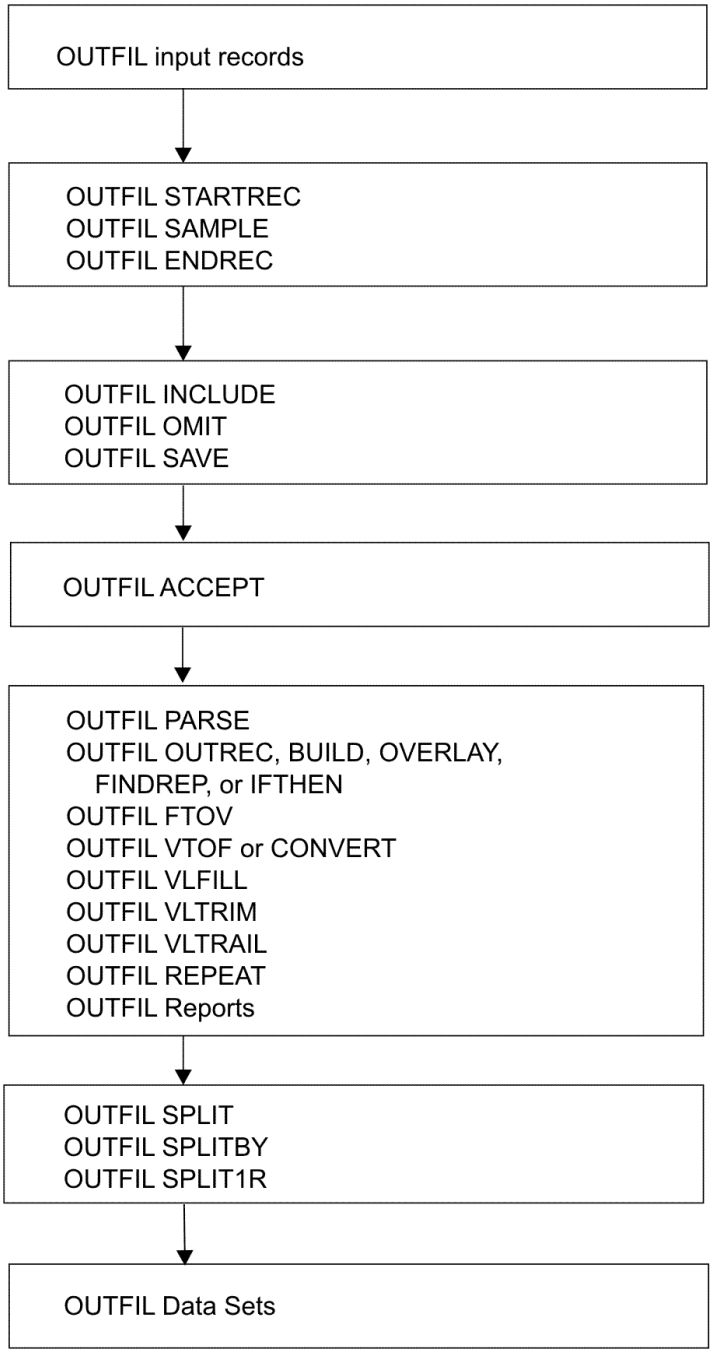
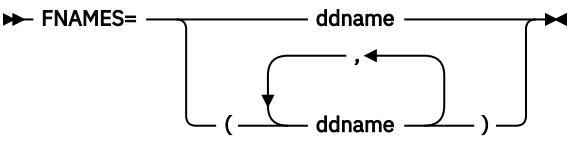


Figure 7. OUTFIL Processing Order

Note:

1. DFSORT accepts but does not process the following OUTFIL operands: BLKSIZE=value, BUFLIM=value, BUFOFF=value, CARDS=value, CLOSE=value, DISK, ESDS, EXIT, FREEOUT, KSDS, LRECL=value, NOTPMK, OPEN=value, OUTPUT, PAGES=value, PRINT, PUNCH, REUSE, RRDS, SPAN, SYSLST, TAPE, and TOL.
2. Sample syntax is shown throughout this section. Complete OUTFIL statement examples are shown and explained under “OUTFIL features—examples” on page 364.

FNAMES



Specifies ddnames associated with the OUTFIL data sets for this OUTFIL statement. The ddnames specified using the FNAMES and FILES parameters constitute the output data sets for this OUTFIL group to which all of the other parameters for this OUTFIL statement apply.

If FNAMES specifies the ddname in effect for the SORTOUT data set (that is, whichever is in effect among ddname from SORTOUT=ddname, ccccOUT from SORTDD=cccc, or SORTOUT) DFSORT will treat the data set associated with that ddname as an OUTFIL data set rather than as the SORTOUT data set.

ddname
specifies a 1- through 8-character ddname. A DD statement must be present for this ddname.

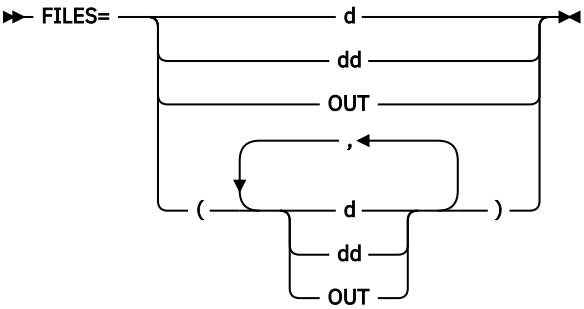
Sample Syntax:

```
OUTFIL FNAMES=(OUT1,OUT2,PRINTER,TAPE)
OUTFIL FNAMES=BACKUP
```

Default for FNAMES: If neither FNAMES nor FILES is specified for an OUTFIL statement, the default ddname is:

- ddname if SORTOUT=ddname is in effect
- ccccOUT if SORTDD=cccc is in effect and SORTOUT=ddname is not in effect
- SORTOUT if neither SORTOUT=ddname nor SORTDD=cccc is in effect.

FILES



Specifies suffixes for ddnames to be associated with the OUTFIL data sets for this OUTFIL statement. The ddnames specified using the FNAMES and FILES parameters constitute the output data sets for this OUTFIL group to which all of the other parameters for this OUTFIL statement apply.

If FILES specifies the ddname in effect for the SORTOUT data set (that is, whichever is in effect among ddname from SORTOUT=name, ccccOUT from SORTDD=cccc, or SORTOUT), DFSORT will treat the data set associated with that ddname as an OUTFIL data set rather than as the SORTOUT data set.

d
specifies the 1-character suffix to be used to form the ddname SORTOFd or ccccOFd if SORTDD=cccc is in effect. A DD statement must be present for this ddname.

dd
specifies the 2-character suffix to be used to form the ddname SORTOFdd or ccccOFdd if SORTDD=cccc is in effect. A DD statement must be present for this ddname.

OUT
specifies the suffix OUT is to be used to form the ddname SORTOUT or ccccOUT if SORTDD=cccc is in effect. A DD statement must be present for this ddname.

Sample Syntax:

```
OUTFIL FILES=(1,2,PR,TP)
OUTFIL FILES=OUT
```

Default for FILES: If neither FNAMES nor FILES is specified for an OUTFIL statement, the default ddname is:

- ddname if SORTOUT=ddname is in effect
- ccccOUT if SORTDD=cccc is in effect and SORTOUT=ddname is not in effect
- SORTOUT if neither SORTOUT=ddname nor SORTDD=cccc is in effect.

STARTREC

Specifies the OUTFIL input record at which OUTFIL processing is to start for this OUTFIL group. OUTFIL input records before this starting record are not included in the data sets for this OUTFIL group.

n
specifies the relative record number. The value for n starts at 1 (the first record) and is limited to 28 digits (15 significant digits).

Sample Syntax:

```
OUTFIL FNAMES=SKIP20,STARTREC=21
```

Default for STARTREC: 1.

ENDREC

Specifies the OUTFIL input record at which OUTFIL processing is to end for this OUTFIL group. OUTFIL input records after this ending record are not included in the data sets for this OUTFIL group.

The ENDREC value must be equal to or greater than the STARTREC value if both are specified on the same OUTFIL statement.

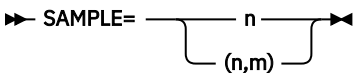
n
specifies the relative record number. The value for n starts at 1 (the first record) and is limited to 28 digits (15 significant digits).

Sample Syntax:

```
OUTFIL FNAMES=TOP10,ENDREC=10
OUTFIL FNAMES=FRONT,ENDREC=500
OUTFIL FNAMES=MIDDLE,STARTREC=501,ENDREC=2205
OUTFIL FNAMES=BACK,STARTREC=2206
```

Default for ENDREC: The last OUTFIL input record.

SAMPLE



Specifies a sample of OUTFIL input records to be processed for this OUTFIL group. The sample consists of the first m records in every nth interval.

n
specifies the interval size. The value for n starts at 2 (sample every other record) and is limited to 28 digits (15 significant digits).

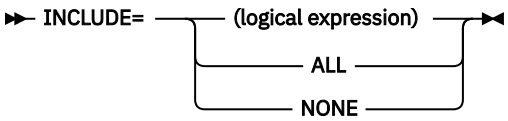
m
specifies the number of records to be processed in each interval. The value for m starts at 1 (process the first record in each interval) and is limited to 28 digits (15 significant digits). If m is not specified, 1 is used for m. If m is specified, it must be less than n.

Sample Syntax:

```
* PROCESS RECORDS 1, 6, 11, ...
  OUTFIL FNAMES=OUT1,SAMPLE=5

* PROCESS RECORDS 1, 2, 1001, 1002, 2001, 2002
  OUTFIL FNAMES=OUT2,SAMPLE=(1000,2),ENDREC=2500

* PROCESS RECORDS 23, 48, 73
  OUTFIL FNAMES=OUT3,STARTREC=23,ENDREC=75,SAMPLE=25

* PROCESS RECORDS 1001, 1002, 1003, 1101, 1102, 1103, ...
  OUTFIL FNAMES=OUT4,STARTREC=1001,SAMPLE=(100,3)
```

Default for SAMPLE: None; must be specified.

INCLUDE



Selects the records to be included in the data sets for this OUTFIL group.

The INCLUDE parameter operates in the same way as the INCLUDE statement, except that:

- The INCLUDE statement applies to all input records; the INCLUDE parameter applies only to the OUTFIL input records for its OUTFIL group.
- FORMAT=f can be specified with the INCLUDE statement, but not with the INCLUDE parameter. Thus, you can use FORMAT=f and p,m or p,m,f fields with the INCLUDE statement, but you must only use p,m,f fields with the INCLUDE parameter. For example:

```
INCLUDE FORMAT=BI,
    COND=(5,4,LT,11,4,OR,21,4,EQ,31,4,OR,
       61,20,SS,EQ,C'FLY')

OUTFIL INCLUDE=(5,4,BI,LT,11,4,BI,OR,21,4,BI,EQ,31,4,BI,OR,
       61,20,SS,EQ,C'FLY')
```

- D2 format can be specified with the INCLUDE statement, but not with the INCLUDE parameter.

See “INCLUDE control statement” on page 91 for complete details.

logical expression
specifies one or more relational conditions logically combined based on fields in the OUTFIL input record. If the logical expression is true for a given record, the record is included in the data sets for this OUTFIL group.

ALL
specifies that all of the OUTFIL input records are to be included in the data sets for this OUTFIL group.

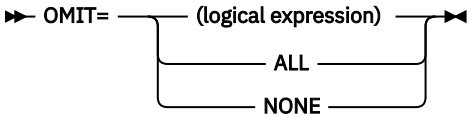
NONE
specifies that none of the OUTFIL input records are to be included in the data sets for this OUTFIL group.

Sample Syntax:

```
OUTFIL FNAMES=J69,INCLUDE=(5,3,CH,EQ,C'J69')
OUTFIL FNAMES=J82,INCLUDE=(5,3,CH,EQ,C'J82')
```

Default for INCLUDE: ALL.

OMIT



Selects the records to be omitted from the data sets for this OUTFIL group.

The OMIT parameter operates in the same way as the OMIT statement, except that:

- The OMIT statement applies to all input records; the OMIT parameter applies only to the OUTFIL input records for its OUTFIL group.
- FORMAT=f can be specified with the OMIT statement, but not with the OMIT parameter. Thus, you can use FORMAT=f and p,m or p,m,f fields with the OMIT statement, but you must only use p,m,f fields with the OMIT parameter. For example:

```
OMIT FORMAT=BI,
  COND=(5,4,LT,11,4,OR,21,4,EQ,31,4,OR,
            61,20,SS,EQ,C'FLY')

OUTFIL OMIT=(5,4,BI,LT,11,4,BI,OR,21,4,BI,EQ,31,4,BI,OR,
       61,20,SS,EQ,C'FLY')
```

- The D2 format can be specified with the OMIT statement, but not with the OMIT parameter.

See “OMIT control statement” on page 170 and “INCLUDE control statement” on page 91 for complete details.

logical expression
specifies one or more relational conditions logically combined based on fields in the OUTFIL input record. If the logical expression is true for a given record, the record is omitted from the data sets for this OUTFIL group.

ALL
specifies that all of the OUTFIL input records are to be omitted from the data sets for this OUTFIL group.

NONE
specifies that none of the OUTFIL input records are to be omitted from the data sets for this OUTFIL group.

Sample Syntax:

```
OUTFIL FILES=01,OMIT=NONE
OUTFIL OMIT=(5,1,BI,EQ,B'110.....')
OUTFIL FNAMES=(OUT1,OUT2),
  OMIT=(7,2,CH,EQ,C'32',OR,18,3,CH,EQ,C'XYZ')
```

Default for OMIT: NONE.

SAVE

Specifies that OUTFIL input records not included by STARTREC, ENDREC, ACCEPT, SAMPLE, INCLUDE or OMIT for any other OUTFIL group are to be included in the data sets for this OUTFIL group. SAVE operates in a global fashion over all of the other OUTFIL statements for which SAVE is not specified,

enabling you to keep any OUTFIL input records that would not be kept otherwise. SAVE will include the same records for each group for which it is specified.

Sample Syntax:

```
OUTFIL INCLUDE=(8,6,CH,EQ,C'ACCTNG'),FNAMES=GP1
OUTFIL INCLUDE=(8,6,CH,EQ,C'DVPMNT'),FNAMES=GP2
OUTFIL SAVE,FNAMES=NOT1OR2
```

Default for SAVE: None; must be specified.

ACCEPT

Specifies the number of OUTFIL input records to be accepted for processing for this OUTFIL group. A record is accepted if it is not deleted by STARTREC, ENDREC, SAMPLE, INCLUDE or OMIT processing for the group. After n OUTFIL input records have been accepted, additional OUTFIL input records are not included in the data sets for this OUTFIL group.

n
specifies the number of records to be accepted. The value for n starts at 1 and is limited to 28 digits (15 significant digits).

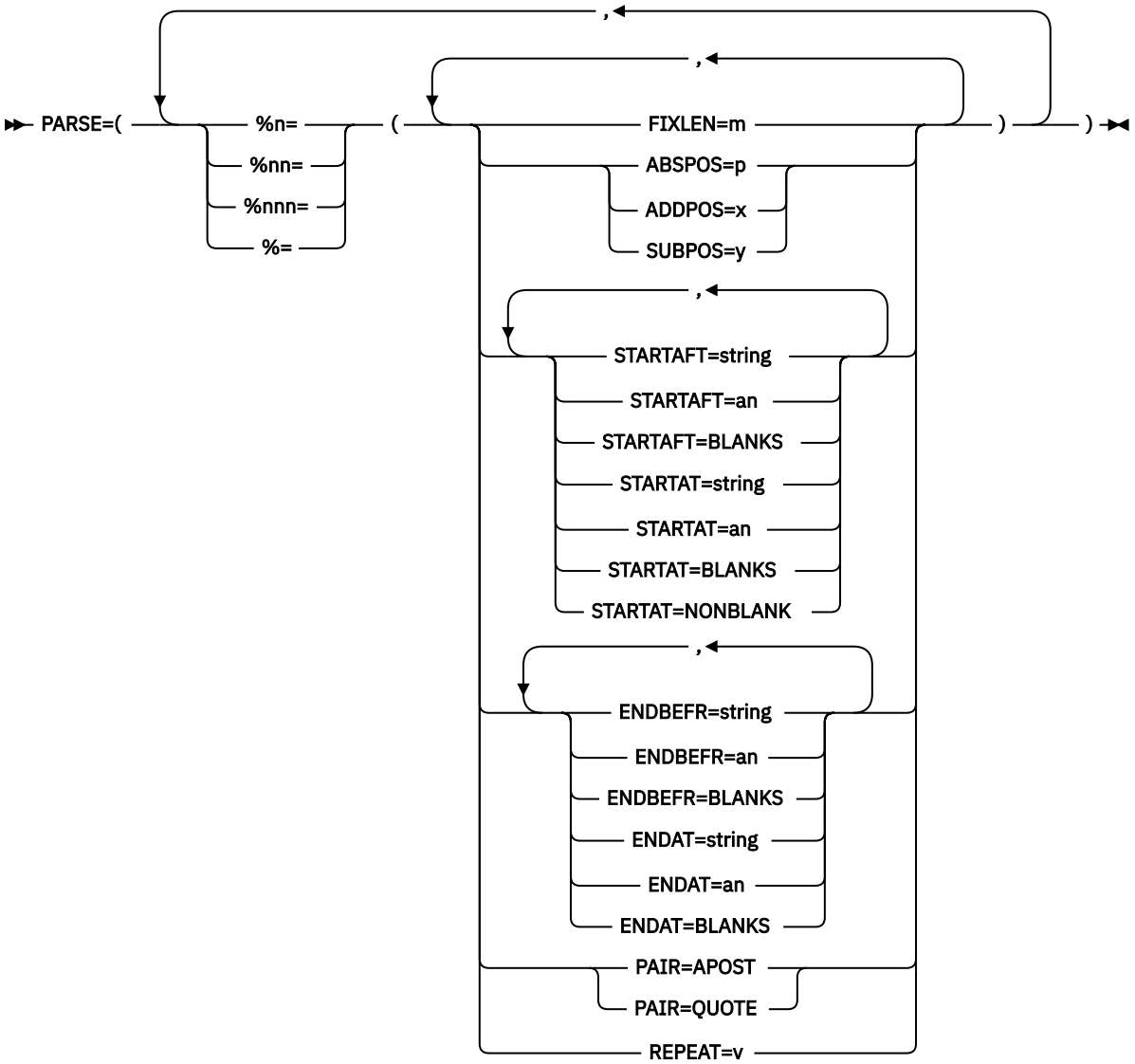
ACCEPT=n operates in a similar way to the ENDREC=n option. However, whereas ENDREC=n stops processing OUTFIL input records for a group at relative record n, ACCEPT=n stops processing OUTFIL input records for a group after n records have been "accepted". If both ACCEPT=n and ENDREC=n are specified, processing will stop when the first criteria is met.

Sample Syntax:

```
OUTFIL FNAMES=OUT1,INCLUDE=(11,3,CH,EQ,C'D51'),ACCEPT=3
OUTFIL FNAMES=OUT2,STARTREC=5,SAMPLE=1000,ACCEPT=12
```

Default for ACCEPT: None; must be specified.

PARSE



This operand allows you to extract variable position/length fields into fixed parsed fields. Parsed fields (%n, %nn or %nnn) can be used where fixed position/length fields (p,m) can be used in the BUILD (or OUTREC) or OVERLAY operands as described later in this section.

Note: Although you can use %n (%0-%9), %nn (%00-%99) or %nnn (%000-%999) for a parsed field, for convenience in this book %nn will be used in general when referring to a parsed field. %n, %0n or %00n can be used interchangeably for parsed field n (for example, %1, %01 or %001 for parsed field 1). %nn or %0nn can be used interchangeably for parsed field nn (for example, %12 or %012 for parsed field 12).

PARSE can be used for many different types of variable fields including delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings, and so on. You can assign up to 1000 parsed fields (%0-%999) to the variable fields you want to extract.

Note that if all of the fields in your records have fixed positions and lengths, you don't need to use PARSE. But if any of the fields in your records have variable positions or lengths, you can use PARSE to treat them as fixed parsed fields in BUILD or OVERLAY. You can mix p,m fields (fixed fields) and %nn fields (parsed fields) in BUILD and OVERLAY.

For each %nn parsed field, you define where the data to be extracted starts and ends, and the length (m) of the fixed field to contain the extracted data. For example, if your input records contained CSV data as follows:

```
AA,BBBB,C,DDDDD
EEEEE,,F,GG
HHH,IIIII,JJ,K
```

and you wanted to reformat the data into fixed positions like this:

```
AA        BBBB      C         DDDDD
EEEEE               F         GG
HHH       IIIII     JJ        K
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%01=(ENDBEFR=C',',FIXLEN=5),
                %02=(ENDBEFR=C',',FIXLEN=5),
                %03=(ENDBEFR=C',',FIXLEN=5),
                %04=(FIXLEN=5)),
         BUILD=(1:%01,11:%02,21:%03,31:%04)
```

The PARSE operand:

- assigns %01 to the first parsed field, and defines it as starting at position 1 and ending before the next (first) comma with a fixed length of 5 bytes
- assigns %02 to the second parsed field, and defines it as starting after the (first) comma and ending before the next (second) comma with a fixed length of 5 bytes
- assigns %03 to the third parsed field, and defines it as starting after the (second) comma and ending before the next (third) comma with a fixed length of 5 bytes
- assigns %04 to the fourth parsed field, and defines it as starting after the (third) comma and ending after 5 bytes with a fixed length of 5 bytes

You can start extracting data:

- at a specific position (ABSPOS=p), a relative position (ADDPOS=x or SUBPOS=y)
- after the start of one or more specific strings (STARTAFT=string), alphanumeric character sets (STARTAFT=an) or blanks (STARTAFT=BLANKS)
- at the start of one or more specific strings (STARTAT=string), alphanumeric character sets (STARTAT=an) or blanks (STARTAT=BLANKS) or a nonblank (STARTAT=NONBLANK).

You can end extracting data:

- before the start of one or more specific strings (ENDBEFR=string), alphanumeric character sets (ENDBEFR=an) or blanks (ENDBEFR=BLANKS),
- at the end of one or more specific strings (ENDAT=string), alphanumeric character sets (ENDAT=an) or blanks (ENDAT=BLANKS)
- after a specified number of bytes (FIXLEN=m without ENDBEFR or ENDAT).

For STARTAFT=an, STARTAT=an, ENDBEFR=an and ENDAT=an, you can specify one of the following for an to indicate the alphanumeric character set you want to use:

- UC: Uppercase characters (A-Z)
- LC: Lowercase characters (a-z)
- MC: Mixed case characters (A-Z, a-z)
- UN: Uppercase and numeric characters (A-Z, 0-9)
- LN: Lowercase and numeric characters (a-z, 0-9)
- MN: Mixed case and numeric characters (A-Z, a-z, 0-9)
- NUM: Numeric characters (0-9)

Multiple STARTAFT/STARTAT strings, UC, LC, MC, UN, LN, MN, NUM, BLANKS or NONBLANK can be used for a %nn parsed field to search for more than one string, set of characters, blanks or a nonblank. The first search satisfied is used.

Multiple ENDBEFR/ENDAT strings, UC, LC, MC, UN, LN, MN, NUM, or BLANKS can be used for a %nn parsed field to search for more than one string, set of characters or blanks. The first search satisfied is used.

For each record, each %nn and % parsed field you define is processed according to the suboperands you define, in the following order:

- ABSPOS or ADDPOS or SUBPOS
- STARTAFT, STARTAT and PAIR
- ENDBEFR, ENDAT and PAIR
- FIXLEN

As each %nn parsed field is processed, data is extracted according to the suboperands you specify.

In addition, as each %nn or % parsed field is processed, the Start Pointer for the next parsed field advances through the record according to the suboperands you specify. By default, for the first parsed field, the Start Pointer is set to position 1 for fixed-length records or to position 5 for variable-length records, and the Start Pointer for each subsequent %nn parsed field uses the Start Pointer as set by the previous %nn parsed field. For any parsed field, the Start Pointer advances as follows depending on the suboperands specified, and may advance more than once (for example, if STARTAFT and ENDBEFR are both specified, or if STARTAT and FIXLEN are both specified):

- if ABSPOS=p is specified, the Start Pointer is set to position p.
- if ADDPOS=x is specified, the Start Pointer is incremented by x.
- if SUBPOS=y is specified, the Start Pointer is decremented by y.
- if STARTAFT=string or STARTAT=string is specified, the search for string starts at the Start Pointer. If string is found, the Start Pointer is set to the byte after the end of the string. If string is not found, the current parsed field and any subsequent parsed fields contain all blanks.
- if STARTAFT=an or STARTAT=an is specified, the search for the character in the specified alphanumeric set starts at the Start Pointer. If a character in the set is found, the Start Pointer is set to the byte after the character. If a character in the set is not found, the current parsed field and any subsequent parsed fields contain all blanks.
- if STARTAFT=BLANKS or STARTAT=BLANKS is specified, the search for blanks starts at the Start Pointer. If a blank is found, the Start Pointer is set to the next nonblank. If a blank is not found, the current parsed field and any subsequent parsed fields contain all blanks.
- if STARTAT=NONBLANK is specified, the search for a nonblank starts at the Start Pointer. If a nonblank is found, the Start Pointer is set to the nonblank. If a nonblank is not found, the current parsed field and any subsequent parsed fields contain all blanks.
- if ENDBEFR=string or ENDAT=string is specified, the search for string starts at the Start Pointer. If string is found, the Start Pointer is set to the byte after the end of the string. If string is not found, data for this parsed field is extracted up to the end of the record. Subsequent parsed fields contain all blanks.
- if ENDBEFR=an or ENDAT=an is specified, the search for the character in the specified alphanumeric set starts at the Start Pointer. If a character in the set is found, the Start Pointer is set to the byte after the character. If a character in the set is not found, data for this parsed field is extracted up to the end of the record. Subsequent parsed fields contain all blanks.
- if ENDBEFR=BLANKS or ENDAT=BLANKS is specified, the search for blanks starts at the Start Pointer. If a blank is found, the Start Pointer is set to the next nonblank. If a blank is not found, data for this parsed field is extracted up to the end of the record. Subsequent parsed fields contain all blanks.
- if ENDBEFR and ENDAT are not specified and FIXLEN=m is specified, the Start Pointer is set m bytes past the start of the extracted data.
- if the Start Pointer advances past the end of the record, PARSE processing stops.

For example, if you had these records as input:

```
11*+23  NEW  MAXCC=00   (Monica) 18
1*-6 OLD  MAXCC=04 (Joey)           -2735
232*+86342 SHR MAXCC=04  (Rachel)  0
```

and you wanted this output:

```
Monica    MAXCC=00  +000023  011     18
Joey      MAXCC=04  -000006  001  -2735
Rachel    MAXCC=04  +086342  232      0
```

you could use this OUTFIL statement:

```
  OUTFIL FNAMES=OUT1,
    PARSE=(%00=(ENDBEFR=C'*',FIXLEN=3),
      %01=(ENDBEFR=BLANKS,FIXLEN=6),
      %02=(STARTAT=C'MAX',FIXLEN=8),
      %03=(STARTAFT=C'(',ENDBEFR=C')',FIXLEN=6),
      %04=(STARTAFT=BLANKS,FIXLEN=5)),
    BUILD=(%03,11:%02,21:%01,SFF,M26,LENGTH=7,
           30:%00,UFF,M11,LENGTH=3,35:%04,JFY=(SHIFT=RIGHT))
```

For this example, the Start Pointer advances as follows for the first record:

- For %00: Set Start Pointer to position 1. Start searching for '*' end string at position 1. Extract '11' (per ENDBEFR=C'*'). Set Start Pointer after '*' (at '+').
- For %01: Start searching for end blank at the Start Pointer (at '+'). Extract '+23' (per ENDBEFR=BLANKS). Set Start Pointer after the blanks (at 'N').
- For %02: Start searching for 'MAX' start string at the Start Pointer (at 'N'). Extract 'MAXCC=00' (per FIXLEN=8). Increment Start Pointer by m (8) bytes from the start of the extracted string (at blank after '00').
- For %03: Start searching for '(' start string at Start Pointer (at blank after '00'). Set Start Pointer after '(' (at 'M'). Start searching for ')' end string at Start Pointer (at 'M'). Extract 'Monica' (per ENDBEFR=C')'). Set Start Pointer after ')' (at blank after ')').
- For %04: Start searching for start blank at the Start Pointer (at blank after ')'). Extract '18 ' (per FIXLEN=5).

If the Start Pointer advances past the end of the record, PARSE processing stops. For example, if you had these input records:

```
First=George Last=Washington
First=John Middle=Quincy Last=Adams
```

and you used this OUTFIL statement:

```
  OUTFIL FNAMES=OUT1,
     PARSE=(%00=(STARTAFT=C'First=',ENDBEFR=C' ',FIXLEN=12),
            %01=(STARTAFT=C'Middle=',ENDBEFR=C' ',FIXLEN=12),
            %02=(STARTAFT=C'Last=',ENDBEFR=C' ',FIXLEN=12)),
     BUILD=(%00,%01,%02)
```

the %01 parsed field (middle name) is not found in the first record, so PARSE processing stops and the %02 parsed field is set to blanks. You can handle this kind of possible missing field situation by using IFTHEN PARSE instead of PARSE. For example:

```
  OUTFIL FNAMES=OUT1,
     IFTHEN=(WHEN=INIT,
       PARSE=(%00=(STARTAFT=C'First=',ENDBEFR=C' ',FIXLEN=12))),
     IFTHEN=(WHEN=INIT,
       PARSE=(%01=(STARTAFT=C'Middle=',ENDBEFR=C' ',FIXLEN=12))),
     IFTHEN=(WHEN=INIT,
       PARSE=(%02=(STARTAFT=C'Last=',ENDBEFR=C' ',FIXLEN=12)),
     BUILD=(%00,%01,%02))
```

By default, the Start Pointer is set to 1 (F) or 5 (V) for each IFTHEN PARSE, so the missing middle name in the first record does not prevent the last name from being extracted.

See the discussion of "IFTHEN" for more information on IFTHEN PARSE.

You must define each %nn field with PARSE before you use it in BUILD or OVERLAY, so generally you would want to specify PARSE before BUILD or OVERLAY. If you define a %nn field with PARSE, but do not use it in BUILD or OVERLAY, data will not be extracted for that parsed field.

Each %nn parsed field (%0-%999) can only be defined once per run with PARSE, but can be used as many times as needed in BUILD or OVERLAY.

The %nn parsed fields defined for a particular source (INREC, OUTREC or OUTFIL) can only be used in the BUILD or OVERLAY operands for that source. Generally, you will only use PARSE for one source (for example, INREC) per run. But you could define a separate set of %nn parsed fields for each source, if needed. For example, you could define %21 and %22 for the INREC statement, %101, %102 and %103 for the OUTREC statement, %40 and %45 for OUTFIL statement 1, and %30, %31 and %32 for OUTFIL statement 2. You could then use %21 and %22 for INREC BUILD or OVERLAY, %102 and %103 for OUTREC BUILD or OVERLAY, %40 and %45 for OUTFIL statement 1 BUILD or OVERLAY, and %30, %31 and %32 for OUTFIL statement 2 BUILD or OVERLAY. Note that you could not, for example, use %21 and %22 for OUTREC BUILD or OVERLAY.

If you specify PARSE=(...),IFTHEN=(...) or IFTHEN=(...),PARSE=(...), DFSORT will terminate. If you need to specify PARSE with IFTHEN, specify it within one or more individual IFTHEN clauses, as appropriate. For details, see the description of "IFTHEN".

%n, %nn or %nnn

Assigns %n, %nn or %nnn to a parsed field. The variable-length data defined for the parsed field is extracted to a fixed length area (padded with blanks or truncated as appropriate) and can be used where a p,m field can be used in BUILD or OVERLAY.

n can be 0 to 9, nn can be 00 to 99, and nnn can be 000 to 999 but each %n, %nn or %nnn value can only be defined once per run. This gives you a maximum of 1000 parsed fields per run.

%0 to %9 will be treated as equivalent to %00 to %09. %010 to %099 will be treated as equivalent to %10 to %99. You can define parse field n with %n, %0n or %00n; you can use %n, %0n and %00n interchangeably. You can define parse field nn with %nn or %0nn; you can use %nn and %0nn interchangeably.

Sample Syntax

```
OUTFIL PARSE=(%99=(ABSPOS=6,STARTAT=C'KW=(',ENDAT=C')',FIXLEN=20),
  %10=(ENDBEFR=UC,ENDBEFR=C'$',FIXLEN=5),
  %03=(STARTAFT=BLANKS,FIXLEN=10)),
  OVERLAY=(1:1,5,%3,C'=',%99,1:1,36,SQZ=(SHIFT=LEFT),%010)
```

%

Specifies a parsed field to be ignored. No data is extracted, but the starting point for the next parsed field advances according to the suboperands specified. Use % when you don't need the data from a particular field, but you do need to get to the next field. For example, if we had the four CSV fields shown earlier as input, but we only wanted to extract the first and fourth fields, we could use this OUTFIL statement:

```
OUTFIL PARSE=(%01=(ENDBEFR=C',',FIXLEN=5),
              %=(ENDBEFR=C','),
              %=(ENDBEFR=C','),
              %04=(FIXLEN=5)),
       BUILD=(1:%01,11:%04,21:%01,HEX)
```

Data is extracted for %01 (first field) and %04 (fourth field), but not for % (second and third fields).

FIXLEN=m

Specifies the length (m) of the fixed area to contain the extracted variable-length data for this %nn fixed parsed field. m can be 1 to 32752. You must specify FIXLEN=m for each %nn parsed field. FIXLEN=m is optional for a % ignored field.

When %nn is specified in BUILD or OVERLAY, the variable-length data is extracted to the fixed area and then used for BUILD or OVERLAY processing. The fixed area always contains m bytes. If the extracted data is longer than m, the fixed area contains the first m bytes of the extracted

data (the data is truncated on the right to m bytes). If the extracted data is shorter than m, the fixed area contains the extracted data padded with blanks on the right to m bytes. If no data is extracted, the fixed area contains all blanks.

The length m from FIXLEN=m for fixed parsed fields (%nn) corresponds to the length (m) for fixed fields (p,m). Keep in mind that %nn fields are left-aligned and may be padded on the right with blanks. You must ensure that the data in each %nn field and its length (m) are valid for the way you want to use that %nn field in BUILD or OVERLAY. For example, if you had these records as input:

```
123,8621
1,302
18345,17
```

and you wanted this output:

```
  1.23   86.21
  0.01    3.02
183.45    0.17
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%00=(ENDBEFR=C',',FIXLEN=5),
                %01=(FIXLEN=5)),
         BUILD=(%00,UFF,EDIT=(IIT.TT),2X,%01,UFF,EDIT=(IIT.TT))
```

Note that the 5-byte extracted parsed fields for %00 are left-aligned and padded with blanks (b) like this:

```
123bb
1bbbb
18345
```

and the 5-byte extracted parsed fields for %01 are left-aligned and padded with blanks (b) like this:

```
8621b
302bb
17bbb
```

Thus, you cannot use numeric formats like ZD or FS for these parsed fields, but you can use UFF. Note also that you must ensure that m from FIXLEN=m for each %nn parsed field is valid for the specific BUILD or OVERLAY item in which you use %nn. For example, you could use 1-44 as m for UFF, but you couldn't use 45 as m.

If ENDBEFR and ENDAT are not specified, the Start Pointer is set m bytes past the start of the extracted data. For example, if you had this record as input:

```
    MAX=(ABCDEF)
```

and you specified:

```
    OUTFIL PARSE=(%100=(STARTAT=C'MAX',FIXLEN=8),
                 %101=(FIXLEN=3),%102=(FIXLEN=1)),
           BUILD=(%100,12:%101,20:%102)
```

The output record would be:

```
MAX=(ABC   DEF     )
```

Sample Syntax

```
  OUTFIL PARSE=(%00=(STARTAFT=C'KW=',FIXLEN=12)),
      BUILD=(%00)
```

ABSPOS=p

Sets the Start Pointer for this parsed field to p. p can be 1 to 32752. By default, the Start Pointer for the first %nn parsed field is position 1 for fixed-length records or position 5 for variable-length records, and the Start Pointer for each subsequent %nn parsed field is the Start Pointer set by the previous %nn field. You can use ABSPOS=p to set the Start Pointer to position p to override the default Start Pointer. If the resulting Start Pointer is less than position 5 for variable length records, it will be set to position 5.

Example

If your input is:

```
****|BB|CCCC|
****|EEEE|FF|
```

and you wanted to reformat the data into fixed positions like this:

```
**** BB   CCCC
**** EEEE FF
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%01=(ABSPOS=6,ENDBEFR=C'|',FIXLEN=4),
       %02=(ENDBEFR=C'|',FIXLEN=4)),
     BUILD=(1,4,X,%01,X,%02)
```

The initial Start Pointer for the %01 parsed field is set to position 6 instead of to position 1.

ADDPOS=x

Increments the Start Pointer for this parsed field by x. x can be 1 to 32752. By default, the Start Pointer for the first %nn parsed field is position 1 for fixed-length records or position 5 for variable-length records, and the Start Pointer for each subsequent %nn parsed field is the Start Pointer set by the previous %nn field. You can use ADDPOS=x to increment the Start Pointer by x to override the default Start Pointer.

Sample Syntax

```
   OUTFIL PARSE=(%00=(ENDAT=C'||',FIXLEN=10),
        %01=(ADDPOS=5,STARTAFT=C';',FIXLEN=6)),
     BUILD=(%00,TRAN=ALTSEQ,%01)
```

SUBPOS=y

Decrements the Start Pointer for this parsed field by y. y can be 1 to 32752. By default, the Start Pointer for the first %nn parsed field is position 1 for fixed-length records or position 5 for variable-length records, and the Start Pointer for each subsequent %nn parsed field is the Start Pointer set by the previous %nn field. You can use SUBPOS=y to decrement the Start Pointer by x to override the default Start Pointer. If the resulting Start Pointer is less than position 1 for fixed-length records, it will be set to position 1. If the resulting Start Pointer is less than position 5 for variable-length records, it will be set to position 5.

Sample Syntax

```
  OUTFIL PARSE=(%00=(ENDBEFR=C'|;|',FIXLEN=10),
       %01=(SUBPOS=1,STARTAT=C'|',ENDAT=C'|',FIXLEN=12)),
     BUILD=(%00,TRAN=ALTSEQ,%01)
```

STARTAFT=string

Data is to be extracted for this parsed field starting after the last byte of the specified string. The search for the string begins at the Start Pointer. If the specified string is not found, data is not extracted for this parsed field or any subsequent parsed fields. If the specified string is found, data is extracted to the fixed area for this parsed field starting at the byte after the end of the string, and the Start Pointer is set to the byte after the end of the string.

string can be 1 to 256 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE control statement" on page 91 for details of coding character and hexadecimal string constants.

Example

If your input is:

```
         MAX=25,832        MIN=2,831  $4
  MAX=1,275    MIN=17 %3
```

and you wanted your output to contain MAX-MIN, you could use this OUTFIL statement:

```
OUTFIL PARSE=(%00=(STARTAFT=C'MAX=',ENDBEFR=X'40',FIXLEN=6),
    %01=(STARTAFT=C'MIN=',ENDBEFR=X'40',FIXLEN=6)),
  OVERLAY=(50:C'DELTA:  ',%00,UFF,SUB,%01,UFF,TO=ZD,LENGTH=8)
```

STARTAFT=an

Data is to be extracted for this parsed field starting after the first character found from the specified alphanumeric set. The search for a character in the specified set begins at the Start Pointer. If a character in the set is not found, data is not extracted for this parsed field or any subsequent parsed field. If a character in the set is found, data is extracted to the fixed area for this parsed field starting at the byte after that character, and the Start Pointer is set to the byte after that character.

an can be one of the following alphanumeric character sets:

- **UC**: Uppercase characters (A-Z)
- **LC**: Lowercase characters (a-z)
- **MC**: Mixed case characters (A-Z, a-z)
- **UN**: Uppercase and numeric characters (A-Z, 0-9)
- **LN**: Lowercase and numeric characters (a-z, 0-9)
- **MN**: Mixed case and numeric characters (A-Z, a-z, 0-9)
- **NUM**: Numeric characters (0-9)

Note that STARTAFT=an is equivalent to specifying a STARTAFT=c operand for each character in the set. For example:

```
%=(STARTAFT=NUM)
```

is equivalent to specifying

```
      %=(STARTAFT=C'0',STARTAFT=C'1',STARTAFT=C'2',STARTAFT=C'3',
         STARTAFT=C'4',STARTAFT=C'5',STARTAFT=C'6',STARTAFT=C'7',
         STARTAFT=C'8',STARTAFT=C'9')
```

You can add more characters to a set with additional STARTAFT operands. For example, if you wanted the lowercase characters and the $ and @ characters for the set, you could use:

```
%05=(STARTAFT=LC,STARTAFT=C'$',STARTAFT=C'@',FIXLEN=10)
```

Example

If your input is:

```
          a123    1Stop
     Z0056    2rest
      q8    3go
```

and you wanted your output to be:

```
000123 STOP
000056 REST
000008 GO
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%00=(STARTAFT=MC,ENDBEFR=C' ',FIXLEN=5),
                %01=(STARTAFT=NUM,FIXLEN=8)),
         BUILD=(%00,UFF,EDIT=(TTTTTT),X,%01,TRAN=LTOU)
```

STARTAFT=BLANKS

Data is to be extracted for this parsed field starting at the first nonblank after one or more blanks. The search for a blank begins at the Start Pointer. If a blank is not found, data is not extracted for this parsed field or any subsequent parsed fields. If a blank is found, data is extracted to the fixed area for this parsed field starting at the first nonblank, and the Start Pointer is set to the first nonblank.

Example

If your input is:

```
         Frank     D28
  Vicky                    D52
```

and you wanted your output to be:

```
Frank        D28
Vicky        D52
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%00=(STARTAFT=BLANKS,FIXLEN=8),
     %01=(STARTAFT=BLANKS,FIXLEN=3)),
    BUILD=(%00,14:%01)
```

STARTAT=string

Data is to be extracted for this parsed field starting at the first byte of the specified string. The search for the string begins at the Start Pointer. If the specified string is not found, data is not extracted for this parsed field or any subsequent parsed fields. If the specified string is found, data is extracted to the fixed area for this parsed field starting at the first byte of the string, and the Start Pointer is set to the byte after the end of the string.

string can be 1 to 256 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See “INCLUDE control statement” on page 91 for details of coding character and hexadecimal string constants.

Example

If your input is:

```
          MAX=58321        MIN=00273
  MAX=01275    MIN=00017
```

and you wanted your output to be:

```
MAX=58321 MIN=00273
MAX=01275 MIN=00017
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%00=(STARTAT=C'MAX=',FIXLEN=9),
     %01=(STARTAT=C'MIN=',FIXLEN=9)),
    BUILD=(%00,X,%01)
```

STARTAT=an

Data is to be extracted for this parsed field starting at the first character found from the specified alphanumeric set. The search for a character in the specified set begins at the Start Pointer. If a character in the set is not found, data is not extracted for this parsed field or any subsequent parsed field. If a character in the set is found, data is extracted to the fixed area for this parsed field starting at that character, and the Start Pointer is set to the byte after that character.

See STARTAFT=an previously in this section for more information on using the available alphanumeric character sets.

Example

If your input is:

```
$@$321%@%$Frank
@053$Susan
%%%$$836%$Vicky
```

and you wanted your output to contain the amount and name fields:

```
321 Frank
053 Susan
836 Vicky
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%00=(STARTAT=NUM,FIXLEN=3),
      %01=(STARTAT=UC,FIXLEN=10)),
    BUILD=(%00,X,%01)
```

STARTAT=BLANKS
Data is to be extracted for this parsed field starting at the first blank. The search for a blank begins at the Start Pointer. If a blank is not found, data is not extracted for this parsed field or any subsequent parsed fields. If a blank is found, data is extracted to the fixed area for this parsed field starting at the first blank, and the Start Pointer is set to the first nonblank.

STARTAT=NONBLANK
Data is to be extracted for this parsed field starting at the first nonblank. The search for a nonblank begins at the Start Pointer. If a nonblank is not found, data is not extracted for this parsed field or any subsequent parsed fields. If a nonblank is found, data is extracted to the fixed area for this parsed field starting at the nonblank, and the Start Pointer is set to the nonblank.

Example

If your input is:

```
Frank     D28
   Victoria           D52
  Holly D15
Roberta     D52
```

and you wanted your output to be:

```
Frank        D28
Victoria     D52
Holly        D15
Roberta      D52
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%00=(STARTAT=NONBLANK,ENDBEFR=BLANKS,FIXLEN=8),
      %01=(FIXLEN=3)),
    BUILD=(%00,14:%01)
```

Note: Multiple STARTAFT/STARTAT strings, UC, LC, MC, UN, LN, MN, NUM, BLANKS or NONBLANK can be used for a %nn parsed field to search for more than one string, set of characters, blanks or a nonblank. The first search satisfied is used.

Example

If your input is:

```
  12   MAXCC=08
58 MINCC=00
         MAXCC=04
```

and you wanted your output to be:

```
MAXCC=08
MINCC=00
MAXCC=04
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%00=(STARTAT=C'MAXCC=',STARTAT=C'MINCC=',
               FIXLEN=8)),BUILD=(%00)
```

ENDBEFR=string

Data is to be extracted for this parsed field ending before the first byte of the specified string. The search for the string begins at the Start Pointer. If the specified string is not found, data is extracted to the fixed area for this parsed field up to the end of the record, but data is not extracted for any subsequent parsed fields. If the specified string is found, data is extracted to the fixed area for this parsed field up to the byte before the start of the string, and the Start Pointer is set to the byte after the end of the string.

string can be 1 to 256 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE control statement" on page 91 for details of coding character and hexadecimal string constants.

Example

If your input is:

```
Morgan Hill;California;5000:
San Jose;California;2000:
Austin;Texas;8000:
```

and you wanted your output to be:

```
Morgan Hill   California  5000
San Jose      California  2000
Austin        Texas       8000
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%00=(ENDBEFR=C';',FIXLEN=14),
      %01=(ENDBEFR=C';',FIXLEN=12),
      %02=(ENDBEFR=C':',FIXLEN=4)),
    BUILD=(%00,%01,%02)
```

ENDBEFR=X'00' can be used to extract null-terminated strings to fixed parsed fields.

Example

If your input contains null-terminated strings that look like this in hex:

```
F0F5F800
F1F2F3F4F500
F9F7F2F000
F500
F0F0F3F700
```

and you wanted to convert the null-terminated strings to right-aligned numeric values like this:

```
   58
12345
 9720
    5
   37
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%01=(ENDBEFR=X'00',FIXLEN=5)),
    BUILD=(%01,UFF,EDIT=(IIIIT))
```

ENDBEFR=an

Data is to be extracted for this parsed field ending before the first character found from the specified alphanumeric set. The search for a character in the specified set begins at the Start Pointer. If a character in the set is not found, data is extracted to the fixed area for this parsed field up to the end of the record, but data is not extracted for any subsequent parsed fields. If a character in the set is found, data is extracted to the fixed area for this parsed field up to the byte before that character, and the Start Pointer is set to the byte after that character.

See STARTAFT=an previously in this section for more information on using the available alphanumeric character sets.

Example

If your input is:

```
$$$$Samantha
$052
$$$$$cat
$$358721
```

and you wanted to extract the leading $ characters in each input record, you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%10=(ENDBEFR=MN,FIXLEN=10)),
     BUILD=(%10)
```

ENDBEFR=BLANKS

Data is to be extracted for this parsed field ending before the first blank. The search for a blank begins at the Start Pointer. If a blank is not found, data is extracted to the fixed area for this parsed field up to the end of the record, but data is not extracted for any subsequent parsed fields. If a blank is found, data is extracted to the fixed area for this parsed field up to the byte before the blank, and the Start Pointer is set to the first nonblank.

Example

If your input is:

```
1     Frank     D28 123   No
2 Loretta            D52   58 Yes
```

and you wanted your output to be:

```
0123 Frank   D28
0058 Loretta D52
```

you could use this OUTFIL statement:

```
   OUTFIL PARSE=(%00=(STARTAFT=BLANKS,ENDBEFR=BLANKS,FIXLEN=8),
      %01=(ENDBEFR=BLANKS,FIXLEN=3),
      %02=(ENDBEFR=C' ',FIXLEN=4)),
    BUILD=(%02,UFF,EDIT=(TTTT),X,%00,%01)
```

ENDAT=string

Data is to be extracted for this parsed field ending at the last byte of the specified string. The search for the string begins at the Start Pointer. If the specified string is not found, data is extracted to the fixed area for this parsed field up to the end of the record, but data is not extracted for any subsequent parsed fields. If the specified string is found, data is extracted to the fixed area for this parsed field up to the last byte of the string, and the Start Pointer is set to the byte after the end of the string

string can be 1 to 256 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See “INCLUDE control statement” on page 91 for details of coding character and hexadecimal string constants.

Example

If your input is:

```
12,(June)
5852,(Gracie)
```

and you wanted your output to be:

```
(June)
(Gracie)
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%00=(STARTAT=C'(',ENDAT=C')',FIXLEN=12)),
    BUILD=(%00)
```

ENDAT=an

Data is to be extracted for this parsed field ending at the first character found from the specified alphanumeric set. The search for a character in the specified set begins at the Start Pointer. If a character in the set is not found, data is extracted to the fixed area for this parsed field up to the end of the record, but data is not extracted for any subsequent parsed fields. If a character in the set is found, data is extracted to the fixed area for this parsed field up to that character, and the Start Pointer is set to the byte after that character.

See STARTAFT=an previously in this section for more information on using the available alphanumeric character sets.

Example

If your input is:

```
        A0005X2
           Z081789Q331
R3M05
    S52J06834
```

and you wanted your output to be:

```
A0005X     2
Z081789Q   331
R3M        05
S52J       06834
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%01=(STARTAT=UC,ENDAT=UC,FIXLEN=10),
                %02=(FIXLEN=5)),
         BUILD=(%01,X,%02)
```

ENDAT=BLANKS

Data is to be extracted for this parsed field ending at the last blank before a nonblank. The search for a blank begins at the Start Pointer. If a blank is not found, data is extracted to the fixed area for this parsed field up to the end of the record, but data is not extracted for any subsequent parsed fields. If a blank is found, data is extracted to the fixed area for this parsed field up to the byte before the first nonblank, and the Start Pointer is set to the first nonblank.

Note: Multiple ENDBEFR/ENDAT strings, UC, LC, MC, UN, LN, MN, NUM, or BLANKS can be used for a %nn parsed field to search for more than one string, set of characters or blanks. The first search satisfied is used.

Example

If your input is:

```
  12;
58:
         04/
```

and you wanted your output to be:

```
12
58
04
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%01=(ENDBEFR=C';',ENDBEFR=C':',ENDBEFR=C'/',
                FIXLEN=10)),
     BUILD=(%01,UFF,TO=ZD,LENGTH=2,80:X)
```

PAIR=APOST

Do not search for strings or blanks between apostrophe (') pairs. Use POST=APOST when you might have strings you are searching for within literals that should not satisfy the search.

Once an apostrophe is found, searching is discontinued until another apostrophe is found. Searching then continues after the second apostrophe of the pair. If an unpaired apostrophe is found, strings or BLANKS will not be recognized from that point on, so be careful not to set the Start Pointer in the middle of an apostrophe pair.

Do not specify PAIR=APOST if you want to search for a string containing an apostrophe, because the string will not be searched for within the paired apostrophes.

Example

If your input is:

```
'23','12,567,823','5,032'
'9,903','18,321','8'
```

and you wanted your output to be:

```
        23 12,567,823      5,032
    9,903     18,321          8
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%00=(ENDBEFR=C',',FIXLEN=12,PAIR=APOST),
                %01=(ENDBEFR=C',',FIXLEN=12,PAIR=APOST),
                %02=(FIXLEN=12)),
         BUILD=(%00,UFF,EDIT=(II,III,IIT),X,
                %01,UFF,EDIT=(II,III,IIT),X,
                %02,UFF,EDIT=(II,III,IIT))
```

With PAIR=APOST, the commas outside the apostrophe pairs satisfy the search, but the commas within the apostrophe pairs do not satisfy the search.

PAIR=QUOTE

Do not search for strings or blanks between quote (") pairs. Use POST=QUOTE when you might have strings you are searching for within literals that should not satisfy the search.

Once a quote is found, searching is discontinued until another quote is found. Searching then continues after the second quote of the pair. If an unpaired quote is found, strings or BLANKS will not be recognized from that point on, so be careful not to set the Start Pointer in the middle of a quote pair.

Do not specify PAIR=QUOTE if you want to search for a string containing a quote, because the string will not be searched for within the paired quotes.

Example

If your input is:

```
"23","12,567,823","5,032"
"9,903","18,321","8"
```

and you wanted your output to be:

```
       23 12,567,823      5,032
    9,903     18,321          8
```

you could use this OUTFIL statement:

```
OUTFIL PARSE=(%00=(ENDBEFR=C',',FIXLEN=12,PAIR=QUOTE),
              %01=(ENDBEFR=C',',FIXLEN=12,PAIR=QUOTE),
              %02=(FIXLEN=12)),
       BUILD=(%00,UFF,EDIT=(II,III,IIT),X,
              %01,UFF,EDIT=(II,III,IIT),X,
              %02,UFF,EDIT=(II,III,IIT))
```

With PAIR=QUOTE, the commas outside the quote pairs satisfy the search, but the commas within the quote pairs do not satisfy the search.

Default for PARSE: None; must be specified.

REPEAT=v

Repeat this parsed field v times.

REPEAT=v can be used with % to specify v identically defined consecutive parsed fields to be ignored. v can be 2 to 1000. For example, to ignore five consecutive comma delimited fields, you can use:

```
%=(ENDBEFR=C',',REPEAT=5),
```

which is equivalent to using:

```
%=(ENDBEFR=C','),
%=(ENDBEFR=C','),
%=(ENDBEFR=C','),
%=(ENDBEFR=C','),
%=(ENDBEFR=C','),
```

REPEAT=v can be used with %nn to specify v identically defined consecutive parsed fields for which data is to be extracted. v can be 2 to 1000. The parsed fields will start with the %nn field you select and be incremented by 1 for each repeated parsed field.

For example, to extract four consecutive 10-byte comma delimited fields as %11, %12, %13 and %14, you can use:

```
%11=(ENDBEFR=C',',FIXLEN=10,REPEAT=4),
```

which is equivalent to using:

```
%11=(ENDBEFR=C',',FIXLEN=10),
%12=(ENDBEFR=C',',FIXLEN=10),
%13=(ENDBEFR=C',',FIXLEN=10),
%14=(ENDBEFR=C',',FIXLEN=10),
```

You must ensure that the %nn fields resulting from the use of REPEAT=v are unique for the run. For example, %21=(ENDBEFR=C',',FIXLEN=8,REPEAT=10) defines %21-%30, so if you used any of %21-%30 separately, DFSORT would issue an error message and terminate; the next available %nn field would be %31.

You must also ensure that any %nnn field resulting from the use of REPEAT=v does not exceed %999. For example, %998=(STARTAFT=C'$',FIXLEN=4,REPEAT=3) defines %998-%1000; since %1000 is invalid, DFSORT would issue an error message and terminate.

Example

If your input is the following period delimited values:

```
11.2.30.4.55.160.7.82.95.37
10.22.3.41.5.16.72.2.5.42
3.18.33.12.7.9.52.12
```

and you wanted to extract the first value, skip the next three values and then extract the last six values to produce output like this:

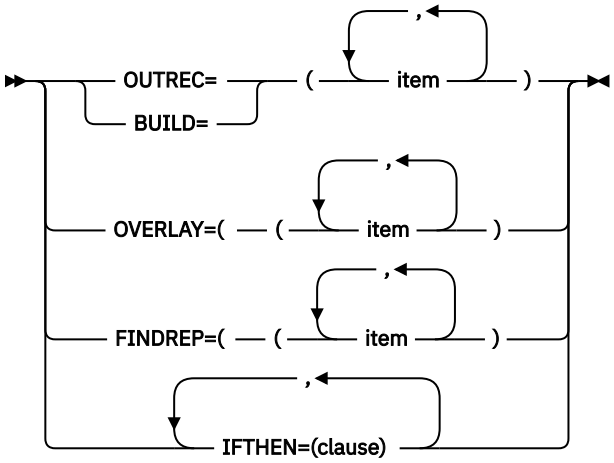
```
011 055 160 007 082 095 037
010 005 016 072 002 005 042
003 007 009 052 012 000 000
```

you could use this OUTFIL statement:

```
  OUTFIL PARSE=(%00=(ENDBEFR=C'.',FIXLEN=3),
             %=(ENDBEFR=C'.',REPEAT=3),
             %01=(ENDBEFR=C'.',FIXLEN=3,REPEAT=6)),
         BUILD=(%00,UFF,EDIT=(TTT),X,
             %01,UFF,EDIT=(TTT),X,
             %02,UFF,EDIT=(TTT),X,
             %03,UFF,EDIT=(TTT),X,
             %04,UFF,EDIT=(TTT),X,
             %05,UFF,EDIT=(TTT),X,
             %06,UFF,EDIT=(TTT))
```

Default for PARSE: None; must be specified.

OUTREC, BUILD, OVERLAY, FINDREP, or IFTHEN



These operands allow you to reformat the OUTFIL input records in this OUTFIL group.

You can create the reformatted OUTFIL records in one of the following ways using unedited, edited, or converted input fields (p,m for fixed fields, or %nn for parsed fields - see PARSE) and a variety of constants:

- **BUILD or OUTREC:** Reformat each record by specifying all of its items one by one. Build gives you complete control over the items you want in your reformatted OUTFIL records and the order in which they appear. You can delete, rearrange and insert fields and constants. Examples:

```
 OUTFIL FNAMES=OUT1,BUILD=(1,20,C'ABC',26:5C'*',
 15,3,PD,EDIT=(TTT.TT),21,30,80:X)

OUTFIL FNAMES=OUT2,
  PARSE=(%00=(ENDBEFR=C',',FIXLEN=6),
         %01=(ENDBEFR=C',',FIXLEN=5),
         %=(ENDBEFR=C','),
         %02=(FIXLEN=6)),
  BUILD=(C'|',%01,SFF,ADD,%02,SFF,M4,LENGTH=12,
         C'|',%00,C'|')
```

- **OVERLAY:** Reformat each record by specifying just the items that overlay specific columns. Overlay lets you change specific existing columns without affecting the entire record. Example:

```
  OUTFIL OVERLAY=(45:45,8,TRAN=LTOU)
```

- **FINDREP**: Reformat each record by doing various types of find and replace operations. Example:

```
OUTREC FINDREP=(INOUT=(C'Pigeon',C'Dove',C'Hawk',C'Eagle'))
```

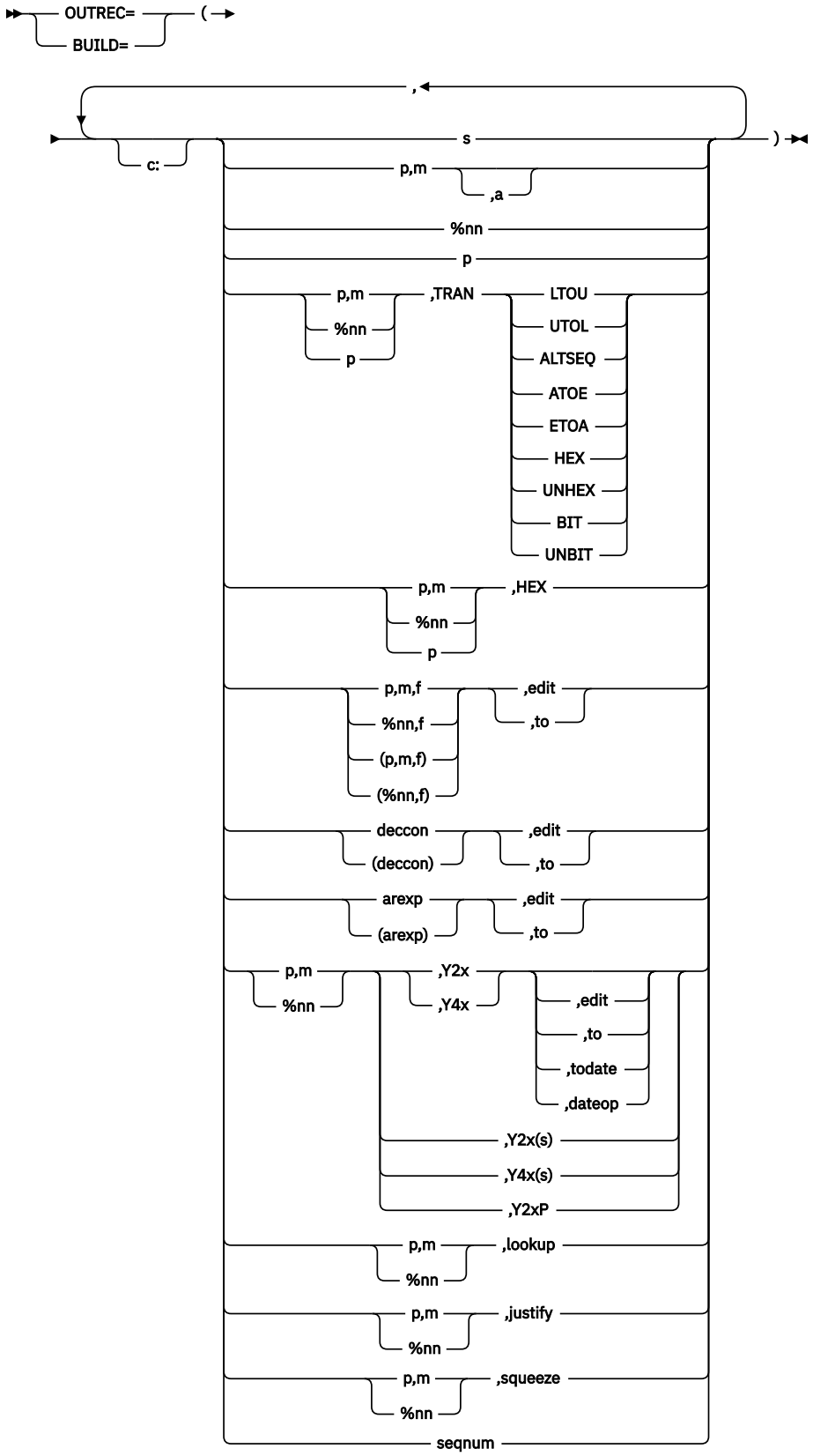
- **IFTHEN clauses:** Reformat different records in different ways by specifying how build, overlay, find/replace, or group operation items are applied to records that meet given criteria. IFTHEN clauses let you use sophisticated conditional logic to choose how different record types are reformatted. Example:

```
OUTFIL IFTHEN=(WHEN=(1,5,CH,EQ,C'TYPE1'),
         BUILD=(1,40,C'**',+1,TO=PD)),
       IFTHEN=(WHEN=(1,5,CH,EQ,C'TYPE2'),
         BUILD=(1,40,+2,TO=PD,X'FFFF')),
       IFTHEN=(WHEN=NONE,OVERLAY=(45:C'NONE'))
```

You can choose to include any or all of the following items in your reformatted OUTFIL records:

- Fixed position/length fields or variable position/length fields. For fixed fields, you specify the starting position and length of the field directly. For variable fields, such as delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings (and many other types), you define rules that allow DFSORT to extract the relevant data into fixed parsed fields, and then use the parsed fields as you would use fixed fields.
- Blanks, binary zeros, character strings, and hexadecimal strings.
- Current date, future date, past date, and current time in various forms.
- Replaced or removed strings.
- Unedited input fields aligned on byte, halfword, fullword, and doubleword boundaries.
- Hexadecimal or bit representations of binary input fields.
- Characters translated from uppercase to lowercase, lowercase to uppercase, ASCII to EBCDIC or EBCDIC to ASCII.
- Left-justified, right-justified, left-squeezed or right-squeezed input fields.
- Numeric input fields of various formats converted to different numeric formats, or to character format edited to contain signs, thousands separators, decimal points, leading zeros or no leading zeros, and so on.
- Decimal constants converted to different numeric formats, or to character format edited to contain signs, thousands separators, decimal points, leading zeros or no leading zeros, and so on.
- The results of arithmetic expressions combining fields, decimal constants, operators (MIN, MAX, MUL, DIV, MOD, ADD and SUB) and parentheses converted to different numeric formats, or to character format edited to contain signs, thousands separators, decimal points, leading zeros or no leading zeros, and so on.
- SMF, TOD, and ETOD date and time fields converted to different numeric formats, or to character format edited to contain separators, leading zeros or no leading zeros, and so on.
- The results of various types of arithmetic operations for input date fields.
- Input date fields of one type (CH, ZD, PD, 2-digit year, 4-digit year, Julian, Gregorian) converted to corresponding output date fields of another type or to a corresponding day of the week.
- Sequence numbers in various formats.
- A character constant, hexadecimal constant or input field selected from a lookup table, based on a character, hexadecimal or bit constant as input.
- A zoned decimal group identifier, a zoned decimal group sequence number, or a field propagated from the first record of a group to all of the records of a group.

OUTREC or BUILD



Specifies all of the items in the reformatted OUTFIL record in the order in which they are to be included. The reformatted OUTFIL record consists of the separation fields, edited and unedited input fields (p,m for fixed fields, or %nn for parsed fields - see PARSE), edited decimal constants, edited

results of arithmetic expressions, and sequence numbers you select, in the order in which you select them, aligned on the boundaries or in the columns you indicate.

For variable-length records, the first item in the BUILD or OUTREC parameter must specify or include the unedited 4-byte record descriptor word (RDW), that is, you must start with 1,m with m equal to or greater than 4. If you want to include the bytes from a specific position to the end of each input record at the end of each reformatted output record, you can specify that starting position (p) as the last item in the BUILD or OUTREC parameter. For example:

```
OUTFIL OUTREC=(1,4,          unedited RDW
   1,2,BI,TO=ZD,LENGTH=5,    display RDW length in decimal
   C'|',                     | separator
   5)                        display input positions 5 to end
```

For fixed-length records, the first input and output data byte starts at position 1. For variable-length records, the first input and output data byte starts at position 5, after the RDW in positions 1-4.

You can use the BUILD or OUTREC parameter to produce multiple reformatted output records for each OUTFIL input record, with or without intervening blank output records.

You can use the BUILD or OUTREC parameter in conjunction with the VTOF or CONVERT parameter to convert variable-length record data sets to fixed-length record data sets.

You can use the BUILD or OUTREC parameter with the FTOV parameter to convert fixed-length record data sets to variable-length record data sets.

You can use the VLFILL parameter to allow processing of variable-length input records which are too short to contain all specified BUILD or OUTREC fields.

You can use the VLTRIM parameter in conjunction with the BUILD or OUTREC parameter to remove specified trailing bytes from the end of variable-length records.

You can use the VLTRAIL parameter in conjunction with the BUILD or OUTREC parameter to insert a string at the end of variable-length records.

The BUILD or OUTREC parameter can be used with any or all of the report parameters (LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS, BLKCCH1, BLKCCH2, BLKCCT1, and NODETAIL) to produce reports. The report parameters specify the report records to be produced, while the BUILD or OUTREC parameter specifies the reformatted data records to be produced. DFSORT uses ANSI carriage control characters to control page ejects and the placement of the lines in your report, according to your specifications. You can use the REMOVECC parameter to remove the ANSI carriage control characters.

When you create an OUTFIL report, the length for the longest or only data record must be equal to or greater than the maximum report record length. You can use the BUILD or OUTREC parameter to force a length for the data records that is longer than any report record; you can then either let DFSORT compute and set the LRECL, or ensure that the computed LRECL is equal to the existing or specified LRECL. Remember to allow an extra byte in the LRECL for the ANSI carriage control character.

For example, if your data records are 40 bytes, but your longest report record is 60 bytes, you can use a BUILD or OUTREC parameter such as:

```
OUTREC=(1,40,80:X)
```

DFSORT will then set the LRECL to 81 (1 byte for the ANSI carriage control character plus 80 bytes for the length of the data records), and pad the data records with blanks on the right.

If you don't want the ANSI carriage control characters to appear in the output data set, use the REMOVECC parameter to remove them. For example, if you specify:

```
OUTREC=(1,40,80:X),REMOVECC
```

DFSORT will set the LRECL to 80 instead of 81 and remove the ANSI carriage control character from each record before it is written.

The BUILD or FIELDS parameter of the OUTREC statement differs from the BUILD or OUTREC parameter of the OUTFIL statement in the following ways:

- The BUILD or FIELDS parameter of the OUTREC statement applies to all input records; the BUILD or OUTREC parameter of the OUTFIL statement only applies to the OUTFIL input records for its OUTFIL group.
- The BUILD or OUTREC parameter of the OUTFIL statement supports the slash (/) separator for creating blank records and new records; the BUILD or FIELDS parameter of the OUTREC statement does not.

The reformatted OUTFIL output record consists of the separation fields, edited and unedited input fields, (p,m for fixed fields, or %nn for parsed fields - see PARSE), edited decimal constants, edited results of arithmetic expressions, and sequence numbers you select, in the order in which you select them, aligned on the boundaries or in the columns you indicate.

c:

specifies the position (column) for a separation field, input field, decimal constant, arithmetic expression, or sequence number, relative to the start of the reformatted OUTFIL output record. Count the RDW (variable-length output records only) but not the carriage control character (for reports) when specifying c:. That is, 1: indicates the first byte of the data in fixed-length output records and 5: indicates the first byte of the data in variable-length output records.

Unused space preceding the specified column is padded with EBCDIC blanks. The following rules apply:

- c must be a number between 1 and 32752.
- c: must be followed by a separation field, input field, decimal constant, or arithmetic expression.
- c must not overlap the previous input field or separation field in the reformatted OUTFIL output record.
- For variable-length records, c: must not be specified before the first input field (the record descriptor word) nor after the variable part of the OUTFIL input record.
- The colon (:) is treated like the comma (,) or semicolon (;) for continuation to another line.

See Table 33 on page 132 for examples of valid and invalid column alignment.

s

specifies that a separation field (blanks, zeros, character string, hexadecimal string, current date, future date, past date, or current time) is to appear in the reformatted OUTFIL output record, or that a new output record is to be started, with or without intervening blank output records. These separation elements (separation fields, new record indicators, and blank record indicators) can be specified before or after any input field. Consecutive separation elements may be specified. For variable-length records, separation elements must not be specified before the first input field (the record descriptor word) or after the variable part of the OUTFIL input record. Permissible values are nX, nZ, nC'xx...x', nX'yy...yy', various date and time constants, /.../ and n/.

nX

Blank separation. n bytes of EBCDIC blanks (X'40') are to appear in the reformatted OUTFIL output records. n can range from 1 to 4095. If n is omitted, 1 is used.

See Table 34 on page 133 for examples of valid and invalid blank separation.

nZ

Binary zero separation. n bytes of binary zeros (X'00') are to appear in the reformatted OUTFIL output records. n can range from 1 to 4095. If n is omitted, 1 is used.

See Table 35 on page 133 for examples of valid and invalid binary zero separation.

nC'xx...x'

Character string separation. n repetitions of the character string constant (C'xx...x') are to appear in the reformatted OUTFIL output records. n can range from 1 to 4095. If n is omitted, 1 is used. x can be any EBCDIC character. You can specify from 1 to 256 characters.

If you want to include a single apostrophe in the character string, you must specify it as two single apostrophes:

```
Required:  O'NEILL Specify:  C'O''NEILL'
```

See Table 36 on page 134 for examples of valid and invalid character string separation.

nX'yy...yy'

Hexadecimal string separation. n repetitions of the hexadecimal string constant (X'yy...yy') are to appear in the reformatted OUTFIL output records. n can range from 1 to 4095. If n is omitted, 1 is used.

The value yy represents any pair of hexadecimal digits. You can specify from 1 to 256 pairs of hexadecimal digits.

See Table 37 on page 135 for examples of valid and invalid hexadecimal string separation.

DATEn, DATEN(c), DATEnP

Constant for current date. The current date of the run is to appear in the reformatted OUTFIL output records.

DATE1, &DATE1, DATE1(c), &DATE1(c), DATE2, &DATE2, DATE2(c), &DATE2(c), DATE3, &DATE3, DATE3(c) or &DATE3(c) can be used to generate a character constant for the current date of the run. DATE1P, &DATE1P, DATE2P, &DATE2P, DATE3P or &DATE3P can be used to generate a packed decimal constant for the current date of the run.

Table 43 on page 252 shows the form of the constant generated for each current date operand and an example of the actual constant generated when the date of the run is June 21, 2005 at 4:42:45 PM, using (/) for (c) where relevant. yyyy represents the year, mm (for date) represents the month (01-12), dd represents the day (01-31), ddd represents the day of the year (001-366), hh represents the hour (00-23), mm (for time) represents the minutes (00-59), ss represents the seconds (00-59), and c can be any character *except* a blank.

Table 43. Current date constants

Format of Operand	Format of Constant	Length (bytes)	Example of Constant
DATE1	C'yyyymmdd'	8	C'20050621'
DATE1(c)	C'yyyycmmcdd'	10	C'2005/06/21'
DATE1P	P'yyyymmdd'	5	P'20050621'
DATE2	C'yyyymm'	6	C'200506'
DATE2(c)	C'yyyycmm'	7	C'2005/06'
DATE2P	P'yyyymm'	4	P'200506'
DATE3	C'yyyyddd'	7	C'2005172'
DATE3(c)	C'yyyycddd'	8	C'2005/172'
DATE3P	P'yyyyddd'	4	P'2005172'
DATE4	C'yyyy-mm-dd-hh.mm.ss'	19	C'2005-06-21-16.52.45'
DATE5	C'yyyy-mm-dd-hh.mm.ss.nnnnnn'	26	C'2005-06-21-16.52.45.602837'

Note: You can precede each of the operands in the table with an & with identical results.

&DATEn, &DATEN(c), &DATEnP
Can be used instead of DATEn, DATEn(c) and DATEnP, respectively

DATEn+r, DATEn(c)+r, DATEnP+r
Constant for future date. A future date relative to the current date of the run is to appear in the reformatted OUTFIL output records.

DATE1+d, &DATE1+d, DATE1(c)+d, &DATE1(c)+d, DATE2+m, &DATE2+m, DATE2(c)+m, &DATE2(c)+m, DATE3+d, &DATE3+d, DATE3(c)+d or &DATE3(c)+d can be used to generate a character constant for a future date relative to the current date of the run. DATE1P+d, &DATE1P+d, DATE2P+m, &DATE2P+m, DATE3P+d or &DATE3P+d can be used to generate a packed decimal constant for a future date relative to the current date of the run. d is days in the future and m is months in the future. d and m can be 0 to 9999.

Table 44 on page 253 shows the form of the constant generated for each future date operand and an example of the actual constant generated when the date of the run is June 21, 2005, using (/) for (c) where relevant. yyyy represents the year, mm (for date) represents the month (01-12), dd represents the day (01-31), ddd represents the day of the year (001-366), and c can be any character *except* a blank.

Table 44. Future Date Constants

Format of Operand	Format of Constant	Length (bytes)	Example of Operand	Example of Constant
DATE1+d	C'yyyymmdd'	8	DATE1+11	C'20050702'
DATE1(c)+d	C'yyyycmmcdd'	10	DATE1(/)+90	C'2005/09/19'
DATE1P+d	P'yyyymmdd'	5	DATE1P+11	P'20050702'
DATE2+m	C'yyyymm'	6	DATE2+2	C'200508'
DATE2(c)+m	C'yyyycmm'	7	DATE2(.)+25	C'2007.07'
DATE2P+m	P'yyyymm'	4	DATE2P+2	P'200508'
DATE3+d	C'yyyyddd'	7	DATE3+200	C'2006007'
DATE3(c)+d	C'yyyycddd'	8	DATE3(-)+1	C'2005-171'
DATE3P+d	P'yyyyddd'	4	DATE3P+200	P'2006007'

Note: You can precede each of the operands in the table with an & with identical results.

&DATEn+r, &DATEn(c)+r, &DATEnP+r
Can be used instead of DATEn+r, DATEn(c)+r and DATEnP+r, respectively.

DATEn-r, DATEn(c)-r, DATEnP-r
Constant for past date. A past date relative to the current date of the run is to appear in the reformatted OUTFIL output records.

DATE1-d, &DATE1-d, DATE1(c)-d, &DATE1(c)-d, DATE2-m, &DATE2-m, DATE2(c)-m, &DATE2(c)-m, DATE3-d, &DATE3-d, DATE3(c)-d or &DATE3(c)-d can be used to generate a character constant for a past date relative to the current date of the run. DATE1P-d, &DATE1P-d, DATE2P-m, &DATE2P-m, DATE3P-d or &DATE3P-d can be used to generate a packed decimal constant for a past date relative to the current date of the run. d is days in the past and m is months in the past. d and m can be 0 to 9999.

Table 45 on page 254 shows the form of the constant generated for each past date operand and an example of the actual constant generated when the date of the run is June 21, 2005, using (/) for (c) where relevant. yyyy represents the year, mm (for date) represents the month (01-12), dd represents the day (01-31), ddd represents the day of the year (001-366), and c can be any character *except* a blank.

Table 45. Past Date Constants

Format of Operand	Format of Constant	Length (bytes)	Example of Operand	Example of Constant
DATE1-d	C'yyyymmdd'	8	DATE1-1	C'20050620'
DATE1(c)-d	C'yyyycmmcdd'	10	DATE1(-)-60	C'2005-04-22'
DATE1P-d	P'yyyymmdd'	5	DATE1P-30	P'20050522'
DATE2-m	C'yyyymm'	6	DATE2-6	C'200412'
DATE2(c)-m	C'yyyycmm'	7	DATE2(/)-1	C'2005/05'
DATE2P-m	P'yyyymm'	4	DATE2P-12	P'200406'
DATE3-d	C'yyyyddd'	7	DATE3-300	C'2004238'
DATE3(c)-d	C'yyyycddd'	8	DATE3(.)-21	C'2005.151'
DATE3P-d	P'yyyyddd'	4	DATE3P-172	P'2004366'

Note: You can precede each of the operands in the table with an & with identical results.

&DATEn-r, &DATEn(c)-r, &DATEnP-r
Can be used instead of DATEn-r, DATEn(c)-r and DATEnP-r, respectively.

TIMEn, TIMEn(c), TIMEnP
Constant for current time. The time of the run is to appear in the reformatted OUTFIL output records. Table 46 on page 254 shows the constant generated for each separation field you can specify along with its length and an example using (:) for (c) where relevant. hh represents the hour (00-23), mm represents the minutes (00-59), ss represents the seconds (00-59), and c can be any character *except* a blank.

Table 46. Current time constants

Separation Field	Constant	Length (bytes)	01:55:43 PM
TIME1	C'hhmmss'	6	C'135543'
TIME1(c)	C'hhcmmcss'	8	C'13:55:43'
TIME1P	P'hhmmss'	4	P'135543'
TIME2	C'hhmm'	4	C'1355'
TIME2(c)	C'hhcmm'	5	C'13:55'
TIME2P	P'hhmm'	3	P'1355'
TIME3	C'hh'	2	C'13'
TIME3P	P'hh'	2	P'13'

&TIMEn, &TIMEn(c), &TIMEnP
Can be used instead of TIMEn, TIMEn(c), and TIMEnP, respectively.

DATE
specifies that the current date is to appear in the reformatted OUTFIL output records in the form 'mm/dd/yy', where mm represents the month (01-12), dd represents the day (01-31), and yy represents the last two digits of the year (for example, 04).

&DATE
can be used instead of DATE.

DATE=(abcd)
specifies that the current date is to appear in the reformatted OUTFIL output records in the form 'adbdc', where a, b, and c indicate the order in which the month, day, and year are to appear and whether the year is to appear as two or four digits, and d is the character to be used to separate the month, day and year.

For a, b, and c, use M to represent the month (01-12), D to represent the day (01-31), Y to represent the last two digits of the year (for example, 04), or 4 to represent the four digits of the year (for example, 2004). M, D, and Y or 4 can each be specified only once. Examples: DATE=(DMY.) would produce a date of the form 'dd.mm.yy', which on March 29, 2004, would appear as '29.03.04'. DATE=(4MD-) would produce a date of the form 'yyyy-mm-dd', which on March 29, 2004, would appear as '2004-03-29'.

a, b, c, and d must be specified.

&DATE=(abcd)
can be used instead of DATE=(abcd).

DATENS=(abc)
specifies that the current date is to appear in the reformatted OUTFIL output record in the form 'abc', where a, b and c indicate the order in which the month, day, and year are to appear and whether the year is to appear as two or four digits.

For a, b and c, use M to represent the month (01-12), D to represent the day (01-31), Y to represent the last two digits of the year (for example, 04), or 4 to represent the four digits of the year (for example, 2004). M, D, and Y or 4 can each be specified only once. Examples: DATENS=(DMY) would produce a date of the form 'ddmmyy', which on March 29, 2004, would appear as '290304'. DATENS=(4MD) would produce a date of the form 'yyyymmdd', which on March 29, 2004, would appear as '20040329'.

a, b and c must be specified.

&DATENS=(abc)
can be used instead of DATENS=(abc).

YDDD=(abc)
specifies that the current date is to appear in the reformatted OUTFIL output records in the form 'acb', where a and b indicate the order in which the year and day of the year are to appear and whether the year is to appear as two or four digits, and c is the character to be used to separate the year and day of the year.

For a and b, use D to represent the day of the year (001-366), Y to represent the last two digits of the year (for example, 04), or 4 to represent the four digits of the year (for example, 2004). D, and Y or 4 can each be specified only once. Examples: YDDD=(DY-) would produce a date of the form 'ddd-yy', which on April 7, 2004, would appear as '098-04'. YDDD=(4D/) would produce a date of the form 'yyyy/ddd', which on April 7, 2004, would appear as '2004/098'.

a, b and c must be specified.

&YDDD=(abc)
can be used instead of YDDD=(abc).

YDDDNS=(ab)
specifies that the current date is to appear in the reformatted OUTFIL output records in the form 'ab', where a and b indicate the order in which the year and day of the year are to appear and whether the year is to appear as two or four digits.

For a and b, use D to represent the day of the year (001-366), Y to represent the last two digits of the year (for example, 04), or 4 to represent the four digits of the year (for example, 2004). D, and Y or 4 can each be specified only once. Examples: YDDDNS=(DY) would produce a date of the form 'dddyy', which on April 7, 2004, would appear as '09804'. YDDDNS=(4D) would produce a date of the form 'yyyyddd', which on April 7, 2004, would appear as '2004098'.

a and b must be specified.

&YDDDNS=(ab)
can be used instead of YDDDNS=(ab).

TIME
specifies that the current time is to appear in the reformatted OUTFIL output records in the form 'hh:mm:ss', where hh represents the hour (00-23), mm represents the minutes (00-59), and ss represents the seconds (00-59).

&TIME
can be used instead of TIME.

TIME=(abc)
specifies that the current time is to appear in the reformatted OUTFIL output records in the form 'hhcmmcss' (24-hour time) or 'hhcmmcss xx' (12-hour time).

If ab is 24, the time is to appear in the form 'hhcmmcss' (24-hour time) where hh represents the hour (00-23), mm represents the minutes (00-59), ss represents the seconds (00-59), and c is the character used to separate the hours, minutes, and seconds. Example: TIME=(24.) would produce a time of the form 'hh.mm.ss', which at 08:25:13 pm would appear as '20.25.13'.

If ab is 12, the time is to appear in the form 'hhcmmcss xx' (12-hour time) where hh represents the hour (01-12), mm represents the minutes (00-59), ss represents the seconds (00-59), xx is am or pm, and c is the character used to separate the hours, minutes, and seconds. Example: TIME=(12.) would produce a time of the form 'hh.mm.ss xx', which at 08:25:13 pm would appear as '08.25.13 pm'.

ab and c must be specified.

&TIME=(abc)
can be used instead of TIME=(abc).

TIMENS=(ab)
specifies that the current time is to appear in the reformatted OUTFIL output record in the form 'hhmmss' (24-hour time) or 'hhmmss xx' (12-hour time).

If ab is 24, the time is to appear in the form 'hhmmss' (24-hour time) where hh represents the hour (00-23), mm represents the minutes (00-59), and ss represents the seconds (00-59). Example: TIMENS=(24) would produce a time of the form 'hhmmss', which at 08:25:13 pm would appear as '202513'.

If ab is 12, the time is to appear in the form 'hhmmss xx' (12-hour time) where hh represents the hour (01-12), mm represents the minutes (00-59), and ss represents the seconds (00-59). Example: TIMENS=(12) would produce a time of the form 'hhmmss xx', which at 08:25:13 pm would appear as '082513 pm'.

ab must be specified.

&TIMENS=(ab)
can be used instead of TIMENS=(ab).

/.../ or n/
Blank records or a new record. A new output record is to be started with or without intervening blank output records. If /.../ or n/ is specified at the beginning or end of OUTREC, n blank output records are to be produced. If /.../ or n/ is specified in the middle of OUTREC, n-1 blank output records are to be produced (thus, / or 1/ indicates a new output record with no intervening blank output records).

At least one input field or separation field must be specified if you use /.../ or n/. For example, OUTREC=(//) is not allowed, whereas OUTREC=(//X) is allowed.

Either n/ (for example, 5/) or multiple /'s (for example, /////) can be used. n can range from 1 to 255. If n is omitted, 1 is used.

As an example, if you specify:

```
OUTFIL OUTREC=(2/,C'Field 2 contains ',4,3,/,
                  C'Field 1 contains ',1,3)
```

an input record containing:

```
111222
```

would produce the following four output records:

```
Blanks
Blanks
Field 2 contains 222
Field 1 contains 111
```

Note that **four** OUTFIL output records are produced for **each** OUTFIL input record.

p,m,a
specifies that an unedited input field is to appear in the reformatted OUTFIL output record.

p
specifies the first byte of the input field relative to the beginning of the OUTFIL input record. The first data byte of a fixed-length record has relative position 1. The first data byte of a variable-length record has relative position 5, because the first four bytes are occupied by the RDW. All fields must start on a byte boundary, and no field can extend beyond byte 32752. See "OUTFIL statements notes" on page 360 for special rules concerning variable-length records.

m
specifies the length in bytes of the input field.

a
specifies the alignment (displacement) of the input field in the reformatted OUTFIL output record relative to the start of the reformatted OUTFIL output record.

The permissible values of **a** are:

H
Halfword aligned. The displacement (p-1) of the field from the beginning of the reformatted OUTFIL input record, in bytes, is a multiple of 2 (that is, position 1, 3, 5, and so forth).

F
Fullword aligned. The displacement is a multiple of 4 (that is, position 1, 5, 9, and so forth).

D
Doubleword aligned. The displacement is a multiple of 8 (that is, position 1, 9, 17, and so forth).

Alignment can be necessary if, for example, the data is used in a COBOL application program where items are aligned through the SYNCHRONIZED clause. Unused space preceding aligned fields are always padded with binary zeros.

%nn
specifies that an unedited parsed input field is to appear in the reformatted OUTFIL output record. See PARSE for details of parsed fields. See p,m,a for further details. Note that alignment (H, F, D) is not permitted for %nn fields (for example, %nn,F results in an error message and termination).

p
specifies the unedited variable part of the OUTFIL input record (that part beyond the minimum record length) is to appear in the reformatted OUTFIL output record as the last field. p without m can only be used for variable-length records; not for fixed-length records.

Attention: If 1,4,p is specified (only RDW and variable part of record), "null" records containing only an RDW will result if the record length is less than p.

A value must be specified for p that is less than or equal to the minimum OUTFIL input record length plus 1 byte.

p,m,TRAN=keyword
Specifies that an input field is to be translated as indicated by the keyword.

Attention: If TRAN=keyword is used for non-character data (for example, PD), it may have unintended consequences.

p
See p under p,m,a

m
specifies the length in bytes of the numeric field and depends on the keyword used as indicated.

TRAN=keyword
specifies the type of translation to be performed. The keyword can be one of the following:

TRAN=LTOU
translates lowercase EBCDIC letters (that is, a-z) to uppercase EBCDIC letters (that is, A-Z). Other characters are not changed. For example, the characters 'Vicky-123,x' would be replaced by 'VICKY-123,X'.

m can be 1 to 32752. The output length will be m bytes.

TRAN=UTOL
translates uppercase EBCDIC letters (that is, A-Z) to lowercase EBCDIC letters (that is, a-z). Other characters are not changed. For example, the characters 'CARRIE-005, CA' would be replaced by 'carrie-005, ca'.

m can be 1 to 32752. The output length will be m bytes.

TRAN=ALTSEQ
translates characters according to the ALTSEQ translation table in effect. For example, the characters X'5B5C' would be replaced by X'4040' if ALTSEQ CODE=(5B40,5C40) was in effect.

m can be 1 to 32752. The output length will be m bytes.

TRAN=ATOE
translates characters from ASCII to EBCDIC using the default standard TCP/IP service ASCII-to-EBCDIC translation table. For example, with TRAN=ATOE, the ASCII characters aB2 (X'614232') would be replaced by the EBCDIC characters aB2 (X'81C2F2').

m can be 1 to 32752. The output length will be m bytes.

Table 47 on page 258 is the default standard TCP/IP table used for TRAN=ATOE. Each column shows the ASCII (A) character value on the left and its equivalent EBCDIC (E) character value on the right.

Table 47. TRAN=ATOE ASCII-to-EBCDIC translation

Part 1 - TRAN=ATOE ASCII-to-EBCDIC Translation

A	E	A	E	A	E	A	E	A	E	A	E	A	E	A	E
00	00	10	10	20	40	30	F0	40	7C	50	D7	60	79	70	97
01	01	11	11	21	5A	31	F1	41	C1	51	D8	61	81	71	98
02	02	12	12	22	7F	32	F2	42	C2	52	D9	62	82	72	99
03	03	13	13	23	7B	33	F3	43	C3	53	E2	63	83	73	A2
04	37	14	3C	24	5B	34	F4	44	C4	54	E3	64	84	74	A3
05	2D	15	3D	25	6C	35	F5	45	C5	55	E4	65	85	75	A4
06	2E	16	32	26	50	36	F6	46	C6	56	E5	66	86	76	A5
07	2F	17	26	27	7D	37	F7	47	C7	57	E6	67	87	77	A6
08	16	18	18	28	4D	38	F8	48	C8	58	E7	68	88	78	A7
09	05	19	19	29	5D	39	F9	49	C9	59	E8	69	89	79	A8
0A	25	1A	3F	2A	5C	3A	7A	4A	D1	5A	E9	6A	91	7A	A9
0B	0B	1B	27	2B	4E	3B	5E	4B	D2	5B	AD	6B	92	7B	C0
0C	0C	1C	22	2C	6B	3C	4C	4C	D3	5C	E0	6C	93	7C	4F
0D	0D	1D	1D	2D	60	3D	7E	4D	D4	5D	BD	6D	94	7D	D0
0E	0E	1E	35	2E	4B	3E	6E	4E	D5	5E	5F	6E	95	7E	A1
0F	0F	1F	1F	2F	61	3F	6F	4F	D6	5F	6D	6F	96	7F	07

Part 2 - TRAN=ATOE ASCII-to-EBCDIC Translation

Table 47. TRAN=ATOE ASCII-to-EBCDIC translation (continued)

Part 1 - TRAN=ATOE ASCII-to-EBCDIC Translation

A	E	A	E	A	E	A	E	A	E	A	E	A	E	A	E
80	00	90	10	A0	40	B0	F0	C0	7C	D0	D7	E0	79	F0	97
81	01	91	11	A1	5A	B1	F1	C1	C1	D1	D8	E1	81	F1	98
82	02	92	12	A2	7F	B2	F2	C2	C2	D2	D9	E2	82	F2	99
83	03	93	13	A3	7B	B3	F3	C3	C3	D3	E2	E3	83	F3	A2
84	37	94	3C	A4	5B	B4	F4	C4	C4	D4	E3	E4	84	F4	A3
85	2D	95	3D	A5	6C	B5	F5	C5	C5	D5	E4	E5	85	F5	A4
86	2E	96	32	A6	50	B6	F6	C6	C6	D6	E5	E6	86	F6	A5
87	2F	97	26	A7	7D	B7	F7	C7	C7	D7	E6	E7	87	F7	A6
88	16	98	18	A8	4D	B8	F8	C8	C8	D8	E7	E8	88	F8	A7
89	05	99	19	A9	5D	B9	F9	C9	C9	D9	E8	E9	89	F9	A8
8A	25	9A	3F	AA	5C	BA	7A	CA	D1	DA	E9	EA	91	FA	A9
8B	0B	9B	27	AB	4E	BB	5E	CB	D2	DB	AD	EB	92	FB	C0
8C	0C	9C	22	AC	6B	BC	4C	CC	D3	DC	E0	EC	93	FC	4F
8D	0D	9D	1D	AD	60	BD	7E	CD	D4	DD	BD	ED	94	FD	D0
8E	0E	9E	35	AE	4B	BE	6E	CE	D5	DE	5F	EE	95	FE	A1
8F	0F	9F	1F	AF	61	BF	6F	CF	D6	DF	6D	EF	96	FF	07

TRAN=ETOA

translates characters from EBCDIC to ASCII using the default standard TCP/IP service EBCDIC-to-ASCII translation table. For example, with TRAN=ETOA, the EBCDIC characters aB2 (X'81C2F2') would be replaced by the ASCII characters aB2 (X'614232').

m can be 1 to 32752. The output length will be m bytes.

Table 48 on page 259is the default standard TCP/IP table used for TRAN=ETOA. Each column shows the EBCDIC (E) character value on the left and its equivalent ASCII (A) character value on the right.

Table 48. TRAN=ETOA ASCII-to-EBCDIC translation

Part 1 - TRAN=ETOA ASCII-to-EBCDIC Translation

E	A	E	A	E	A	E	A	E	A	E	A	E	A	E	A
00	00	10	10	20	1A	30	1A	40	20	50	26	60	2D	70	D7
01	01	11	11	21	1A	31	1A	41	A6	51	A9	61	2F	71	88
02	02	12	12	22	1C	32	16	42	E1	52	AA	62	DF	72	94
03	03	13	13	23	1A	33	1A	43	80	53	9C	63	DC	73	B0
04	1A	14	1A	24	1A	34	1A	44	EB	54	DB	64	9A	74	B1
05	09	15	0A	25	0A	35	1E	45	90	55	A5	65	DD	75	B2
06	1A	16	08	26	17	36	1A	46	9F	56	99	66	DE	76	FC
07	7F	17	1A	27	1B	37	04	47	E2	57	E3	67	98	77	D6
08	1A	18	18	28	1A	38	1A	48	AB	58	A8	68	9D	78	FB
09	1A	19	19	29	1A	39	1A	49	8B	59	9E	69	AC	79	60
0A	1A	1A	1A	2A	1A	3A	1A	4A	9B	5A	21	6A	BA	7A	3A
0B	0B	1B	1A	2B	1A	3B	1A	4B	2E	5B	24	6B	2C	7B	23
0C	0C	1C	1C	2C	1A	3C	14	4C	3C	5C	2A	6C	25	7C	40
0D	0D	1D	1D	2D	05	3D	15	4D	28	5D	29	6D	5F	7D	27
0E	0E	1E	1E	2E	06	3E	1A	4E	2B	5E	3B	6E	3E	7E	3D
0F	0F	1F	1F	2F	07	3F	1A	4F	7C	5F	5E	6F	3F	7F	22

Part 2 - TRAN=ETOA ASCII-to-EBCDIC Translation

Table 48. TRAN=ETOA ASCII-to-EBCDIC translation (continued)

Part 1 - TRAN=ETOA ASCII-to-EBCDIC Translation

```
| E  A| E  A| E  A| E  A| E  A| E  A| E  A| E  A|
|-- --|-- --|-- --|-- --|-- --|-- --|-- --|-- --|
|80 F8|90 8C|A0 C8|B0 B5|C0 7B|D0 7D|E0 5C|F0 30|
|81 61|91 6A|A1 7E|B1 B6|C1 41|D1 4A|E1 E7|F1 31|
|82 62|92 6B|A2 73|B2 FD|C2 42|D2 4B|E2 53|F2 32|
|83 63|93 6C|A3 74|B3 B7|C3 43|D3 4C|E3 54|F3 33|
|84 64|94 6D|A4 75|B4 B8|C4 44|D4 4D|E4 55|F4 34|
|85 65|95 6E|A5 76|B5 B9|C5 45|D5 4E|E5 56|F5 35|
|86 66|96 6F|A6 77|B6 E6|C6 46|D6 4F|E6 57|F6 36|
|87 67|97 70|A7 78|B7 BB|C7 47|D7 50|E7 58|F7 37|
|88 68|98 71|A8 79|B8 BC|C8 48|D8 51|E8 59|F8 38|
|89 69|99 72|A9 7A|B9 BD|C9 49|D9 52|E9 5A|F9 39|
|8A 96|9A 97|AA EF|BA 8D|CA CB|DA A1|EA A0|FA B3|
|8B A4|9B 87|AB C0|BB D9|CB CA|DB AD|EB 85|FB F7|
|8C F3|9C CE|AC DA|BC BF|CC BE|DC F5|EC 8E|FC F0|
|8D AF|9D 93|AD 5B|BD 5D|CD E8|DD F4|ED E9|FD FA|
|8E AE|9E F1|AE F2|BE D8|CE EC|DE A3|EE E4|FE A7|
|8F C5|9F FE|AF F9|BF C4|CF ED|DF 8F|EF D1|FF FF|
```

TRAN=HEX
translates binary values to their equivalent EBCDIC hexadecimal values. For example, with TRAN=HEX, X'C1F1' (C'A1') would be replaced by C'C1F1'.

m can be 1 to 16376. The output length will be m*2 bytes

TRAN=UNHEX
translates EBCDIC hexadecimal values to their equivalent binary values. For example, with TRAN=UNHEX, C'C1F1' would be replaced by X'C1F1' (C'A1').

m can be 1 to 32752. The output length will be (m+1)/2 bytes.

If m is odd, the last nibble will be 0. As an example, C'C1F1F' would translate to X'C1F1F0' (C'A10').

If an input character is not 0-9 or A-F, the equivalent nibble will be set to 0. As an example, C'G2' will be translated to X'02'.

TRAN=BIT
translates binary values to their equivalent EBCDIC bit values. For example, with TRAN=BIT, X'C1F1' (C'A1') would be replaced by C'1100000111110001'.

m can be 1 to 4094. The output length will be m*8 bytes

TRAN=UNBIT
translates EBCDIC bit values to their equivalent binary values. For example, with TRAN=UNBIT, C'1100000111110001' would be replaced by X'C1F1' (C'A1').

m can be 1 to 32752. The output length will be (m+7)/8 bytes

If m is not a multiple of 8, the missing bits on the right will be set to 0. As an example, C'1100000111111' would translate to X'C1F8'.

If an input character is not 0 or 1, the equivalent bit will be set to 0. As an example, C'11100005' will be translated to X'E0'.

Sample Syntax:

```
OUTFIL FNAMES=OUT1,
  BUILD=(5,6,TRAN=UNHEX,2X,20,15,TRAN=UTOL)
```

%nn,TRAN=keyword
Specifies that a parsed input field is to be translated as indicated by the keyword. See PARSE for details of PARSED fields. See p,m,TRAN=keyword for details of the translation functions.

p,TRAN=keyword
Specifies that the variable part of the input record is to be translated as indicated by the keyword. p,TRAN=keyword must be the last item and can only be used for variable-length records; not for fixed-length records. A value must be specified for p that is less than or equal to the minimum input record length plus 1 byte. See p,m,TRAN=keyword for details of the translation functions.

Attention: If 1,4,p,TRAN=keyword is specified (only RDW and variable part of record), "null" records containing only an RDW will result if the record length is less than p.

Sample Syntax:

```
OUTFIL FNAMES=OUT2,BUILD=(1,4,5,TRAN=ETOA)
```

p,m,HEX
Can be used instead of p,m,TRAN=HEX. See p,m,TRAN=HEX for details.

%nn,HEX
Can be used instead of %nn,TRAN=HEX. See %nn,TRAN=HEX for details.

p,HEX
Can be used instead of p,TRAN=HEX. See p,TRAN=HEX for details.

p,m,f,edit or (p,m,f),edit
specifies that an edited numeric input field is to appear in the reformatted OUTFIL output record. You can edit BI, FI, PD, PD0, ZD, FL,CSF, FS, UFF, SFF, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3 or TM4 fields using either pre-defined edit masks (M0-M26) or specific edit patterns you define. You can control the way the edited fields look with respect to length, leading or suppressed zeros, thousands separators, decimal points, leading and trailing positive and negative signs, and so on.

p
See p under p,m,a.

m
specifies the length in bytes of the numeric field. The length must include the sign, if the data is signed. See Table 49 on page 261 for permissible length values.

f
specifies the format of the numeric field:

Table 49. Edit Field Formats and Lengths

Format Code	Length	Description
BI	1 to 8 bytes	Unsigned binary
FI	1 to 8 bytes	Signed fixed-point
PD	1 to 16 bytes	Signed packed decimal
PD0	2 to 8 bytes	Packed decimal with sign and first digit ignored
ZD	1 to 31 bytes	Signed zoned decimal
FL	4 or 8 bytes	Signed hexadecimal floating-point converted to signed integer
CSF or FS	1 to 32 bytes (31 digit limit)	Signed numeric with optional leading floating sign
UFF	1 to 44 bytes (31 digit limit)	Unsigned free form numeric
SFF	1 to 44 bytes (31 digit limit)	Signed free form numeric
DT1	4 bytes	SMF date interpreted as Z'yyyymmdd'

Table 49. Edit Field Formats and Lengths (continued)

Format Code	Length	Description
DT2	4 bytes	SMF date interpreted as Z'yyyymm'
DT3	4 bytes	SMF date interpreted as Z'yyyyddd'
DC1	8 bytes	TOD date interpreted as Z'yyyymmdd'
DC2	8 bytes	TOD date interpreted as Z'yyyymm'
DC3	8 bytes	TOD date interpreted as Z'yyyyddd'
DE1	8 bytes	ETOD date interpreted as Z'yyyymmdd'
DE2	8 bytes	ETOD date interpreted as Z'yyyymm'
DE3	8 bytes	ETOD date interpreted as Z'yyyyddd'
TM1	4 bytes	SMF time interpreted as Z'hhmmss'
TM2	4 bytes	SMF time interpreted as Z'hhmm'
TM3	4 bytes	SMF time interpreted as Z'hh'
TM4	4 bytes	SMF time interpreted as Z'hhmmssxx'
TC1	8 bytes	TOD time interpreted as Z'hhmmss'
TC2	8 bytes	TOD time interpreted as Z'hhmm'
TC3	8 bytes	TOD time interpreted as Z'hh'
TC4	8 bytes	TOD time interpreted as Z'hhmmssxx'
TE1	8 bytes	ETOD time interpreted as Z'hhmmss'
TE2	8 bytes	ETOD time interpreted as Z'hhmm'
TE3	8 bytes	ETOD time interpreted as Z'hh'
TE4	8 bytes	ETOD time interpreted as Z'hhmmssxx'
Note: See Appendix C, "Data Format Descriptions" for detailed format descriptions.		

For a CSF or FS format field:

- A maximum of 31 digits is allowed. If a CSF or FS value with 32 digits is found, the leftmost digit will be treated as a positive sign indicator.

For a UFF or SFF format field:

- A maximum of 31 digits is allowed. If a UFF or SFF value with more than 31 digits is found, the leftmost digits will be ignored.

For a ZD or PD format field:

- An invalid digit results in a data exception (0C7 ABEND) or incorrect numeric output; A-F are invalid digits. ICETOOL's VERIFY or DISPLAY operator can be used to identify decimal values with invalid digits.
- A value is treated as positive if its sign is F, E, C, A, 8, 6, 4, 2, or 0.
- A value is treated as negative if its sign is D, B, 9, 7, 5, 3, or 1.

For a PD0 format field:

- The first digit is ignored.
- An invalid digit other than the first results in a data exception (0C7 ABEND) or incorrect numeric output; A-F are invalid digits.
- The sign is ignored and the value is treated as positive.

For an FL format field:

- The normalized or unnormalized FL (hexadecimal floating-point) value is converted to a signed integer in the range -9223372036854775808 to 9223372036854775807. The fractional part of the FL value is lost, and in some cases the signed integer may be one of a number of possible signed integers for the FL value depending on its precision. Converted values less than -9223372036854775808 are set to -9223372036854775808. Converted values greater than 9223372036854775807 are set to 9223372036854775807.
- If you are not running in z/Architecture mode, specifying an FL format field results in an error message and termination.

For a DT1, DT2, or DT3 format field:

- An invalid SMF date can result in a data exception (0C7 ABEND) or an incorrect ZD date.
- SMF date values are always treated as positive.

For a DC1, DC2, DC3, DE1, DE2, or DE3 format field:

- TOD and ETOD date values are always treated as positive.

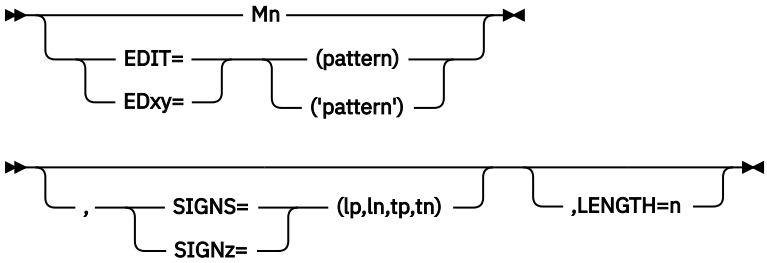
For a TM1, TM2, TM3, or TM4 format field:

- An invalid SMF time can result in an incorrect ZD time.
- SMF time values are always treated as positive.

For a TC1, TC2, TC3, TC4, TE1, TE2, TE3, or TE4 format field:

- TOD and ETOD time values are always treated as positive.

edit



Specifies how the numeric field is to be edited for output. If an Mn, EDIT, or EDxy parameter is not specified:

- a DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3, or TM4 field is edited using the M11 edit mask.
- a BI, FI, PD, PD0, ZD, FL, CSF, FS, UFF, or SFF field is edited using the M0 edit mask.

Mn
specifies one of twenty-seven pre-defined edit masks (M0-M26) for presenting numeric data. If these pre-defined edit masks are not suitable for presenting your numeric data, the EDIT parameter gives you the flexibility to define your own edit patterns.

The twenty-seven pre-defined edit masks can be represented as follows:

Table 50. Edit Mask Patterns

Mask	Pattern	Examples	
		Value	**Result**
M0	IIIIIIIIIIIIIIIIIIIIIIIIIIIITS	+01234	1234
		-00001	1-
M1	TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTS	-00123	00123-
		+00123	00123
M2	II,III,III,III,III,III,III,III,III,IIT.TTS	+123450	1,234.50
		-000020	0.20-
M3	II,III,III,III,III,III,III,III,III,IIT.TTCR	-001234	12.34CR
		+123456	1,234.56
M4	SII,III,III,III,III,III,III,III,III,IIT.TT	+0123456	+1,234.56
		-1234567	-12,345.67
M5	SII,III,III,III,III,III,III,III,III,IIT.TTS	-001234	(12.34)
		+123450	1,234.50
M6	III-TTT-TTTT	00123456	012-3456
		12345678	1-234-56788
M7	TTT-TT-TTTT	00123456	000-12-3456
		12345678	012-34-5678
M8	IT:TT:TT	030553	3:05:53
		121736	12:17:36
M9	IT/TT/TT	123004	12/30/04
		083104	8/31/04
M10	IIIIIIIIIIIIIIIIIIIIIIIIIIIIT	01234	1234
		00000	0
M11	TTTTTTTTTTTTTTTTTTTTTTTTTTTTTT	00010	00010
		01234	01234
M12	SI,III,III,III,III,III,III,III,III,III,IIT	+1234567	1,234,567
		-0012345	-12,345
M13	SI.III.III.III.III.III.III.III.III.III.IIT	+1234567	1.234.567
		-0012345	-12.345
M14	SI III III III III III III III III III IITS	+1234567	1 234 567
		-0012345	(12 345)
M15	I III III III III III III III III III IITS	+1234567	1 234 567
		-0012345	12 345-

Table 50. Edit Mask Patterns (continued)

Mask	Pattern	Examples	
		Value	Result
M16	SI III III III III III III III III III IIT	+1234567	1 234 567
		-0012345	-12 345
M17	SI'III'III'III'III'III'III'III'III'IIT	+1234567	1'234'567
		-0012345	-12'345
M18	SII,III,III,III,III,III,III,III,III,IIT.TT	+0123456	1,234.56
		-1234567	-12,345.67
M19	SII.III.III.III.III.III.III.III.IIT,TT	+0123456	1.234,56
		-1234567	-12.345,67
M20	SI III III III III III III III III IIT,TTS	+0123456	1 234,56
		-1234567	(12 345,67)
M21	II III III III III III III III III IIT,TTS	+0123456	1 234,567
		-1234567	12 345,67-
M22	SI III III III III IIII III III III IIT,TT	+0123456	1 234,56
		-1234567	-12 345,67
M23	SII'III'III'III'III'III'III'III'IIT.TT	+0123456	1'234.56
		-1234567	-12'345.67
M24	SII'III'III'III'III'III'III'III'IIT,TT	+0123456	1'234,56
		-1234567	-12'345,67
M25	SIIIIIIIIIIIIIIIIIIIIIIIIIIIIT	+01234	1234
		-00001	-1
M26	STTTTTTTTTTTTTTTTTTTTTTTTTTTTT	1234	+01234
		-1	-00001

The elements used in the representation of the edit masks in Table 50 on page 264 are as follows:

- **I** indicates a leading insignificant digit. If zero, this digit will not be shown.
- **T** indicates a significant digit. If zero, this digit will be shown.
- **CR** (in M3) is printed to the right of the digits if the value is negative; otherwise, two blanks are printed to the right of the digits.
- **S** indicates a sign. If it appears as the first character in the pattern, it is a leading sign. If it appears as the last character in the pattern, it is a trailing sign. If S appears as both the first and last characters in a pattern (example: M5), the first character is a leading sign and the last character is a trailing sign. Four different sign values are used: leading positive sign (lp), leading negative sign (ln), trailing positive sign (tp) and trailing negative sign (tn). Their applicable values for the Mn edit masks are:

Table 51. Edit Mask Signs

Mask	lp	ln	tp	tn
M0	none	none	blank	-

Table 51. Edit Mask Signs (continued)

Mask	lp	ln	tp	tn
M1	none	none	blank	-
M2	none	none	blank	-
M3	none	none	none	none
M4	+	-	none	none
M5	blank	(	blank	)
M6	none	none	none	none
M7	none	none	none	none
M8	none	none	none	none
M9	none	none	none	none
M10	none	none	none	none
M11	none	none	none	none
M12	blank	-	none	none
M13	blank	-	none	none
M14	blank	(	blank	)
M15	none	none	blank	-
M16	blank	-	none	none
M17	blank	-	none	none
M18	blank	-	none	none
M19	blank	-	none	none
M20	blank	(	blank	)
M21	none	none	blank	-
M22	blank	-	none	none
M23	blank	-	none	none
M24	blank	-	none	none
M25	blank	-	none	none
M26	+	-	none	none

- any other character (for example, /) will be printed as shown, subject to certain rules to be subsequently discussed.

The implied length of the edited output field depends on the number of digits and characters needed for the pattern of the particular edit mask used. The LENGTH parameter can be used to change the implied length of the edited output field.

The number of digits needed depends on the format and length of the numeric field as follows:

Table 52. Digits Needed for Numeric Fields

Format	Input Length	Digits Needed
ZD	m	m

Table 52. Digits Needed for Numeric Fields (continued)

Format	Input Length	Digits Needed
PD	m	2m-1
PD0	m	2m-2
BI, FI	1	3
BI, FI	2	5
BI, FI	3	8
BI, FI	4	10
BI, FI	5	13
BI, FI	6	15
BI, FI	7	17
BI, FI	8	20
FL	4 or 8	20
CSF or FS	32	31
CSF or FS	m (less than 32)	m
UFF, SFF	32 to 44	31
UFF, SFF	m (less than 32)	m

The length of the output field can be represented as follows for each pattern, where d is the number of digits needed, as shown in Table 52 on page 266, and the result is rounded down to the nearest integer:

Table 53. Edit Mask Output Field Lengths

Mask	Output Field Length	Example	
		Input (f,m)	Output Length
M0	d + 1	ZD,3	4
M1	d + 1	PD,10	20
M2	d + 1 + d/3	BI,4	14
M3	d + 2 + d/3	UFF,20	28
M4	d + 1 + d/3	PD,8	21
M5	d + 2 + d/3	FI,3	12
M6	12	ZD,10	12
M7	11	PD,5	11
M8	8	ZD,6	8
M9	8	PD,4	8
M10	d	BI,6	15
M11	d	PD,5	9
M12	d + 1 + (d - 1)/3	PD,3	7
M13	d + 1 + (d - 1)/3	FS,5	7

Table 53. Edit Mask Output Field Lengths (continued)

Mask	Output Field Length	Example	
		Input (f,m)	Output Length
M14	d + 2 + (d - 1)/3	ZD,5	8
M15	d + 1 + (d - 1)/3	FI,3	11
M16	d + 1 + (d - 1)/3	SFF,41	42
M17	d + 1 + (d - 1)/3	FI,4	14
M18	d + 1 + d/3	BI,4	14
M19	d + 1 + d/3	PD,8	21
M20	d + 2 + d/3	FI,3	12
M21	d + 1 + d/3	ZD,3	5
M22	d + 1 + d/3	BI,2	7
M23	d + 1 + d/3	PD,6	15
M24	d + 1 + d/3	ZD,21	29
M25	d + 1	CSF,16	17
M26	d + 1	FL,4	21

To illustrate conceptually how DFSORT produces the edited output from the numeric value, consider the following example:

```
OUTFIL OUTREC=(5,7,ZD,M5)

with ZD values of C'0123456'(+0123456)
and C'000302J' (-0003021)
```

As shown in the preceding tables, it is determined that:

- The general pattern for M5 is: SI,III,...,IIT.TTS
- The signs to be used are blank for leading positive sign, C'(' for leading negative sign, blank for trailing positive sign and C')' for trailing negative sign
- The number of digits needed is 7
- The length of the output field is 11 (7 + 2 + 7/3)
- The specific pattern for the output field is thus: C'SII,IIT.TTS'

The digits of C'0123456' are mapped to the pattern, resulting in C'S01,234.56S'. Because the value is positive, the leading sign is replaced with blank and the trailing sign is replaced with blank, resulting in C' 01,234.56 '. Finally, all digits before the first non-zero digit (1 in this case), are replaced with blanks, resulting in the final output of C' 1,234.56 '.

The digits of C'000302J' are mapped to the pattern, resulting in C'S00,030.21S'. Because the value is negative, the leading sign is replaced with C'(' and the trailing sign is replaced with C')' resulting in C'(00,030.21)'. All digits before the first non-zero digit (3 in this case), are replaced with blanks, resulting in C'(30.21)'. Finally, the leading sign is "floated" to the right, next to the first non-zero digit, resulting in the final output of C' (30.21)'.

To state the rules in more general terms, the steps DFSORT takes conceptually to produce the edited output from the numeric value are as follows:

- Determine the specific pattern and its length, using the preceding tables.
- Map the digits of the numeric value to the pattern.

- If the value is positive, replace the leading and trailing signs (if any) with the characters for positive values shown in Table 51 on page 265. Otherwise, replace the leading and trailing signs (if any) with the characters for negative values shown in that same table.
- Replace all digits before the first non-zero (I) or significant digit (T) with blanks.
- Float the leading sign (if any) to the right, next to the first non-zero (I) or significant digit (T).

The following additional rule applies to edit masks:

- The specific pattern is determined from the general pattern by including signs, the rightmost digits needed as determined from the input format and length, and any characters in between those rightmost digits. This may unintentionally truncate significant digits (T). As an example, if you specify 5,1,ZD,M4, the length of the output field will be 2 (1 + 1 + 1/3). The general pattern for M4 is SI,III,...,IIT.TT, but the specific pattern will be ST (the leading sign and the rightmost digit).

EDIT
specifies an edit pattern for presenting numeric data. If the pre-defined edit masks (M0-M26) are not suitable for presenting your numeric data, EDIT gives you the flexibility to define your own edit patterns. The elements you use to specify the pattern are the same as those used for the edit masks: I, T, S, and printable characters. However, S will not be recognized as a sign indicator unless the SIGNS parameter is also specified.

pattern
specifies the edit pattern to be used. Not enclosing the pattern in single apostrophes restricts you from specifying the following characters in the pattern: blank, apostrophe, unbalanced left or right parentheses, and hexadecimal digits 20, 21, and 22. For example, EDIT=((IIT.TT)) is valid, whereas EDIT=(C)ITT.TT), EDIT=(I / T) and EDIT=(S'II.T) are not.

The maximum number of digits (I's and T's) you specify in the pattern must not exceed 31. The maximum length of the pattern must not exceed 44 characters.

'pattern'
specifies the edit pattern to be used. Enclosing the pattern in single apostrophes allows you to specify any character in the pattern except hexadecimal digits 20, 21, or 22. If you want to include a single apostrophe in the pattern, you must specify it as two single apostrophes, which will be counted as a single character in the pattern. As examples, EDIT=('C)ITT.TT'), EDIT=('I / T'), and EDIT=('S'II.T') are all valid.

The maximum number of digits (I's and T's) you specify in the pattern must not exceed 31. The maximum length of the pattern must not exceed 44 characters.

The implied length of the edited output field is the same as the length of the pattern. The LENGTH parameter can be used to change the implied length of the edited output field.

To illustrate conceptually how DFSORT produces the edited output from the numeric value, consider the following example:

```
OUTFIL OUTREC=(1,5,ZD,EDIT=(**I/ITTTCR))

with ZD values of C'01230'(+1230)
and C'0004J' (-41)
```

The digits of C'01230' are mapped to the pattern, resulting in C'**0/1230CR'. Because the value is positive, the characters (C'CR') to the right of the last digit are replaced with blanks, resulting in C'**0/1230 '. All digits before the first non-zero digit (1 in this case) are replaced with blanks, resulting in C'** /1230 '. Finally, all characters before the first digit in the pattern are floated to the right, next to the first non-zero digit, resulting in C' **1230 '.

The digits of C'0004J' are mapped to the pattern, resulting in C'**0/0041CR'. Because the value is negative, the characters (C'CR') to the right of the last digit are kept. All digits before the first T digit are replaced with blanks, resulting in C'** / 041CR'. Finally, all characters before the first digit in the pattern are floated to the right, next to the first non-zero digit, resulting in C' **041CR'.

In general terms, the steps DFSORT takes conceptually to produce the edited output from the numeric value are as follows:

- Map the digits of the numeric value to the pattern, padding on the left with zeros, if necessary.
- If the value is positive, replace the leading and trailing signs (if any) with the characters for positive values specified by the SIGNS parameter and replace any characters between the last digit and the trailing sign (if any) with blanks. Otherwise, replace the leading and trailing signs (if any) with the characters for negative values specified by the SIGNS parameter and keep any characters between the last digit and the trailing sign (if any).
- Replace all digits before the first non-zero (I), significant digit (T), or significant decimal point with blanks.
- Float all characters (if any) before the first digit in the pattern to the right, next to the first non-zero (I) or significant digit (T).

The following additional rules apply to edit patterns:

- An insignificant digit (I) after a significant digit (T) or significant decimal point is treated as a significant digit.
- If SIGNS is specified, an S in the first or last character of the pattern is treated as a sign; an S anywhere else in the pattern is treated as the letter S. If SIGNS is not specified, an S anywhere in the pattern is treated as the letter S.
- If the pattern contains fewer digits than the value, the leftmost digits of the value will be lost, intentionally or unintentionally. As an example, if you specify 5,5,ZD,EDIT=(IIT) for a value of C'12345', the result will be C'345'. As another example, if you specify 1,6,ZD,EDIT=($IIT.T) for a value of C'100345', the result will be C' $34.5'.

EDxy
specifies an edit pattern for presenting numeric data. EDxy is a special variation of EDIT that allows other characters to be substituted for I and T in the pattern. For example, if you use EDAB instead of EDIT, you must use A in the pattern instead of I and use B instead of T to represent digits. x and y must not be the same character. If SIGNS is specified, x and y must not be S. If SIGNz is specified, x and y must not be the same character as z. You can select x and y from: A-Z, #, $, @, and 0-9.

SIGNS
specifies the sign values to be used when editing numeric values according to the edit mask (Mn) or pattern (EDIT or EDxy). You can specify any or all of the four sign values. Any value not specified must be represented by a comma. Blank will be used for any sign value you do not specify. As examples, SIGNS=(+,-) specifies + for lp, - for ln, blank for tp, and blank for tn; SIGNS=(,,+,-) specifies blank for lp, blank for ln, + for tp, and - for tn.

lp
specifies the value for the leading positive sign. If an S is specified as the first character of the edit mask or pattern and the value is positive, the lp value will be used as the leading sign.

ln
specifies the value for the leading negative sign. If an S is specified as the first character of the edit mask or pattern and the value is negative, the ln value will be used as the leading sign.

tp
specifies the value for the trailing positive sign. If an S is specified as the last character of the edit mask or pattern and the value is positive, the tp value will be used as the trailing sign.

tn
specifies the value for the trailing negative sign. If an S is specified as the last character of the edit mask or pattern and the value is negative, the tn value will be used as the trailing sign.

If you want to use any of the following characters as sign values, you must enclose them in single apostrophes: comma, blank, or unbalanced left or right parentheses. A single apostrophe must be specified as four single apostrophes (that is, two single apostrophes enclosed in single apostrophes).

A semicolon cannot be substituted for a comma as the delimiter between sign characters.

SIGNz
specifies the sign values to be used when editing numeric values according to the edit pattern (EDIT or EDxy). SIGNz is a special variation of SIGNS which allows another character to be substituted for S

in the pattern. For example, if you use SIGNX instead of SIGNS, you must use X in the pattern instead of S to identify a sign. If EDIT is specified, z must not be I or T. If EDxy is specified, z must not be the same character as either x or y. You can select z from: A-Z, #, $, @, and 0-9.

LENGTH
specifies the length of the edited output field. If the implied length of the edited output field produced using an edit mask or edit pattern is not suitable for presenting your numeric data, LENGTH can be used to make the edited output field shorter or longer.

n
specifies the length of the edited output field. The value for n must be between 1 and 44.

LENGTH does not change the pattern used, only the length of the resulting edited output field. For example, as discussed previously for Mn, if you specify:

```
OUTFIL OUTREC=(5,1,ZD,M4)
```

the pattern will be C'ST' rather than C'ST.TT' because the digit length is 1. Specifying:

```
OUTFIL OUTREC=(5,1,ZD,M4,LENGTH=5)
```

will change the pattern to C' ST', not to C'ST.TT'.

If you specify a value for n that is shorter than the implied length, truncation will occur on the left after editing. For example, if you specify:

```
OUTFIL OUTREC=(1,5,ZD,EDIT=($IIT.TT),LENGTH=5)
```

with a value of C'12345', editing according to the specified $IIT.TT pattern will produce C'$123.45', but the specified length of 5 will truncate this to C'23.45'.

If you specify a value for n that is longer than the implied length, padding on the left with blanks will occur after editing. For example, if you specify:

```
OUTFIL OUTREC=(1,5,ZD,EDIT=($IIT.TT),LENGTH=10)
```

with a value of C'12345', editing according to the specified $IIT.TT pattern will produce C'$123.45', but the specified length of 10 will pad this to C' $123.45'.

Sample Syntax:

```
OUTFIL FNAMES=OUT1,OUTREC=(5:21,19,ZD,M19,35:46,5,ZD,M13)
OUTFIL FILES=1,OUTREC=(4,8,BI,C' * ',13,8,BI,80:X),
   ENDREC=10,OMIT=(4,8,BI,EQ,13,8,BI)
OUTFIL FILES=(2,3),
  OUTREC=(11:55,6,FS,SIGNS=(,,+,-),LENGTH=10,
          31:(41,4,PD),EDIT=(**II,IIT.TTXS),SIGNS=(,,+,-))
```

%nn,f,edit or (%nn,f),edit
specifies that an edited numeric parsed input field is to appear in the reformatted OUTFIL output record. See PARSE for details of parsed fields. See p,m,f,edit or (p,m,f),edit for further details.

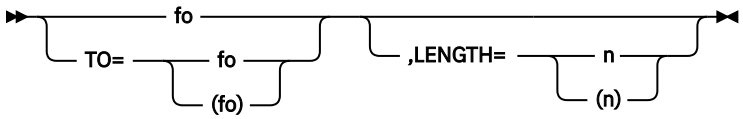
p,m,f,to or (p,m,f),to
specifies that a converted numeric input field is to appear in the reformatted OUTFIL output record. You can convert BI, FI, PD, PD0, ZD, FL, CSF, FS, UFF, SFF, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3, or TM4 fields to BI, FI, PD, PDC, PDF, ZD, ZDF, ZDC, or CSF/FS fields.

p
See p under p,m,f,edit.

m
See m under p,m,f,edit.

f
See f under p,m,f,edit.

to



specifies how the numeric field is to be converted for output.

fo

specifies the format for the output field, which can be BI, FI, PD, PDC, PDF, ZD, ZDF, ZDC, CSF or FS. Any one of these output field formats (fo) can be used with any one of the input field formats (f).

If you do not specify the LENGTH parameter, DFSORT will determine the implied length of the output field from the length (m) and format (f) of the input field and the format (fo) of the output field. The implied length of the output field can be represented as follows for each output format, where d is the number of digits needed for the input field as shown in Table 52 on page 266, and the result is rounded down to the nearest integer:

Table 54. To Output Field Lengths

Output Format	Output Length	Example Input (f,m)	Example Output Length
BI with d <= 9	4	FS,9	4
BI with d > 9	8	FS,10	8
FI with d <= 9	4	ZD,7	4
FI with d > 9	8	ZD,12	8
PD, PDC, or PDF	d/2 + 1	BI,4	6
ZD, ZDF, or ZDC	d	PD,9	17
CSF or FS	d + 1	FI,4	21

For ZD or ZDF output, F is used as the positive sign and D is used as the negative sign. For ZDC output, C is used as the positive sign.

For PD or PDC output, C is used as the positive sign and D is used as the negative sign. For PDF output, F is used as the positive sign.

For CSF or FS output, blank is used as the positive sign, - is used as the negative sign and leading zeros are suppressed.

For ZD, ZDF, ZDC, PD, PDC, PDF, CSF, or FS output, the maximum output value is 9999999999999999999999999999999 (31 digits) and the minimum output value is -9999999999999999999999999999999 (31 digits), which correspond to the maximum and minimum input values, respectively.

For BI output:

- An input value greater than 18446744073709551615 (X'FFFFFFFFFFFFFFFF') produces an output value of 18446744073709551615 (X'FFFFFFFFFFFFFFFF').
- An input value less than zero produces an absolute output value. For example, an input value of P'-5000' produces a BI output value of 5000 (X'1388').

For FI output, an input value greater than 9223372036854775807 (X'7FFFFFFFFFFFFFFF') produces an output value of 9223372036854775807 (X'7FFFFFFFFFFFFFFF'), and an input value less than -9223372036854775808 (X'8000000000000000') produces an output value of -9223372036854775808 (X'8000000000000000').

fo, TO=fo and TO=(fo) are interchangeable except that:

- fo must be specified before the LENGTH parameter whereas TO can be specified before or after the LENGTH parameter.
- TO=fo or TO=(fo) should be used after a symbol rather than fo to prevent the misinterpretation of fo as f. See the discussion of OUTFIL OUTREC in Chapter 7 for details.

LENGTH
specifies the length of the converted output field. If the implied length of the output field is not suitable, LENGTH can be used to make the output field shorter or longer.

n
specifies the length of the converted output field. The value for n must be between 1 and 44.

If you specify a value for n that is shorter than the implied length, truncation on the left will occur after conversion. For example, if you specify:

```
OUTFIL OUTREC=(1,8,ZD,TO=PD,LENGTH=3)
```

with values of ZL8'-12345678' (X'F1F2F3F4F5F6F7D8') and ZL8'58' (X'F0F0F0F0F0F0F5F8'), conversion with the implied length (5) will produce PL5'-12345678' (X'012345678D') and PL5'58' (X'0000000058C'). The specified length of 3 will then result in truncation to PL3'-45678' (X'45678D') and PL3'58' (X'00058C').

If you specify a value for n that is longer than the implied length, padding on the left will occur after conversion using:

- Blanks for CSF and FS output values
- Character zeros for ZD output values
- Binary zeros for PD and BI output values
- Binary zeros for positive FI output values
- Binary ones for negative FI output values

For example, if you specify:

```
OUTFIL OUTREC=(1,4,ZD,TO=FI,LENGTH=6)
```

with values of ZL4'-1234' (X'F1F2F3D4') and ZL4'58' (X'F0F0F5F8'), conversion with the implied length (4) will produce FL4'-1234' (X'FFFFFB2E') and FL4'58' (X'000004D2'). The specified length of 6 will then result in padding to FL6'-1234' (X'FFFFFFFFFB2E') and FL6'58' (X'00000000003A').

Sample Syntax:

```
OUTFIL OUTREC=((21,5,ZD),PD,X,8,4,ZD,TO=FI,LENGTH=2)
```

%nn,f,to or (%nn,f),to
specifies that a converted numeric parsed input field is to appear in the reformatted OUTFIL output record. See PARSE for details of parsed fields. See p,m,f,to or (p,m,f),to for further details.

deccon,edit or (deccon),edit
specifies that an edited decimal constant is to appear in the reformatted OUTFIL output record. The decimal constant must be in the form +n or -n where n is 1 to 31 decimal digits. The sign (+ or -) must be specified. A decimal constant produces a signed, 31-digit zoned decimal (ZD) result to be edited as specified. If an Mn, EDIT, or EDxy parameter is not specified, the decimal constant is edited using the M0 edit mask.

The default number of digits (d) used for editing is 15 for a decimal constant with 1 to 15 significant digits, or 31 for a decimal constant with 16 to 31 significant digits. If EDIT or EDxy is specified, the number of digits in the pattern (I's and T's) is used.

See edit under p,m,f,edit for further details on the edit fields you can use.

Sample Syntax:

```
OUTFIL OUTREC=(5,8,+4096,2X,-17,M18,LENGTH=7,2X,
  (+2000000),EDIT=(STTTTT.TT),SIGNS=(+))
```

deccon,to or (deccon),to

specifies that a converted decimal constant is to appear in the reformatted OUTFIL output record. The decimal constant must be in the form +n or -n where n is 1 to 31 decimal digits. The sign (+ or -) must be specified. A decimal constant produces a signed, 31-digit zoned decimal (ZD) result to be converted as specified.

The default number of digits (d) used for conversion is 15 for a decimal constant with 1 to 15 significant digits, or 31 for a decimal constant with 16 to 31 significant digits.

See to under p,m,f,to for further details on the to fields you can use.

Sample Syntax:

```
OUTFIL FNAMES=OUT1,
  OUTREC=(6:+0,TO=PD,LENGTH=6,+0,TO=PD,LENGTH=6,/,
         6:(-4096),ZD,LENGTH=12)
```

arexp,edit or (arexp),edit

Specifies that the edited result of an arithmetic expression is to appear in the reformatted OUTFIL output record. An arithmetic expression takes the form:

```
term,operator,term<,operator,...>
```

where:

- **term** is a field (p,m,f), a parsed field (%nn,f), or a decimal constant (+n or -n). See p,m,f under p,m,f,edit for details on the fields you can use. See deccon under deccon,edit for details on the decimal constants you can use.
- **operator** is MIN (minimum), MAX (maximum), MUL (multiplication), DIV (division), MOD (modulus), ADD (addition) or SUB (subtraction).

The order of evaluation precedence for the operators is as follows unless it is changed by parentheses:

1. MIN and MAX
2. MUL, DIV and MOD
3. ADD and SUB

The intermediate or final result of a DIV operation is rounded down to the nearest integer. The intermediate or final result of a MOD operation is an integer remainder with the same sign as the dividend. If an intermediate or final result of an arithmetic expression overflows 31 digits, the overflowing intermediate or final result will be truncated to 31 digits, intentionally or unintentionally. If an intermediate or final result of an arithmetic expression requires division or modulus by 0, the intermediate or final result will be set to 0, intentionally or unintentionally.

An arithmetic expression produces a signed, 31-digit zoned decimal (ZD) result to be edited as specified. If an Mn, EDIT, or EDxy parameter is not specified, the result is edited using the M0 edit mask.

The default number of digits (d) used for editing is 15 if **every** term in the expression is one of the following:

- a 1-4 byte BI or FI field
- a 1-8 byte PD field
- a 1-15 byte ZD, FS, CSF, UFF or SFF field
- a decimal constant with 1-15 significant digits.

The default number of digits (d) used for editing is 31 if **any** term in the expression is one of the following:

- a 5-8 byte BI or FI field
- a 9-16 byte PD field
- a 16-31 byte ZD field
- a 4-byte or 8-byte FL field
- a 16-32 byte FS or CSF field
- a 16-44 byte UFF or SFF field
- a decimal constant with 16-31 significant digits.

If EDIT or EDxy is specified, the number of digits in the pattern (I's and T's) is used.

See edit under p,m,f,edit for further details on the edit fields you can use.

Sample Syntax:

```
OUTFIL OUTREC=(5:C'% REDUCTION FOR ',21,8,C' IS ',
  ((11,6,ZD,SUB,31,6,ZD),MUL,+1000),DIV,11,6,ZD,
  EDIT=(SIIT.T),SIGNS=(+,-))
```

arexp,to or (arexp),to

specifies that the converted result of an arithmetic expression is to appear in the reformatted OUTFIL output record. See arexp under arexp,edit for further details on arithmetic expressions.

An arithmetic expression produces a signed, 31-digit zoned decimal (ZD) result to be converted as specified.

The default number of digits (d) used for conversion is 15 if **every** term in the expression is one of the following:

- a 1-4 byte BI or FI field
- a 1-8 byte PD field
- a 1-15 byte ZD, FS, CSF, UFF or SFF field
- a decimal constant with 1-15 significant digits.

The default number of digits (d) used for conversion is 31 if **any** term in the expression is one of the following:

- a 5-8 byte BI or FI field
- a 9-16 byte PD field
- a 16-31 byte ZD field
- a 4-byte or 8-byte FL field
- a 16-32 byte FS or CSF field
- a 16-44 byte UFF or SFF field
- a decimal constant with 16-31 significant digits.

See to under p,m,f,to for further details on the to fields you can use.

Sample Syntax:

```
OUTFIL FNAMES=OUT,
  OUTREC=(61,3,X,
          35,6,FS,ADD,45,6,FS,ADD,55,6,FS,TO=FS,LENGTH=7,X,
          (5,11,PD,MIN,112,11,PD),PD,LENGTH=11,X,
          64,5,SEQNUM,5,ZD)
```

p,m,Y2x or p,m,Y4x

Specifies that an input date field is to be edited. Real Y2x dates are edited using the century window established by the Y2PAST option in effect. Y2x and Y4x dates with special indicators are expanded appropriately (for example, p,6,Y2T transforms C'000000' to C'00000000').

Editing involving an input date with an invalid digit (A-F) can result in a data exception (0C7 ABEND) or an incorrect output value. Editing involving an invalid input date can result in an invalid output value.

p
See p under p,m,a.

m
Specifies the length in bytes of the Y2x (two-digit year) or Y4x (four-digit year) date field.

Y2x or Y4x
Specifies the Y2 or Y4 format for the date field. See Appendix C, “Data format descriptions,” on page 823 for detailed format descriptions.

Sample Syntax:

```
OUTFIL BUILD=(21,3,Y2U,X,3,8,Y4W)
```

Table 55 on page 276 shows the output produced for each type of edited date.

Table 55. Input and result fields for Yxx date editing

m,Yxx	Input date	Output for p,m,Yxx
3,Y2T	C'yyx' or Z'yyx'	C'ccyyx'
4,Y2T	C'yyxx' or Z'yyxx'	C'ccyyxx'
5,Y2T	C'yyddd' or Z'yyddd'	C'ccyyddd'
6,Y2T	C'yymmdd' or Z'yymmdd'	C'ccyymmdd'
7,Y4T	C'ccyyddd' or Z'ccyyddd'	C'ccyyddd'
8,Y4T	C'ccyymmdd' or Z'ccyymmdd'	C'ccyymmdd'
2,Y2U	P'yyx'	C'ccyyx'
3,Y2V	P'yyxx'	C'ccyyxx'
3,Y2U	P'yyddd'	C'ccyyddd'
4,Y2V	P'yymmdd'	C'ccyymmdd'
4,Y4U	P'ccyyddd'	C'ccyyddd'
5,Y4V	P'ccyymmdd'	C'ccyymmdd'
3,Y2W	C'xyy' or Z'xyy'	C'xccyy'
4,Y2W	C'xxyy' or Z'xxyy'	C'xxccyy'
5,Y2W	C'dddyy' or Z'dddyy'	C'dddccyy'
6,Y2W	C'mmddyy' or Z'mmddyy'	C'mmddccyy'
7,Y4W	C'dddccyy' or Z'dddccyy'	C'dddccyy'
8,Y4W	C'mmddccyy' or Z'mmddccyy'	C'mmddccyy'
2,Y2X	P'xyy'	C'xccyy'
3,Y2Y	P'xxyy'	C'xxccyy'
3,Y2X	P'dddyy'	C'dddccyy'
4,Y2Y	P'mmddyy'	C'mmddccyy'
4,Y4X	P'dddccyy'	C'dddccyy'
5,Y4Y	P'mmddccyy'	C'mmddccyy'
2,Y2C	C'yy'	C'ccyy'
2,Y2Z	Z'yy'	C'ccyy'
2,Y2S	C'yy'	C'ccyy'
2,Y2P	P'yy'	C'ccyy'
1,Y2D	X'yy'	C'ccyy'
1,Y2B	B'yy'	C'ccyy'

%nn,Y2x or %nn,Y4x
Specifies that a parsed input date field is to be edited. See PARSE for details of parsed fields. See "p,m,Y2x or p,m,Y4x" for further details.

p,m,Y2x,edit or p,m,Y4x,edit
Specifies that the output for a p,m,Yxx input date field as shown in Table 55 on page 276 is to be further edited according to the edit parameters you specify. For example, if you specify:

```
OUTFIL BUILD=(28,5,Y4V,EDIT=(TTTT-TT-TT)
```

The P'ccyymmdd' (X'0ccyymmddC') date value will be transformed to a C'ccyymmdd' date value and then edited to a C'ccyy-mm-dd' date value.

See "p,m,Y2x or p,m,Y4x" and "p,m,f,edit" for related details.

nn,Y2x,edit or %nn,Y4x,edit
Specifies that the output for a parsed Yxx input date field as shown in Table 55 on page 276 is to be further edited according to the edit parameters you specify. See PARSE for details of parsed fields. See "p,m,Y2x,edit or p,m,Y4x,edit" for further details.

p,m,Y2x,to or p,m,Y4x,to
Specifies that an input date field as shown in Table 55 on page 276 is to be converted according to the to parameters you specify. For example, if you specify:

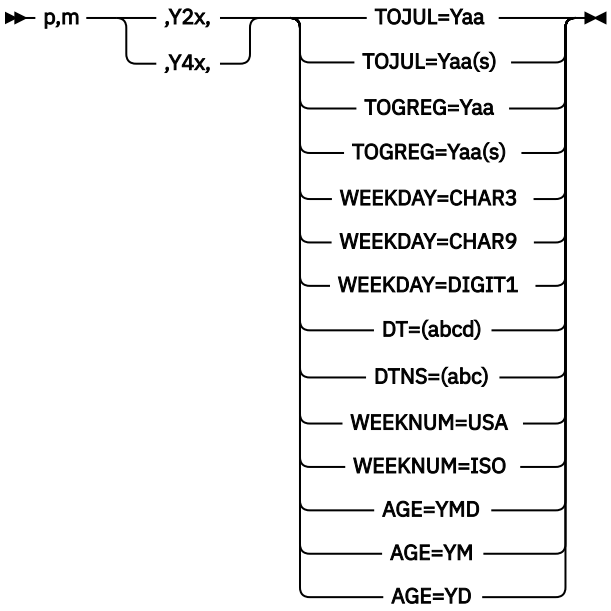
```
OUTFIL BUILD=(5,6,Y2W,TO=PD,LENGTH=5)
```

The C'mmddyy' date value will be transformed to a Z'mmddccyy' date value and then converted to a P'mmddccyy' (X'0mmddccyyC') date value.

See "p,m,Y2x or p,m,Y4x" and "p,m,f,to" for related details.

%nn,Y2x,to or %nn,Y4x,to
Specifies that a parsed input date field as shown in Table 55 on page 276 is to be converted according to the to parameters you specify. See PARSE for details of parsed fields. See "p,m,Y2x,to or p,m,Y4x,to" for further details.

p,m,Y2x,todate or p,m,Y4x,todate



Specifies that an input date field of one type is to be converted to a corresponding output date field of another type. You can convert date fields between combinations of 2-digit and 4-digit year dates, CH/ZD and PD dates, and Julian and Gregorian dates. You can also convert a date field to a corresponding day of the week in several forms.

Each type of date field you can use as input for date conversions is shown in Table 56 on page 278.

Table 56. Input fields for p,m,Yxx date conversion

m,Yxx	Input date
5,Y2T	C'yyddd' or Z'yyddd'
6,Y2T	C'yymmdd' or Z'yymmdd'
7,Y4T	C'ccyyddd' or Z'ccyyddd'
8,Y4T	C'ccyymmdd' or Z'ccyymmdd'
3,Y2U	P'yyddd'
4,Y2V	P'yymmdd'
4,Y4U	P'ccyyddd'
5,Y4V	P'ccyymmdd'
5,Y2W	C'dddyy' or Z'dddyy'
6,Y2W	C'mmddyy' or Z'mmddyy'
7,Y4W	C'dddccyy' or Z'dddccyy'
8,Y4W	C'mmddccyy' or Z'mmddccyy'
3,Y2X	P'dddyy'
4,Y2Y	P'mmddyy'
4,Y4X	P'dddccyy'
5,Y4Y	P'mmddccyy'

p
See p under p,m,a.

m
Specifies the length in bytes of the Y2x (two-digit year) or Y4x (four-digit year) date field.

Y2x or Y4x
Specifies the Y2 or Y4 format for the date field. See Appendix C, “Data format descriptions,” on page 823 for detailed format descriptions.

todate
Specifies the type of date conversion to be performed.

p,m,Yxx,TOJUL=Yaa
Converts the input date to a Julian output date.

p,m,Yxx,TOJUL=Yaa(s)
Converts the input date to a Julian output date with s separators. s can be any character except a blank.

p,m,Yxx,TOGREG=Yaa
Converts the input date to a Gregorian output date.

p,m,Yxx,TOGREG=Yaa(s)
Converts the input date to a Gregorian output date with s separators. s can be any character except a blank.

The output date field created by each valid TOJUL and TOGREG combination is shown in Table 57 on page 279.

Table 57. TOJUL and TOGREG output date fields

Yaa	TOJUL=Yaa	TOJUL=Yaa(s)	TOGREG=Yaa	TOGREG=Yaa(s)
Y2T	C'yyddd'	C'yysddd'	C'yymmdd'	C'yysmmsdd'
Y2W	C'dddyy'	C'dddsyy'	C'mmddyy'	C'mmsddsyy'
Y2U	P'yyddd'	n/a	n/a	n/a
Y2V	n/a	n/a	P'yymmdd'	n/a
Y2X	P'dddyy'	n/a	n/a	n/a
Y2Y	n/a	n/a	P'mmddyy'	n/a
Y4T	C'ccyyddd'	C'ccyysddd'	C'ccyymmdd'	C'ccyysmmsdd'
Y4W	C'dddccyy'	C'dddsccyy'	C'mmddccyy'	C'mmsddsccyy'
Y4U	P'ccyyddd'	n/a	n/a	n/a
Y4V	n/a	n/a	P'ccyymmdd'	n/a
Y4X	P'dddccyy'	n/a	n/a	n/a
Y4Y	n/a	n/a	P'mmddccyy'	n/a

Sample Syntax:

```
* Convert a P'dddyy' input date to a C'ccyy/mm/dd' output date
  OUTFIL BUILD=(21,3,Y2X,TOGREG=Y4T(/),X,
* Convert a C'ccyymmdd' input date to a P'ccyyddd' output date
     42,8,Y4T,TOJUL=Y4U,X,
* Convert a C'mmddyy' input date to a C'yymmdd' output date
     11,6,Y2W,TOGREG=Y2T)
```

p,m,Yxx,WEEKDAY={CHAR3|CHAR9|DIGIT1}

Converts an input date field to a corresponding output day of the week in one of several forms.

Each type of input date field you can use is shown in Table 56 on page 278. The different types of output you can display are shown in Table 58 on page 279.

Table 58. Output for weekdays

Day	CHAR3	CHAR9	DIGIT1
Sunday	C'SUN'	C'SUNDAY '	C'1'
Monday	C'MON'	C'MONDAY '	C'2'
Tuesday	C'TUE'	C'TUESDAY '	C'3'
Wednesday	C'WED'	C'WEDNESDAY'	C'4'
Thursday	C'THU'	C'THURSDAY '	C'5'
Friday	C'FRI'	C'FRIDAY '	C'6'
Saturday	C'SAT'	C'SATURDAY '	C'7'

Sample Syntax:

```
* Convert a P'mmddccyy' input date to a 3-character weekday
  OUTFIL BUILD=(5:15,5,Y4Y,WEEKDAY=CHAR3,
* Convert a C'yyddd' input date to a 1-digit weekday
     18:27,5,Y2T,WEEKDAY=DIGIT1,
* Convert a P'dddccyy' input date to a 9-character weekday
     41:121,4,Y4X,WEEKDAY=CHAR9)
```

p,m,Yxx,DT=(abcd) or p,m,Yxx,DTNS=(abc)

Converts an input date field of one type to a corresponding Gregorian output date field of another type.

Each type of input date field you can use is shown in Table 56 on page 278.

DT=(abcd) creates an output date in the form C'adbdc', where a, b, and c indicate the order in which the month, day, and year are to appear and whether the year is to appear as two or four digits, and d is the character to be used to separate the month, day and year. For a, b, and c, use M to represent the month (01-12), D to represent the day (01-31), Y to represent the

last two digits of the year (for example, 09), or 4 to represent the four digits of the year (for example, 2009). M, D, and Y or 4 can each be specified only once.

DTNS=(abc) creates an output date in the form C'abc', where a, b and c indicate the order in which the month, day, and year are to appear and whether the year is to appear as two or four digits. For a, b and c, use M to represent the month (01-12), D to represent the day (01-31), Y to represent the last two digits of the year (for example, 09), or 4 to represent the four digits of the year (for example, 2009). M, D, and Y or 4 can each be specified only once.

Sample Syntax:

```
* Convert a C'yyddd' input date to a C'dd/mm/ccyy' output date
  OUTFIL BUILD=(92,5,Y2T,DT=(DM4/),X,
* Convert a P'ccyyddd' input date to a C'mmddyy' output date
     53:32,4,Y4U,DTNS=(MDY))
```

p,m,Yxx,WEEKNUM={USA|ISO}

Converts an input date field to a corresponding 2 byte output number of the week in 2 different forms.

USA: The WEEKNUM=USA function returns an integer in the range of 1 to 54 that represents the week of the year. The week starts with Sunday, and January 1 is always in the first week.

ISO: The WEEKNUM=ISO function returns an integer in the range of 1 to 53 that represents the week of the year. The week starts with Monday and includes 7 days. Week 1 is the first week of the year to contain a Thursday, which is equivalent to the first week containing January 4. Thus, it is possible to have up to 3 days at the beginning of the year appear as the last week of the previous year, or to have up to 3 days at the end of a year appear as the first week of the next year.

Each type of input date field you can use is shown in Table 56 on page 278. The different types of output you can display are shown in Table 59 on page 280.

Table 59. Output for weeknum

Input - Date	WEEKNUM=USA	WEEKNUM=ISO
2000-01-01	01	52
2000-01-02	02	52
2000-01-03	02	01
2000-12-31	54	52
2014-01-02	01	01
2014-06-09	24	24
2014-12-31	53	01

Sample Syntax:

```
* Convert a P'mmddccyy' input date to a 2-character weeknum in
* ISO format
  OUTFIL BUILD=(5:15,5,Y4Y,WEEKNUM=ISO,
* Convert a C'yyddd' input date to a 2-digit weeknum in
* USA format
     18:27,5,Y2T,WEEKNUM=USA)
```

p,m,Yxx,AGE={YMD|YM|YD}

Can be used to calculate the date duration that specifies the number of years, months, and days between a given date and the current date. The Input date cannot be greater than the current date.

Age=YMD produces a 8 byte result which has duration in years (0-9999), months (00-12), and days (00-31).

Age=YM produces a 6 byte result which has duration in years (0-9999), months (00-12).

Age=YD produces a 7 byte result which has duration in years (0-9999), days (00-366).

For example, if the input birth date is March 12th 2015 and the current date is September 30th 2015 then Age=YMD will produce a duration of 00000618 (0000 years, 06 months, and 18 days).

Sample Syntax: If the input birth date is March 12th 2015 and the current date is September 30th 2015 then Age=YMD will produce a duration of 00000618 (0000 years, 06 months, and 18 days).

```
* Calculate the date duration of a C'ccyyddd' input date to
* current date as years, months and days
  OUTFIL BUILD=(15:1,7,Y4T,AGE=YMD,
* Calculate the date duration of a P'yymmdd' input date to
* current date as years and days
                                55:27,4,Y2V,AGE=YD)
```

Conversion of Real Dates, Special Indicators and Invalid Dates

For CH/ZD dates (Y2T, Y4T, Y2W, Y4W), the zone and sign are ignored; only the digits are used. For example, X'F2F0F0F9F1F2F3D0' and X'A2B0C0D9E1F21320' are treated as 20091230. For PD dates (Y2U, Y4U, Y2V, Y4V, Y2X, Y4X, Y2Y, Y4Y), the sign is ignored; only the relevant digits are used. For example, X'120091230D' is treated as 20091230.

For CH/ZD dates (Y2T, Y4T, Y2W, Y4W), the special indicators are X'00...00' (BI zeros), X'40...40' (blanks), C'0...0' (CH zeros), Z'0...0' (ZD zeros), C'9...9' (CH nines), Z'9...9' (ZD nines) and X'FF...FF' (BI ones). For PD dates (Y2U, Y4U, Y2V, Y4V, Y2X, Y4X, Y2Y, Y4Y), the special indicators are P'0...0' (PD zeros) and P'9...9' (PD nines).

yy for real 2-digit year dates is transformed to ccyy when appropriate using the century window established by the Y2PAST option in effect. ccyy for real 4-digit year dates is transformed to yy when appropriate by removing cc.

Date conversion is not performed for special indicators; the special indicator is just used appropriately for the output date field. For example, if p,5,Y2T,TOGREG=Y4T(/) is used, an input yyddd special indicator of C'99999' results in an output date field of C'9999/99/99'. However, CH/ZD special indicators of BI zeros, blanks and BI ones cannot be converted to PD special indicators.

Conversion involving an input date with an invalid digit (A-F) will result in a data exception (0C7 ABEND) or an incorrect output value.

Conversion involving an invalid input date or invalid output date will result in an output value of asterisks and an informational message (issued once). A date is considered invalid if any of the following range conditions are not met:

- yy must be between 00 and 99
- ccyy must be between 0001 and 9999
- mm must be between 01 and 12
- dd must be between 01 and 31, and must be valid for the year and month
- ddd must be 001 to 366 for a leap year, or between 001 and 365 for a non-leap year

A date is also considered invalid if the input field is a CH/ZD special indicator of binary zeros, blanks or binary ones, and the output field is PD.

%nn,Y2x,todate or %nn,Y4x,todate

Specifies that an input date field of one type is to be converted to a corresponding output date field of another type. See PARSE for details of parsed fields. See "p,m,Y2x,todate or p,m,Y4x,todate" for further details.

p,m,Y2x,dateop or p,m,Y4x,dateop

Specifies an arithmetic operation for an input date field. The available operations are as follows:

- **ADDDAYS**, **ADDMONS** or **ADDYEARS** can be used to add days, months or years to a date field. As a simple example, you could use the following to add 10 days to a C'ccyymmdd' date and produce the resulting C'ccyy/mm/dd' date:

```
15,8,Y4T,ADDDAYS,+10,TOGREG=Y4T(/)
```

- **SUBDAYS**, **SUBMONS** or **SUBYEARS** can be used to subtract days, months or years from a date field. As a simple example, you could use the following to subtract 3 months from a P'dddyy' date and produce the resulting P'ccyyddd' date:

```
21,3,Y2X,SUBMONS,+3,TOJUL=Y4U
```

- **DATEDIFF** can be used to calculate the number of days between two date fields. As a simple example, you could use the following to calculate the difference in days between two C'ccyymmdd' dates:

```
41,8,Y4T,DATEDIFF,31,8,Y4T
```

- **NEXTDday** can be used to calculate the next specified day for a date field. As a simple example, you could use the following to calculate the next Friday for a C'ccyyddd' date as a C'ccyy.ddd' date:

```
3,7,Y4T,NEXTDFRI,TOJUL=Y4T(.)
```

- **PREVDday** can be used to calculate the previous specified day for a date field. As a simple example, you could use the following to calculate the previous Wednesday for a P'yyddd' date as a C'ccyymmdd' date:

```
51,3,Y2U,PREVDWED,TOGREG=Y4T
```

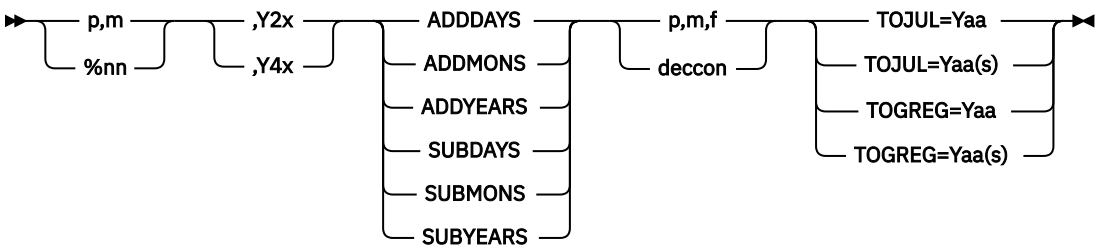
- **LASTDAYW**, **LASTDAYM**, **LASTDAYQ** or **LASTDAYY** can be used to calculate the last day of the week, month, quarter or year for a date field. As a simple example, you could use the following to calculate the last day of the month for a C'mmddccyy' date as a C'mmddccyy' date:

```
28,8,Y4W,LASTDAYM,TOGREG=Y4W
```

%nn,Y2x,dateop or %nn,Y4x,dateop

Specifies an arithmetic operation for a parsed input date field.

Adding or Subtracting Days, Months or Years



p,m,Yxx,keyword,numeric,todate

Can be used to add n days, months or years to an input date field, or subtract n days, months or years from an input date field. The valid keywords are ADDDAYS, ADDMONS, ADDYEARS, SUBDAYS, SUBMONS and SUBYEARS.

The numeric value specifies the number of days, months or years to be added to or subtracted from the input date field. The resulting output date field is converted to the form indicated by todate.

p,m,Yxx specifies the starting position (p), length (m) and format (Yxx) of a 2-digit or 4-digit year input date field. The valid length and format for each type of date field you can use is shown in Table 60 on page 283.

Table 60. p,m,Yxx fields for date arithmetic

m,Yxx	Type of date
5,Y2T	C'yyddd' or Z'yyddd'
6,Y2T	C'yymmdd' or Z'yymmdd'
7,Y4T	C'ccyyddd' or Z'ccyyddd'
8,Y4T	C'ccyymmdd' or Z'ccyymmdd'
5,Y2W	C'dddyy' or Z'dddyy'
6,Y2W	C'mmddyy' or Z'mmddyy'
7,Y4W	C'dddccyy' or Z'dddccyy'
8,Y4W	C'mmddccyy' or Z'mmddccyy'
3,Y2U	P'yyddd'
4,Y2V	P'yymmdd'
4,Y4U	P'ccyyddd'
5,Y4V	P'ccyymmdd'
3,Y2X	P'dddyy'
4,Y2Y	P'mmddyy'
4,Y4X	P'dddccyy'
5,Y4Y	P'mmddccyy'

ADDDAYS adds n days to the p,m,Yxx date field.

ADDMONS adds n months to the p,m,Yxx date field. If necessary, the resulting date is adjusted backwards to the last valid day of the month. For example, if 1 month is added to 20101031, the resulting date will be 20101130 rather than 20101131.

ADDYEARS adds n years to the p,m,Yxx date field. If the resulting date is February 29 of a non-leap year, it will be adjusted backwards to a valid date of February 28.

SUBDAYS subtracts n days from the p,m,Yxx date field.

SUBMONS subtracts n months from the p,m,Yxx date field. If necessary, the resulting date is adjusted backwards to the last valid day of the month. For example, if 1 month is subtracted from 20101031, the resulting date will be 20100930 rather than 20100931.

SUBYEARS subtracts n years from the p,m,Yxx date field. If the resulting date is February 29 of a non-leap year, it will be adjusted backwards to a valid date of February 28.

numeric can be a decimal constant (+n or -n), or a valid numeric field (p,m,f) with BI, FI, ZD, PD, FS, UFF or SFF format and an appropriately corresponding length.

For ADDDAYS or SUBDAYS, the numeric constant or field value can be from -3652058 to +3652058.

For ADDMONS or SUBMONS, the numeric constant or field value can be from -119987 to +119987.

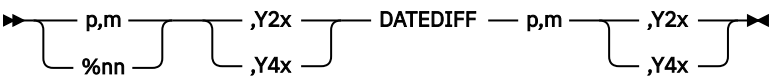
For ADDYEARS or SUBYEARS, the numeric constant or field value can be from -9998 to +9998.

TOJUL and TOGREG
See TOJUL and TOGREG under "p,m,Y2x,todate or p,m,Y4x,todate" for details.

%nn,Yxx,keyword,numeric,todate
Can be used to add n days, months or years to a parsed input date field, or subtract n days, months or years from a parsed input date field. See PARSE for details of parsed fields. See "p,m,Yxx,keyword,numeric,todate" for further details.

%nn cannot be used for the numeric field.

Calculating Difference in Days



p,m,Yxx,DATEDIFF,p,m,Yxx
Can be used to calculate the number of days difference between two input date fields. The result is an 8-byte value consisting of a sign and 7 digits (sddddddd). If the first date is greater than or equal to the second date, the sign is + (plus). If the first date is less than the second date, the sign is - (minus).

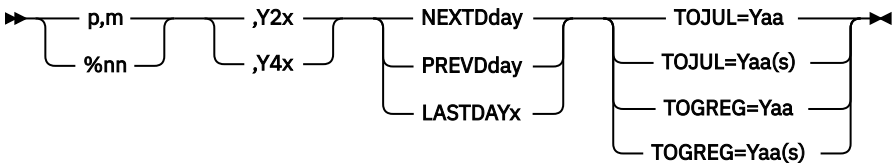
p,m,Yxx specifies the starting position (p), length (m) and format (Yxx) of a 2-digit or 4-digit year input date field. The valid length and format for each type of date field you can use is shown in Table 60 on page 283.

DATEDIFF calculates the number of days that result when the second date is subtracted from the first date. The result is in the form sddddddd with + or - for s as appropriate and leading zeros. The result can be between -3652058 and +3652058 days.

%nn,Yxx,DATEDIFF,p,m,Yxx
Can be used to calculate the number of days difference between a parsed input date field and another input date field. See PARSE for details of parsed fields. See "p,m,Yxx,DATEDIFF,p,m,Yxx" for further details.

%nn cannot be used for the second date field.

Next, Previous or Last Day



p,m,Yxx,keyword,todate
Can be used to specify the next specified day, previous specified day, or last day of the week, month, quarter or year for an input date. The valid keywords are NEXTDday, PREVDday or LASTDAYx (where day can be SUN, MON, TUE, WED, THU, FRI or SAT and x can be W, M, Q or Y).

The result is an output date field which is converted to the form indicated by todate.

p,m,Yxx specifies the starting position (p), length (m) and format (Yxx) of a 2-digit or 4-digit year input date field. The valid length and format for each type of date field you can use is shown in Table 60 on page 283.

NEXTDSUN calculates the next Sunday for a date field.

NEXTDMON calculates the next Monday for a date field.

NEXTDTUE calculates the next Tuesday for a date field.

NEXTDWED calculates the next Wednesday for a date field.

NEXTDTHU calculates the next Thursday for a date field.

NEXTDFRI calculates the next Friday for a date field.

NEXTDSAT calculates the next Saturday for a date field.

PREVDSUN calculates the previous Sunday for a date field.

PREVDMON calculates the previous Monday for a date field.

PREVDTUE calculates the previous Tuesday for a date field.

PREVDWED calculates the previous Wednesday for a date field

PREVDTHU calculates the previous Thursday for a date field.

PREVDFRI calculates the previous Friday for a date field.

PREVDSAT calculates the previous Saturday for a date

LASTDAYW calculates the last day of the week (Friday) for a date field.

LASTDAYM calculates the last day of the month for a date field.

LASTDAYQ calculates the last day of the quarter for a date field.

LASTDAYY calculates the last day of the year for a date field.

TOJUL and TOGREG
See TOJUL and TOGREG under "p,m,Y2x,todate or p,m,Y4x,todate" for details.

%nn,Yxx,keyword,todate
Can be used to specify the next specified day, previous specified day, or last day of the week, month, quarter or year for a parsed input date. See PARSE for details of parsed fields. See “p,m,Yxx,keyword,todate” for further details.

Arithmetic with Real Dates, Special Indicators and Invalid Dates
For CH/ZD dates (Y2T, Y4T, Y2W, Y4W), the zone and sign are ignored; only the digits are used. For example, X'F2F0F0F9F1F2F3D0' and X'A2B0C0D9E1F21320' are treated as 20091230. For PD dates (Y2U, Y4U, Y2V, Y4V, Y2X, Y4X, Y2Y, Y4Y), the sign is ignored; only the relevant digits are used. For example, X'120091230D' is treated as 20091230.

For CH/ZD dates (Y2T, Y4T, Y2W, Y4W), the special indicators are X'00...00' (BI zeros), X'40...40' (blanks), C'0...0' (CH zeros), Z'0...0' (ZD zeros), C'9...9' (CH nines), Z'9...9' (ZD nines) and X'FF...FF' (BI ones). For PD dates (Y2U, Y4U, Y2V, Y4V, Y2X, Y4X, Y2Y, Y4Y), the special indicators are P'0...0' (PD zeros) and P'9...9' (PD nines).

yy for real 2-digit year dates is transformed to ccyy when appropriate using the century window established by the Y2PAST option in effect. ccyy for real 4-digit year dates is transformed to yy when appropriate by removing cc.

If a special indicator is used for DATEDIFF, the result will be an output value of asterisks and an informational message (issued once).

Date arithmetic is not performed for special indicators; the special indicator is just used appropriately for the output date field. For example, if p,5,Y2T,ADDDAYS,+10,TOGREG=Y4T(/) is used, an input yyddd special indicator of C'99999' results in an output date field of C'9999/99/99'. However, CH/ZD special indicators of BI zeros, blanks and BI ones cannot be converted to PD special indicators.

Arithmetic involving an input date with an invalid digit (A-F) will result in a data exception (0C7 ABEND) or an incorrect output value.

Arithmetic involving an invalid input date or invalid output date will result in an output value of asterisks and an informational message (issued once). A date is considered invalid if any of the following range conditions are not met:

- yy must be between 00 and 99
- ccyy must be between 0001 and 9999
- mm must be between 01 and 12
- dd must be between 01 and 31, and must be valid for the year and month
- ddd must be between 001 and 366 for a leap year, or between 001 and 365 for a non-leap year.

A date is also considered invalid if the input field is a CH/ZD special indicator of binary zeros, blanks or binary ones, and the output field is PD.

p,m,Y2x(s) or p,m,Y4x(s)
Specifies that an input date field is to be edited with separators. s can be any character except a blank. Real Y2x dates are edited using the century window established by the Y2PAST option in effect. Y2x and Y4x dates with special indicators are expanded appropriately (for example, p,8,Y4T(s) transforms C'00000000' to C'0000/00/00').

Editing involving an input date with an invalid digit (A-F) can result in a data exception (0C7 ABEND) or an incorrect output value. Editing involving an invalid input date can result in an invalid output value.

p
See p under p,m,a.

m
Specifies the length in bytes of the Y2x (two-digit year) or Y4x (four-digit year) date field.

Y2x(s) or Y4x(s)
Specifies the Y2 or Y4 format for the date field and the separator character(s). See Appendix C, "Data format descriptions," on page 823 for detailed format descriptions.

Sample Syntax:

```
* Convert a Z'dddccyy' date to a C'ddd/ccyy' date.
  OUTFIL BUILD=(19,7,Y4W(/),X,
* Convert a P'ccyymmdd' date to a C'ccyy-mm-dd' date.
     43,5,Y4V(-))
```

Table 61 on page 286 shows the output produced for each type of edited date.

Table 61. Input and result fields for Yxx(s) date editing

m,Yxx	Input date	Output for p,m,Yxx(s)
3,Y2T(s)	C'yyx' or Z'yyx'	C'ccyysx'
4,Y2T(s)	C'yyxx' or Z'yyxx'	C'ccyysxx'
5,Y2T(s)	C'yyddd' or Z'yyddd'	C'ccyysddd'
6,Y2T(s)	C'yymmdd' or Z'yymmdd'	C'ccyysmmsdd'
7,Y4T(s)	C'ccyyddd' or Z'ccyyddd'	C'ccyysddd'
8,Y4T(s)	C'ccyymmdd' or Z'ccyymmdd'	C'ccyysmmsdd'
2,Y2U(s)	P'yyx'	C'ccyysx'
3,Y2V(s)	P'yyxx'	C'ccyysxx'
3,Y2U(s)	P'yyddd'	C'ccyysddd'
4,Y2V(s)	P'yymmdd'	C'ccyysmmsdd'
4,Y4U(s)	P'ccyyddd'	C'ccyysddd'
5,Y4V(s)	P'ccyymmdd'	C'ccyysmmsdd'
3,Y2W(s)	C'xyy' or Z'xyy'	C'xsccyy'
4,Y2W(s)	C'xxyy' or Z'xxyy'	C'xxsccyy'
5,Y2W(s)	C'dddyy' or Z'dddyy'	C'dddsccyy'
6,Y2W(s)	C'mmddyy' or Z'mmddyy'	C'mmsddsccyy'
7,Y4W(s)	C'dddccyy' or Z'dddccyy'	C'dddsccyy'
8,Y4W(s)	C'mmddccyy' or Z'mmddccyy'	C'mmsddsccyy'
2,Y2X(s)	P'xyy'	C'xsccyy'
3,Y2Y(s)	P'xxyy'	C'xxsccyy'
3,Y2X(s)	P'dddyy'	C'dddsccyy'
4,Y2Y(s)	P'mmddyy'	C'mmsddsccyy'
4,Y4X(s)	P'dddccyy'	C'dddsccyy'
5,Y4Y(s)	P'mmddccyy'	C'mmsddsccyy'

%nn,Y2x(s) or %nn,Y4x(s)
Specifies that a parsed input date field is to be edited with separators. See PARSE for details of parsed fields. See "p,m,Y2x(s) or p,m,Y4x(s)" for further details.

p,m,Y2xP
Specifies that an input date field is to be converted to a packed decimal output date field. Real Y2x dates are edited using the century window established by the Y2PAST option in effect. Y2x and Y4x dates with special indicators are expanded appropriately (for example, p,6,Y2TP transforms C'000000' to P'00000000').

Editing involving an input date with an invalid digit (A-F) can result in a data exception (0C7 ABEND) or an incorrect output value. Editing involving an invalid input date can result in an invalid output value.

p
See p under p,m,a.

m
Specifies the length in bytes of the Y2x (two-digit year) or Y4x (four-digit year) date field.

Y2xP
Specifies the Y2 format for the date field. See Appendix C, “Data format descriptions,” on page 823 for detailed format descriptions.

Sample Syntax:

```
OUTFIL BUILD=(11,3,Y2XP,X,21,4,Y2WP)
```

Table 62 on page 287 shows the output produced for each type of date.

Table 62. Input and result fields for Yxx(s) date editing

m,Y2xP	Input date	Output for p,m,Y2xP
3,Y2TP	C'yyx' or Z'yyx'	P'ccyyx'
4,Y2TP	C'yyxx' or Z'yyxx'	P'ccyyxx'
5,Y2TP	C'yyddd' or Z'yyddd'	P'ccyyddd'
6,Y2TP	C'yymmdd' or Z'yymmdd'	P'ccyymmdd'
2,Y2UP	P'yyx'	P'ccyyx'
3,Y2VP	P'yyxx'	P'ccyyxx'
3,Y2UP	P'yyddd'	C'ccyyddd'
4,Y2VP	P'yymmdd'	C'ccyymmdd'
3,Y2WP	C'xyy' or Z'xyy'	C'xccyy'
4,Y2WP	C'xxyy' or Z'xxyy'	C'xxccyy'
5,Y2WP	C'dddyy' or Z'dddyy'	C'dddccyy'
6,Y2WP	C'mmddyy' or Z'mmddyy'	C'mmddccyy'
2,Y2XP	P'xyy'	C'xccyy'
3,Y2YP	P'xxyy'	C'xxccyy'
3,Y2XP	P'dddyy'	C'dddccyy'
4,Y2YP	P'mmddyy'	C'mmddccyy'
2,Y2PP	P'yy'	C'ccyy'
1,Y2DP	X'yy'	C'ccyy'

%nn,Y2xP
Specifies that a parsed input date field is to be converted to a packed decimal output date field. See PARSE for details of parsed fields. See "p,m,Y2xP" for further details

p,m,lookup or %nn,lookup
specifies that a character constant, hexadecimal constant, input field (p,m), or parsed input field (%nn) from a lookup table is to appear in the reformatted OUTFIL output record. You can use p,m,lookup or %nn,lookup to select a specified set constant (that is, a character or hexadecimal string) or set field (that is, an input field or parsed input field) based on matching an input value against find constants (that is, character, hexadecimal, or bit constants).

p
See p under p,m,a.

m
specifies the length in bytes of the input field to be compared to the find constants. The value for m must be 1 to 64 if character or hexadecimal find constants are used, or 1 if bit find constants are used.

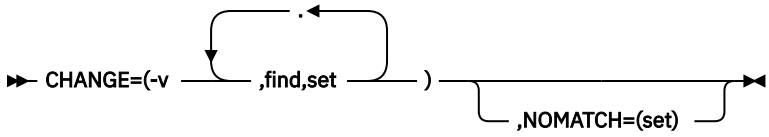
%nn

specifies a parsed input field to be compared to the find constants. See PARSE for details of parsed fields.

lookup

Specifies how the input field or parsed input field is to be changed to the output field, using a lookup table.

CHANGE



specifies a list of change pairs, each consisting of a find constant to be compared to the input field value or parsed input field value and a set constant or set field to use as the output field when a match occurs.

v
specifies the length in bytes of the output field to be inserted in the reformatted OUTFIL output record. The value for v must be between 1 and 64.

find
specifies a find constant to be compared to the input field value or parsed input field value. If the input field value or parsed input field value matches the find constant, the corresponding set constant or set field is used for the output field.

The find constants can be either character string constants, hexadecimal string constants, or bit constants:

- Character string constants (C'xx...x') and hexadecimal string constants (X'yy...yy') can be 1 to m bytes and can be intermixed with each other, but not with bit constants. See “INCLUDE control statement” on page 91 for details of coding character and hexadecimal string constants.

  If the string is less than m bytes, it will be padded on the right to a length of m bytes, using blanks (X'40') for a character string constant or zeros (X'00') for a hexadecimal string constant.
- Bit constants (B'bbbbbbbb') must be 1 byte and cannot be intermixed with character or string constants. See “INCLUDE control statement” on page 91 for details of coding bit constants.

  For bit constants, because of the specification of bits to be ignored, more than one find constant can match an input field value; the set constant for the first match found will be used as the output field. For example, if you specify:

```
OUTFIL OUTREC=(5,1,
  CHANGE=(1,B'11......',C'A',B'1.......',C'B'))
```

  input field value X'C0' (B'11000000') matches both bit constants, but C'A' will be used for the set constant, because its find constant is the first match.

set
specifies a set constant or set field to be used as the output field if the corresponding find constant matches the input field value or parsed input field value. Set constants and set fields can be intermixed.

Set constants can be either character string constants (C'xx...x') or hexadecimal string constants (X'yy...yy') of 1 to v bytes. See “INCLUDE control statement” on page 91 for details of coding character and hexadecimal string constants.

If the string is less than v bytes, it will be padded on the right to a length of v bytes, using blanks (X'40') for a character string constant or zeros (X'00') for a hexadecimal string constant.

Set fields are specified as q,n or %nn. q specifies the input position. See p under p,m,a for details of coding q. n specifies the input length of 1 to v bytes. %nn specifies a parsed input field. See PARSE for details of parsed fields. If the set field length is less than v, the input field or parsed input field will be padded on the right to a length of v bytes, using blanks (X'40').

NOMATCH
specifies the action to be taken if an input field value or parsed input field value does not match any of the find constants. If you do not specify NOMATCH, and a match is not found for any input value, DFSORT terminates processing.

If you specify NOMATCH, it must follow CHANGE.

set
specifies a set constant or set field to be used as the output field if a match is not found.

Set constants can be either character string constants (C'xx...x') or hexadecimal string constants (X'yy...yy') of 1 to v bytes and can be intermixed. See "INCLUDE control statement" on page 91 for details of coding character and hexadecimal string constants.

If the string is less than v bytes, it will be padded on the right to a length of v bytes, using blanks (X'40') for a character string constant or zeros (X'00') for a hexadecimal string constant.

Set fields are specified as q,n or %nn. q specifies the input position. See p under p,m,a for details of coding q. n specifies the input length of 1 to v bytes. %nn specifies a parsed input field. See PARSE for details of parsed fields. If the set field length is less than v, the input field or parsed input field will be padded on the right to a length of v bytes, using blanks (X'40').

Sample Syntax:

```
OUTFIL FILES=1,
 OUTREC=(11,1,
         CHANGE=(6,
           C'R',C'READ',
           C'U',C'UPDATE',
           X'FF',C'EMPTY',
           C'X',35,6,
           C'A',C'ALTER'),
         NOMATCH=(11,6),
         4X,
         21,1,
         CHANGE=(10,
           B'.1.....',C'VSAM',
           B'.0.....',C'NON-VSAM'))
```

p,m,justify
specifies that a left-justified or right-justified input field is to appear in the reformatted OUTFIL output record. For a left-justified field, leading blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For a right-justified field, trailing blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed.

Optionally:

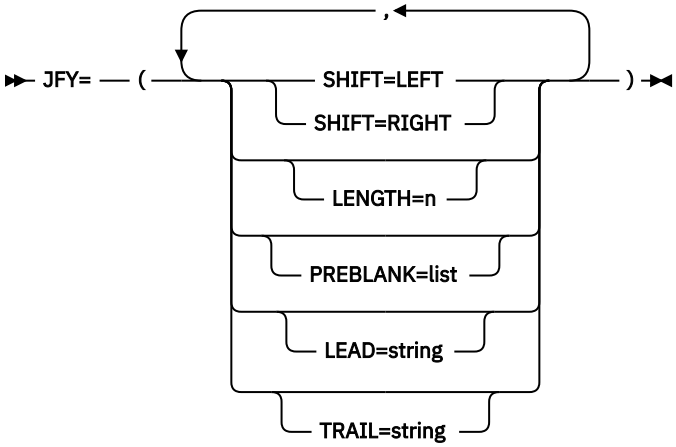
- specific leading and trailing characters can be changed to blanks before justification begins
- a leading string can be inserted
- a trailing string can be inserted
- the output length can be changed (it's equal to the input length by default)

p
See p under p,m,a.

m
See m under p,m,a.

justify
specifies how the input field is to be justified for output.

JFY



Note: For clarity, in the examples in this section, b is used to represent a blank in the input fields.

SHIFT=LEFT
specifies that a left-justified input field is to appear in the reformatted OUTFIL output record. Leading blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For example, with:

```
1,15,JFY=(SHIFT=LEFT)
```

an input field of:

```
bbbbABbCDbbEFbb
```

results in an output field of:

```
ABbCDbbEFbbbbbb
```

m is used for the output field length unless LENGTH=n is specified. If the left-justified input field is shorter than the output field length, blanks are inserted on the right. If the left-justified input field is longer than the output field length, characters are truncated on the right. You can use LENGTH=n to prevent padding or truncation.

SHIFT=RIGHT
specifies that a right-justified input field is to appear in the reformatted OUTFIL output record. Trailing blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed. For example, with:

```
1,15,JFY=(SHIFT=RIGHT)
```

an input field of:

```
bbbbABbCDbbEFbb
```

results in an output field of:

```
bbbbbbABbCDbbEF
```

m is used for the output field length unless LENGTH=n is specified. If the right-justified input field is shorter than the output field length, blanks are inserted on the left. If the right-justified input field is longer than the output field length, characters are truncated on the left. You can use LENGTH=n to prevent padding or truncation.

LENGTH=n
specifies the length of the justified output field. The value for n must be between 1 and 32752. If LENGTH=n is not specified, m (the input field length) is used for the length of the justified output field. You can use LENGTH=n to prevent padding or truncation.

PREBLANK=list
specifies a list of one or more characters to be changed to blanks **before** justify processing begins. Only leading and trailing characters are changed to blanks. Scanning from left to right, each nonblank character at the start of the input field that matches a character in the list is replaced by a blank character, until a nonblank character not in the list is found. Scanning from right to left, each nonblank character at the end of the input field that matches a character in the list will be replaced by a blank character, until a nonblank character not in the list is found.

list can be 1 to 10 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE control statement" on page 91 for details of coding character and hexadecimal string constants.

Each character in the list is treated as one character to be preblanked. Thus PREBLANK=C'*' specifies an asterisk is to be preblanked, and PREBLANK=C',;$' specifies a comma, a semicolon and a dollar sign are each to be preblanked.

For example, let's say we have an input field of:

```
**b<*ABbCD*bEF>*b**
```

If we specify:

```
1,19,JFY=(SHIFT=LEFT,PREBLANK=C'*')
```

each leading or trailing asterisk is changed to a blank before left- justify processing begins. So the output field is:

```
<*ABbCD*bEF>bbbbbbb
```

If we specify:

```
1,19,JFY=(SHIFT=RIGHT,PREBLANK=C'*<>')
```

each leading or trailing asterisk, less than sign and greater than sign is changed to a blank before right-justify processing begins. So the output field is:

```
bbbbbbbbbbABbCD*bEF
```

LEAD=string
specifies a string to be inserted in the output field before the first nonblank character in the input field.

string can be 1 to 50 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE control statement" on page 91 for details of coding character and hexadecimal string constants.

For example, let's say we have an input field of:

```
bABCbEbbbbb
```

If we specify:

```
1,11,JFY=(SHIFT=RIGHT,LEAD=C'XYZ')
```

the output field is:

```
bbbXYZABCbE
```

When we add characters with LEAD=string, it's often necessary to specify LENGTH=n to avoid truncation. For example, let's say we have an input field of:

```
AbBbC
```

If we specify:

```
1,5,JFY=(SHIFT=LEFT,LEAD=C'XYZ')
```

the output field is:

```
XYZAb
```

Since the output field length is defaulted to the input field length of 5, the resulting 8 characters (XYZAbBbC) are truncated on the right to 5 characters (XYZAb) for output. If we instead specify:

```
1,5,JFY=(SHIFT=LEFT,LEAD=C'XYZ',LENGTH=8)
```

the output field is:

```
XYZAbBbC
```

LENGTH=8 increases the output field by the 3 LEAD characters and truncation is prevented.

TRAIL=string
specifies a string to be inserted in the output field after the last nonblank character in the input field.

string can be 1 to 50 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See “INCLUDE control statement” on page 91 for details of coding character and hexadecimal string constants.

For example, let's say we have an input field of:

```
bABCbEbbbbb
```

If we specify:

```
1,11,JFY=(SHIFT=LEFT,TRAIL=C'XYZ')
```

the output field is:

```
ABCbEXYZbbb
```

When we add characters with TRAIL=string, it's often necessary to specify LENGTH=n to avoid truncation. For example, let's say we have an input field of:

```
AbBbC
```

If we specify:

```
1,5,JFY=(SHIFT=RIGHT,TRAIL=C'XYZ')
```

the output field is:

```
bCXYZ
```

Since the output field length is defaulted to the input field length of 5, the resulting 8 characters (AbBbCXYZ) are truncated on the left to 5 characters (bCXYZ) for output. If we instead specify:

```
1,5,JFY=(SHIFT=RIGHT,TRAIL=C'XYZ',LENGTH=8)
```

the output field is:

```
AbBbCXYZ
```

LENGTH=8 increases the output field by the 3 TRAIL characters and truncation is prevented.

Sample Syntax:

```
OUTFIL BUILD=(5:16,20,JFY=(SHIFT=LEFT,PREBLANK=C'*',
   LEAD=C'<A>',TRAIL=C'</A>',LENGTH=22))
```

%nn,justify
specifies that a left-justified or right-justified parsed input field is to appear in the reformatted OUTFIL output record. See PARSE for details of parsed fields. See p,m,justify for further details.

p,m,squeeze
specifies that a left-squeezed or right-squeezed input field is to appear in the reformatted OUTFIL output record. For a left-squeezed field, all blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For a right-squeezed field, all blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed. Optionally:

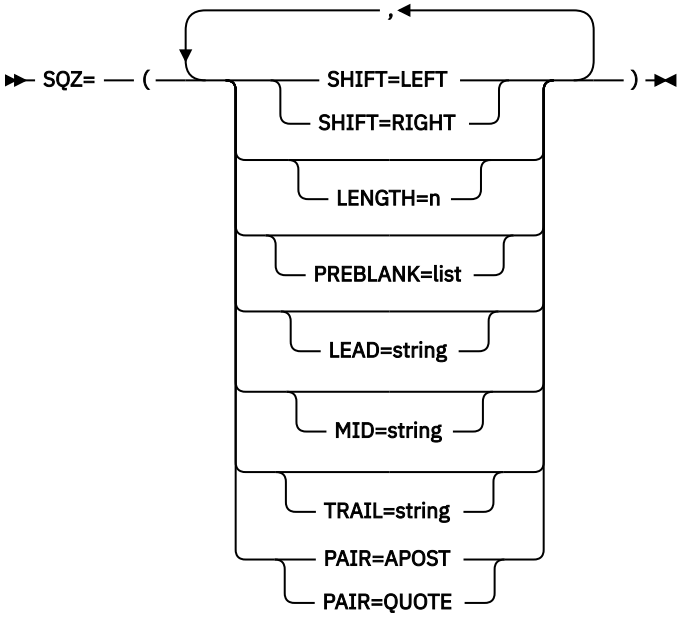
- specific characters can be changed to blanks before squeezing begins
- a leading string can be inserted
- a trailing string can be inserted
- a string (for example, a comma delimiter) can be inserted wherever a group of blanks is removed between the first nonblank and the last nonblank
- blanks can be kept as is between paired apostrophes ('AB CD EF') or paired quotes ("AB CD EF")
- the output length can be changed (it is equal to the input length by default)

p
See p under p,m,a.

m
See m under p,m,a.

squeeze
Specifies how the input field is to be squeezed for output.

SQZ



Note: For clarity, in the examples in this section, b is used to represent a blank in the input fields.

SHIFT=LEFT
specifies that a left-squeezed input field is to appear in the reformatted OUTFIL output record. All blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For example, with:

```
1,15,SQZ=(SHIFT=LEFT)
```

an input field of:

```
bbbbABbCDbbEFbb
```

results in an output field of:

```
ABCDEFbbbbbbbbb
```

m is used for the output field length unless LENGTH=n is specified. If the left-squeezed input field is shorter than the output field length, blanks are inserted on the right. If the left-squeezed input field is longer than the output field length, characters are truncated on the right.

SHIFT=RIGHT
specifies that a right-squeezed input field is to appear in the reformatted OUTFIL output record. All blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed. For example, with:

```
1,15,SQZ=(SHIFT=RIGHT)
```

an input field of:

```
bbbbABbCDbbEFbb
```

results in an output field of:

```
bbbbbbbbbABCDEF
```

m is used for the output field length unless LENGTH=n is specified. If the right-squeezed input field is shorter than the output field length, blanks are inserted on the left. If the right-squeezed input field is longer than the output field length, characters are truncated on the left.

LENGTH=n
specifies the length of the squeezed output field. The value for n must be between 1 and 32752. If LENGTH=n is not specified, m (the input field length) is used for the length of the squeezed output field.

PREBLANK=list
specifies a list of one or more characters to be changed to blanks **before** squeeze processing begins. Each nonblank character in the input field that matches a character in the list is replaced by a blank character.

list can be 1 to 10 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See “INCLUDE control statement” on page 91 for details of coding character and hexadecimal string constants.

Each character in the list is treated as one character to be preblanked. Thus PREBLANK=C'*' specifies an asterisk is to be preblanked, and PREBLANK=C',;$' specifies a comma, a semicolon and a dollar sign are each to be preblanked.

For example, let's say we have an input field of:

```
**b<*ABbC<>D*bEF>*b**
```

If we specify:

```
1,21,SQZ=(SHIFT=LEFT,PREBLANK=C'*',LENGTH=12)
```

each asterisk is changed to a blank before left-squeeze processing begins. The output field is:

```
<ABC<>DEF>bb
```

If we specify:

```
1,21,SQZ=(SHIFT=RIGHT,PREBLANK=C'*<>')
```

each asterisk, less than sign and greater than sign is changed to a blank before right-squeeze processing begins. The output field is:

```
bbbbbbbbbbbbbbbbABCDEF
```

LEAD=string
specifies a string to be inserted in the output field before the first nonblank character in the input field.

string can be 1 to 50 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See “INCLUDE control statement” on page 91 for details of coding character and hexadecimal string constants.

For example, let's say we have an input field of:

```
bABCbEbbbbb
```

If we specify:

```
1,11,SQZ=(SHIFT=RIGHT,LEAD=C'XYZ')
```

the output field is:

```
bbbbXYZABCE
```

When we add characters with LEAD=string, it's often necessary to specify LENGTH=n to avoid truncation. For additional information on this, see "LEAD=string" for JFY shown previously in this section.

MID=string
specifies a string to be inserted in the output field wherever one or more blanks is removed between the first nonblank and the last nonblank.

string can be 1 to 10 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See “INCLUDE control statement” on page 91 for details of coding character and hexadecimal string constants.

For example, let's say we have an input field of:

```
bABbbCbbbDbE
```

If we specify:

```
1,12,SQZ=(SHIFT=LEFT,MID=C',')
```

the output field is:

```
AB,C,D,Ebbbb
```

When we add characters with MID=string, it's often necessary to specify LENGTH=n to avoid truncation. For example, let's say we have an input field of:

```
AbBbC
```

If we specify:

```
1,5,SQZ=(SHIFT=LEFT,MID=C'XY')
```

the output field is:

```
AXYBX
```

Since the output field length is defaulted to the input field length of 5, the resulting 7 characters (AXYBXYC) are truncated on the right to 5 characters (AXYBX) for output. If we instead specify:

```
1,5,SQZ=(SHIFT=LEFT,MID=C'XY',LENGTH=7)
```

the output field is:

```
AXYBXYC
```

LENGTH=7 increases the output field to accommodate the MID strings and truncation is prevented.

TRAIL=string
specifies a string to be inserted in the output field after the last nonblank character in the input field.

string can be 1 to 50 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See “INCLUDE control statement” on page 91 for details of coding character and hexadecimal string constants.

For example, let's say we have an input field of:

```
bABCbEbbbbb
```

If we specify:

```
1,11,SQZ=(SHIFT=LEFT,LEAD=X'7D',TRAIL=X'7D')
```

the output field is:

```
'ABCE'bbbbb
```

When we add characters with TRAIL=string, it's often necessary to specify LENGTH=n to avoid truncation. For additional information on this, see "TRAIL=string" for JFY as shown previously in this section.

PAIR=APOST
specifies that blanks and PREBLANK characters between apostrophe (') pairs are to be kept as is. Use PAIR=APOST when you have literals that should not be squeezed.

For example, let's say we have an input field of:

```
b*b'ABbb*''*C'bbb'D*Ebb'*
```

If we specify:

```
1,25,SQZ=(SHIFT=LEFT,PREBLANK=C'*',MID=C',')
```

all of the blanks and asterisks, including those inside the apostrophe pairs, are squeezed out. The output field is:

```
'AB,'',C','D,E,'bbbbbbbbb
```

However, if we specify:

```
 1,25,SQZ=(SHIFT=LEFT,PREBLANK=C'*',MID=C',',
           PAIR=APOST)
```

only the blanks and asterisks outside of the apostrophe pairs are squeezed out. The output field is:

```
'ABbb*''*C','D*Ebb'bbbbbb
```

If an apostrophe is specified in the PREBLANK list (for example, PREBLANK=C'''' or PREBLANK=X'7D'), it is ignored, that is, an apostrophe in the input field is not replaced by a blank. So if you want each apostrophe to be replaced by a blank, do not specify PAIR=APOST.

For SHIFT=LEFT, scanning is left to right. If an apostrophe is found with no subsequent apostrophe, all of the characters from the apostrophe to the end of the input field are ignored. For example, let's say we have an input field of:

```
bbXbY'ABCbbbDbb
```

If we specify:

```
1,15,SQZ=(SHIFT=LEFT,PAIR=APOST)
```

there's an unpaired apostrophe, so all of the characters from the apostrophe to the end of the input field are ignored. The output field is:

```
XY'ABCbbbDbbbbb
```

For SHIFT=RIGHT, scanning is right to left. If an apostrophe is found with no previous apostrophe, all of the characters from the apostrophe to the beginning of the input field are ignored. For example, let's say we have an input field of:

```
bbXbY'ABCbbbDbb
```

If we specify:

```
1,15,SQZ=(SHIFT=RIGHT,PAIR=APOST)
```

there's an unpaired apostrophe, so all of the characters from the apostrophe to the beginning of the input field are ignored. The output field is:

```
bbbbbbbXbY'ABCD
```

PAIR=QUOTE
specifies that blanks and PREBLANK characters between quote (") pairs are to be kept as is. Use PAIR=QUOTE when you have literals that should not be squeezed.

For example, let's say we have an input field of:

```
b*b"ABbb*""*C"bbb"D*Ebb"*
```

If we specify:

```
1,25,SQZ=(SHIFT=LEFT,PREBLANK=C'*',MID=C',')
```

all of the blanks and asterisks, including those inside the quote pairs, are squeezed out. The output field is:

```
"AB,"",C","D,E,"bbbbbbbbb
```

However, if we specify:

```
1,25,SQZ=(SHIFT=LEFT,PREBLANK=C'*',MID=C',',
          PAIR=QUOTE)
```

only the blanks and asterisks outside of the quote pairs are squeezed out. The output field is:

```
"ABbb*""*C","D*Ebb'bbbbbb
```

If a quote is specified in the PREBLANK list (for example, PREBLANK=C'"' or PREBLANK=X'7F'), it is ignored, that is, a quote in the input field is not replaced by a blank. So if you want each quote to be replaced by a blank, do not specify PAIR=QUOTE.

For SHIFT=LEFT, scanning is left to right. If a quote is found with no subsequent quote, all of the characters from the quote to the end of the input field are ignored. For example, let's say we have an input field of:

```
bbXbY"ABCbbbDbb
```

If we specify:

```
1,15,SQZ=(SHIFT=LEFT,PAIR=QUOTE)
```

there's an unpaired quote, so all of the characters from the quote to the end of the input field are ignored. The output field is:

```
XY"ABCbbbDbbbbb
```

For SHIFT=RIGHT, scanning is right to left. If a quote is found with no previous quote, all of the characters from the quote to the beginning of the input field are ignored. For example, let's say we have an input field of:

```
bbXbY"ABCbbbDbb
```

If we specify:

```
1,15,SQZ=(SHIFT=RIGHT,PAIR=QUOTE)
```

there's an unpaired quote, so all of the characters from the quote to the beginning of the input field are ignored. The output field is:

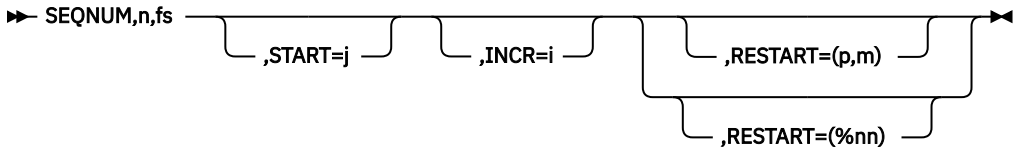
```
bbbbbbbXbY"ABCD
```

Sample Syntax:

```
   OUTFIL BUILD=(5:16,20,SQZ=(SHIFT=LEFT,PAIR=QUOTE,PREBLANK=C'<>'))
```

%nn,squeeze
specifies that a left-squeezed or right-squeezed parsed input field is to appear in the reformatted OUTFIL output record. See PARSE for details of parsed fields. See p,m,squeeze for further details.

seqnum



specifies that a sequence number is to appear in the reformatted OUTFIL output record. The sequence numbers are assigned in the order in which the records are received for OUTFIL OUTREC processing. You can create BI, PD, ZD, CSF, or FS sequence numbers and control their lengths, starting values and increment values. You can restart the sequence number at the start value each time a specified OUTFIL input field (p,m) or parsed input field (%nn) changes.

n
specifies the length of the sequence number. The value for n must be between 1 and 16.

fs
specifies the format for the sequence number, which can be BI, PD, ZD, CSF, or FS.

For a ZD format sequence number, F is used as the sign.

For a PD format sequence number, C is used as the sign.

For a CSF or FS format sequence number, blank is used as the sign and leading zeros are suppressed.

For a PD, ZD, CSF, or FS format sequence number, the maximum value DFSORT can create is limited to the lesser of 15 decimal digits or the output field length (n). If a sequence number overflows this limit, it will be truncated to the lesser of 15 decimal digits or the output field length, and then subsequently incremented as usual.

For a BI format sequence number, the maximum value DFSORT can create is limited to the lesser of 8 bytes of ones (X'FFFFFFFFFFFFFFFF') or the number of ones that will fit in the specified output field length (n). If a sequence number overflows this limit, it will be truncated to the lesser of 8 bytes or the output field length, and then subsequently incremented as usual.

START
specifies the starting value for the sequence number.

j
specifies the starting value. The value for j must be between 0 and 100000000000. The default for j is 1.

INCR
specifies the increment value for the sequence number.

i
specifies the increment value. The value for i must be between 1 and 10000000. The default for i is 1.

RESTART
specifies that DFSORT is to restart the sequence number at the starting value (START=j) when the binary value for the specified OUTFIL input field or parsed input field changes. This allows you to sequence each set of records with the same value (that is, duplicate records) separately. For example: Without RESTART, if you had six OUTFIL input records with 'A', 'A', 'A', 'B', 'B' and 'C', respectively, in position 1, the output records would be given the sequence numbers 1, 2, 3, 4, 5 and 6. But with RESTART=(1,1), the output records are given the sequence numbers 1, 2, 3, 1, 2 and 1; DFSORT restarts at the starting value (1, by default) when the input value at position 1 changes from 'A' to 'B' and again when it changes from 'B' to 'C'.

p,m or %nn
specifies the OUTFIL input field or parsed input field to be used to determine when the sequence number is to be restarted at the starting value. If a variable-length OUTFIL input record is too short to contain a specified restart field, binary zeros (or blanks for a parsed field) will be used for the missing bytes, intentionally or unintentionally.

p
See p under p,m,a

m
specifies the length in bytes of the input field. The value for m must be between 1 and 256.

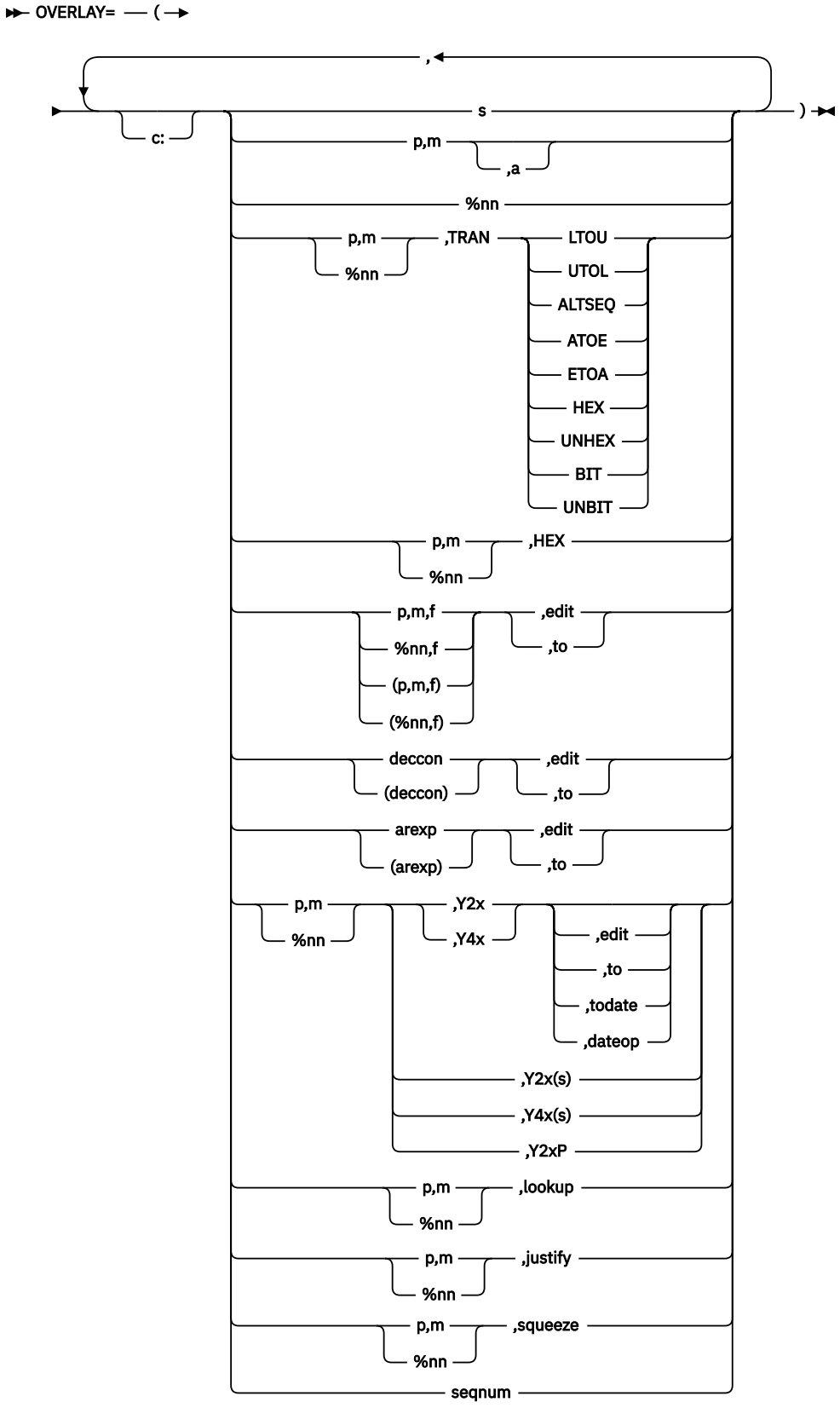
%nn
specifies the parsed input field. See PARSE for details of parsed fields.

Sample Syntax:

```
OUTFIL FNAMES=O1,OUTREC=(SEQNUM,6,ZD,START=1000,INCR=50,
         X,22,8,X,13,5)
OUTFIL FNAMES=O2,OUTREC=(1,12,SEQNUM,4,BI)
OUTFIL FNAMES=O3,OUTREC=(1,80,81:SEQNUM,8,ZD,START=21,INCR=20,
              RESTART=(35,8))
```

Default for BUILD or OUTREC: None; must be specified.

OVERLAY



Specifies each item that is to overlay specific columns in the reformatted record. Columns that are not overlaid remain unchanged. If you want to insert, rearrange, or delete fields, use BUILD or OUTREC rather than OVERLAY. Use OVERLAY only to overlay existing columns or to add fields at the end of

every record. OVERLAY can be easier to use than BUILD or OUTREC when you just want to change a few fields without rebuilding the entire record.

For fixed-length records, the first input and output data byte starts at position 1. For variable-length records, the first input and output data byte starts at position 5, after the RDW in positions 1-4.

Use c: (column) to specify the output positions to be overlaid. If you do not specify c: for the first item, it defaults to 1:. If you do not specify c: for any other item, it starts after the previous item. For example, if you specify:

```
  OUTFIL OVERLAY=(25,2,11:C'A',15,3,C'**')
```

Input positions 25-26 are placed at output positions 1-2; C'A' is placed at output position 11; input positions 15-17 are placed at output positions 12-14; and C'**' is placed at output positions 15-16. The rest of the record remains unchanged.

You can specify items in any order, you can change the same item multiple times and you can overlap output columns. **Changes to earlier items affect changes to later items.** For example, say you specify:

```
OUTFIL OVERLAY=(21:8,4,ZD,MUL,+10,TO=ZD,LENGTH=6,
5:5,1,TRAN=UTOL,
5:5,1,CHANGE=(1,C'a',C'X',C'b',C'Y'),NOMATCH=(5,1))
```

and input position 5 has 'A'. The second item (UTOL) would change 'A' to 'a' and the third item (CHANGE) would change 'a' again to 'X'.

If you specify an OVERLAY item that extends the overlay record beyond the end of the input record, the reformatted record length is automatically increased to that length, and blanks are filled in on the left as needed. For variable-length records, the RDW length is increased to correspond to the larger reformatted record length after all of the OVERLAY items are processed. For example, if your OUTFIL input record has a length of 40 and you specify:

```
OUTFIL OVERLAY=(16:C'ABC',51:5C'*',35:15,2)
```

the OUTFIL output record is given a length of 55. Blanks are filled in from columns 41-50. For variable-length records, the length in the RDW is changed from 40 to 55 after all of the OVERLAY items are processed.

Missing bytes in specified input fields are replaced with blanks so the padded fields can be processed.

You can use the OVERLAY parameter with the FTOV parameter to convert fixed-length record data sets to variable-length record data sets.

You can use the VLTRIM parameter with the OVERLAY parameter to remove specified trailing bytes from the end of variable-length records.

You can use the VLTRAIL parameter with the OVERLAY parameter to insert a string at the end of variable-length records.

You can use the OVERLAY parameter with any or all of the report parameters in the same way as for the BUILD or OUTREC parameter.

The OVERLAY parameter of the OUTREC statement applies to all input records whereas the OVERLAY parameter of the OUTFIL statement only applies to the OUTFIL input records for its OUTFIL group.

See OUTFIL OUTREC for details of the items listed in the OVERLAY syntax diagram shown previously in this section. You can specify all of the items for OVERLAY in the same way that you can specify them for BUILD or OUTREC with the following exceptions:

- You cannot specify p or p,HEX or p,TRAN=keyword for OVERLAY.
- You cannot specify / for OVERLAY.
- For p,m,H or p,m,F or or p,m,D fields specified for OVERLAY, fields are aligned as necessary without changing the preceding bytes.

- For variable-length records, you must not overlay positions 1-4 (the RDW) for OVERLAY, so be sure to specify the first column (c:) as 5 or greater. Do not specify 1:, 2:, 3: or 4: anywhere in your OVERLAY parameter. If you do not specify the first column, it will default to 1: which is invalid for variable-length records with OVERLAY. Whereas OUTREC=(1,m,...) is required, OVERLAY=(1,m) is not allowed since it would overlay the RDW.

Sample Syntax:

Fixed input records:

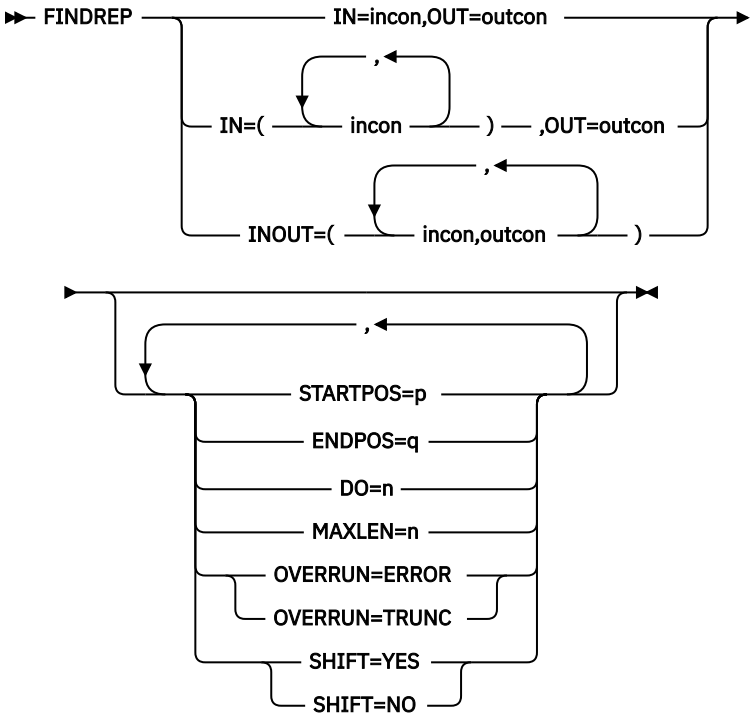
```
OUTFIL FNAMES=FLR,
     OVERLAY=(21:21,4,ZD,TO=PD,LENGTH=4,
         2:5,8,HEX,45:C'*',32,4,C'*',81:SEQNUM,5,ZD)
```

Variable input records:

```
OUTFIL FNAMES=VLR,
    OVERLAY=(5:X'0001',28:C'Date is ',YDDDNS=(4D),
       17:5C'*')
```

Default for OVERLAY: None; must be specified.

FINDREP



You can use FINDREP to find constants anywhere in a record and replace them with other constants of the same or different lengths. You can find character or hexadecimal input constants anywhere in your records and replace them with character, hexadecimal or null output constants. As appropriate, bytes can be shifted left or right, blank padding can be added for fixed-length records, and the length can be changed for variable-length records.

Various options of FINDREP allow you to define one or more input constants and a corresponding output constant, define one or more pairs of input and output constants, start and end the find scan at specified positions, stop after a specified number of constants are replaced, increase or decrease the length of the output record, define the action to be taken if nonblank characters overrun the end of the record, and specify whether output constants are to replace or overlay input constants.

You can use the FINDREP parameter with the FTOV parameter to convert fixed-length record data sets to variable-length record data sets.

You can use the VLTRIM parameter with the FINDREP parameter to remove specified trailing bytes from the end of variable-length records.

You can use the VLTRAIL parameter with the FINDREP parameter to insert a string at the end of variable-length records.

You can use the FINDREP parameter with any or all of the report parameters in the same way as for the BUILD or OUTREC parameter.

The FINDREP parameter of the OUTREC statement applies to all input records whereas the FINDREP parameter of the OUTFIL statement only applies to the OUTFIL input records for its OUTFIL group.

Input and output constants for find and replace processing are defined as follows:

input constant
An input constant can be specified as a single character string, a repeated character string, a single hexadecimal string, or a repeated hexadecimal string. The syntax is: C'string', nC'string', X'string' or nX'string'. n can be 1 to 256. The string can be 1 to 256 characters, or 1 to 256 pairs of hexadecimal digits. The total length of the constant must not exceed 256 bytes. Use two apostrophes for a single apostrophe.

output constant
An output constant can be specified as a null string, a single character string, a repeated character string, a single hexadecimal string, or a repeated hexadecimal string. The syntax is: C'' (null), C'string', nC'string', X'string' or nX'string'. n can be 1 to 256. The string can be a null, or 1 to 256 characters, or 1 to 256 pairs of hexadecimal digits. The total length of the constant must not exceed 256 bytes. Use two apostrophes for a single apostrophe. A null string can be used to remove input constants.

You must specify the input and output constants to be used for find and replace processing in one of the following ways. The default processing for each method described later in this section can be changed with various options described later.

IN=incon,OUT=outcon
Specifies one input constant and one output constant. Position 1 (for fixed-length records) or 5 (for variable-length records) will be set as the current position. The current position of the input record will be checked for the input constant. If a match is not found at the current position, the current position will be incremented by 1, and the process will be repeated. If a match is found at the current position, the output constant will replace the input constant, the current position will be incremented past the input constant, and the process will be repeated. Bytes after the replaced constants up to the end of the record will be shifted left or right as needed. Processing will stop when the current position is beyond the end of the input record.

Example:

```
OUTFIL IFTHEN=(WHEN=(11,1,CH,EQ,C'3'),
         FINDREP=(IN=C'YES',OUT=C'NO'))
```

Replaces every C'YES' input constant in records with a C'3' in position 11 with a C'NO' output constant, and shifts the bytes after each replaced constant to the left.

IN=(incon1,incon2,...,inconx),OUT=outcon
Specifies multiple input constants and one output constant. Position 1 (for fixed-length records) or 5 (for variable-length records) will be set as the current position. The current position of the input record will be checked for each input constant in turn until a match is found or all of the input constants have been checked. If a match is not found at the current position for any input constant, the current position will be incremented by 1, and the process will be repeated. If a match is found at the current position, the output constant will replace the input constant, the current position will be incremented past the input constant, and the process will be repeated. Bytes after the replaced constants up to the end of the record will be shifted left or right as needed. Processing will stop when the current position is beyond the end of the input record.

Example:

```
OUTFIL FINDREP=(IN=(X'FF',3X'00'),OUT=C'')
```

Removes every X'FF' and X'000000' input constant, and shifts the bytes after each removed constant to the left.

INOUT=(incon1,outcon1,incon2,outcon2,...,inconx,outconx)
Specifies one or more pairs of input and output constants. Position 1 (for fixed-length records) or 5 (for variable-length records) will be set as the current position. The current position of the input record will be checked for each input constant in turn until a match is found or all of the input constants have been checked. If a match is not found at the current position for any input constant, the current position will be incremented by 1, and the process will be repeated. If a match is found at the current position for an input constant, the corresponding output constant will replace the input constant, the current position will be incremented past the input constant, and the process will be repeated. Bytes after the replaced constants up to the end of the record will be shifted left or right as needed. Processing will stop when the current position is beyond the end of the input record.

Example:

```
OUTFIL FINDREP=(INOUT=(C'SAT',C'SATURDAY',C'SUN',C'SUNDAY'))
```

Replaces every C'SAT' input constant with a C'SATURDAY' output constant, and every C'SUN' input constant with a C'SUNDAY' output constant, and shifts the bytes after each replaced constant to the right.

You can specify any or all of the following options to change the default processing described earlier:

STARTPOS=p
Specifies the starting position in the input record for the find scan, overriding the default of position 1 for a fixed-length record or position 5 for a variable-length record. Use STARTPOS=p if you want to start your find scan at a particular position. p can be 1 to 32752. If p is less than 5 for a variable-length record, 5 will be used for p. If p is beyond the end of the input record, find and replace processing will not be performed for the record.

Example:

```
OUTFIL FINDREP=(IN=C'Yes',OUT=C'YES',STARTPOS=11)
```

Replaces every C'Yes' input constant found starting at or after position 11 with a C'YES' output constant.

ENDPOS=q
Specifies the ending position in the input record for the find scan, overriding the default of the end of the record. Use ENDPOS=q if you want to end your find scan at a particular position. ENDPOS=q only applies to the end of the find scan; bytes will still be shifted up to the end of the record as needed. q can be 1 to 32752. If q is less than 5 for a variable-length record, 5 will be used for q. If q is beyond the end of the input record, the end of the record will be used for q. If STARTPOS=p and ENDPOS=q are both specified, and p is greater than q, find and replace processing will not be performed for the record.

Example:

```
OUTFIL FINDREP=(IN=(C'D27',C'A52',C'X31'),OUT=C'INVALID',
   ENDPOS=2015)
```

Replaces every C'D27', C'A52' and C'X31' input constant found before or at position 2015 with a C'INVALID' output constant, and shifts the bytes after each replaced constant to the right (past 2015 up to the end of the record).

DO=n

Specifies the maximum number of times find and replace is to be performed for a record, overriding the default of every time. Scanning for the input constant stops when n input constants have been found and replaced. Use DO=n if you want to stop after a particular number of constants have been replaced. n can be 1 to 1000.

Example:

```
OUTFIL FNAMES=OUT1,FINDREP=(IN=X'015C',OUT=X'015D',DO=3)
```

Replaces the first three X'015C' input constants found with X'015D' output constants.

MAXLEN=n

Specifies the maximum length to be used for the output record created by find and replace processing, overriding the default of using the maximum length of the input record. Use MAXLEN=n if you want to increase or decrease the output record length. (For IFTHEN FINDREP, MAXLEN=n can only be used to increase the output record length, not decrease it. See "Notes" later in this section for more information.)

If an output constant is larger than a corresponding input constant, MAXLEN=n can be used to increase the size of the output record to allow for shifting characters to the right. n can be 1 to 32752. MAXLEN=n will be used to set the LRECL of the output data set, when appropriate.

Example:

```
OUTFIL FINDREP=(INOUT=(C'01',C'January',C'02',C'February',
  C'03',C'March'),MAXLEN=150)
```

Replaces every C'01' input constant with a C'January' output constant, every C'02' input constant with a C'February' output constant, and every C'03' input constant with a C'March' output constant, and shifts the bytes after each replaced constant to the right, allowing the record to expand to 150 bytes.

OVERRUN=ERROR or OVERRUN=TRUNC

Specifies the action DFSORT is to take if an overrun occurs, that is:

- if nonblank bytes are shifted to the right past the end of the output record as specified by MAXLEN=n, or as defaulted to the input record length, or
- if MAXLEN=n is used to make the output record smaller than the input record, and nonblank bytes are found in the input record past the end of the output record.

OVERRUN=ERROR is the default; it tells DFSORT to issue an error message and terminate if an overrun occurs.

Use OVERRUN=TRUNC if you want DFSORT to truncate the output record to the MAXLEN=n or input record length if an overrun occurs, rather than terminating. Bytes beyond the end of the output record are lost.

Example:

```
OUTFIL FINDREP=(IN=X'FFFF',OUT=C'INVALID',MAXLEN=50,OVERRUN=TRUNC)
```

Replaces every X'FFFF' input constant with a C'INVALID' output constant, and shifts the bytes after the replaced constants to the right. 50 byte output records are created. Bytes shifted past position 50 are lost. Without OVERRUN=TRUNC, if nonblank characters were shifted past position 50, an overrun error message would be issued and the job would terminate.

SHIFT=YES or SHIFT=NO

Specifies the action DFSORT is to take if an input constant is to be replaced by an output constant of a different length.

SHIFT=YES is the default; it tells DFSORT to shift the bytes after each replaced input constant to the left or right as needed.

Use SHIFT=NO if you want DFSORT to overlay the input constant with the output constant instead of shifting bytes. If a matching input constant is found and the output constant is smaller than the input constant, the current position will be incremented past the output constant rather than past the input constant before processing continues.

Example:

```
  OUTFIL FINDREP=(IN=(C'KW1=YES,',C'KW1=NO,'),OUT=C'KW2',SHIFT=NO)
```

Replaces every C'KW1=YES,' input constant with a C'KW2=YES,' output constant. Replaces every C'KW1=NO,' input constant with a C'KW2=NO,' output constant. SHIFT=NO ensures that C'KW1' is replaced with C'KW2' and the '=YES,' and '=NO,' bytes are kept. Without SHIFT=NO, the '=YES,' and '=NO,' bytes would be removed.

Default for FINDREP: None; must be specified.

FINDREP notes:

- For a single FINDREP operand, when a constant is replaced at the current position, no further checks are performed at that position. So a single FINDREP cannot be used to change a constant and then change it again. For example, if you had an input record with:

  ```
  ABC
  ```

  and used this OUTFIL statement:

  ```
    OUTFIL FINDREP=(INOUT=(C'AB',C'XY',C'XYC',C'RST'))
  ```

  the output record would be:

  ```
  XYC
  ```

  After C'AB' is changed to C'XY', the current pointer is advanced to C'C' so C'XYZ' is not found. However, since each IFTHEN clause starts at the beginning of the record, multiple IFTHEN clauses with FINDREP can be used to change a constant and then change it again. If you used:

  ```
    OUTFIL IFTHEN=(WHEN=INIT,FINDREP=(INOUT=(C'AB',C'XY'))),
      IFTHEN=(WHEN=INIT,FINDREP=(INOUT=(C'XYC',C'RST')))
  ```

  for the ABC record, the output would be:

  ```
  RST
  ```

  The first IFTHEN clause would change C'AB' to C'XY'. The second IFTHEN clause would start over from the first position and change C'XYC' to C'RST'.

- Duplicates and supersets of the same input constant after the first are effectively ignored, whereas subsets of the same input constant after the first are processed. For example, if you had this input record:

  ```
  ABCD ABQ
  ```

  and you used this OUTFIL statement:

  ```
    OUTFIL FINDREP=(INOUT=(C'AB',C'XY',C'ABCD',C'RSTU'))
  ```

  the output record would be:

  ```
  XYCD XYQ
  ```

  Since C'AB' is changed to C'XY', C'ABCD' is not found. If you wanted to change C'ABCD' to C'RSTU' and other instances of C'AB' to C'XY', you would need to specify C'ABCD' first. If you used this OUTFIL statement:

```
  OUTFIL FINDREP=(INOUT=(C'ABCD',C'RSTU',C'AB',C'XY'))
```

the output record would be:

```
RSTU XYQ
```

If C'ABCD' is found, it is changed to C'RSTU'. Otherwise, if C'AB' is found, it is changed to C'XY'.

- For fixed-length records, FINDREP in an IFTHEN clause operates against the maximum record padded with blanks on the right as needed. If a blank constant is being replaced, the blanks on the right will be replaced. For example, if we have an 80 byte FB input record and use:

```
  OUTFIL IFTHEN=(WHEN=INIT,BUILD=(1,20)),
         IFTHEN=(WHEN=(2,1,CH,EQ,C'R'),
           FINDREP=(IN=C' ',OUT=C'ABC'))
```

Although in this case BUILD=(1,20) will result in a 20-byte output record, IFTHEN FINDREP will operate against the maximum 80-byte record padded with 60 blanks on the right. For a record with 'R' in position 2, each of those 60 blanks on the right will be replaced with C'ABC'. This will cause an overrun of the 80 byte record, thus resulting in termination (OVERRUN=ERROR) or truncation (OVERRUN=TRUNC). ENDPOS=20 could be used to stop FINDREP from replacing the 60 blanks on the right.

Note that if we have an 80-byte input record with 'ABC' in positions 1-3 and we use:

```
  OUTFIL IFTHEN=(WHEN=INIT,BUILD=(1,3)),
         IFTHEN=(WHEN=INIT,FINDREP=(IN=C'ABC',OUT=C'12345'))
```

an overrun will not occur because IFTHEN FINDREP will operate against an 80-byte record padded with 77 blanks on the right, rather than against a 3-byte record even though the output record will be 3 bytes.

- For FINDREP in an IFTHEN clause, MAXLEN=n can be used to increase the maximum output length, but cannot be used to decrease the maximum output length. For example, with:

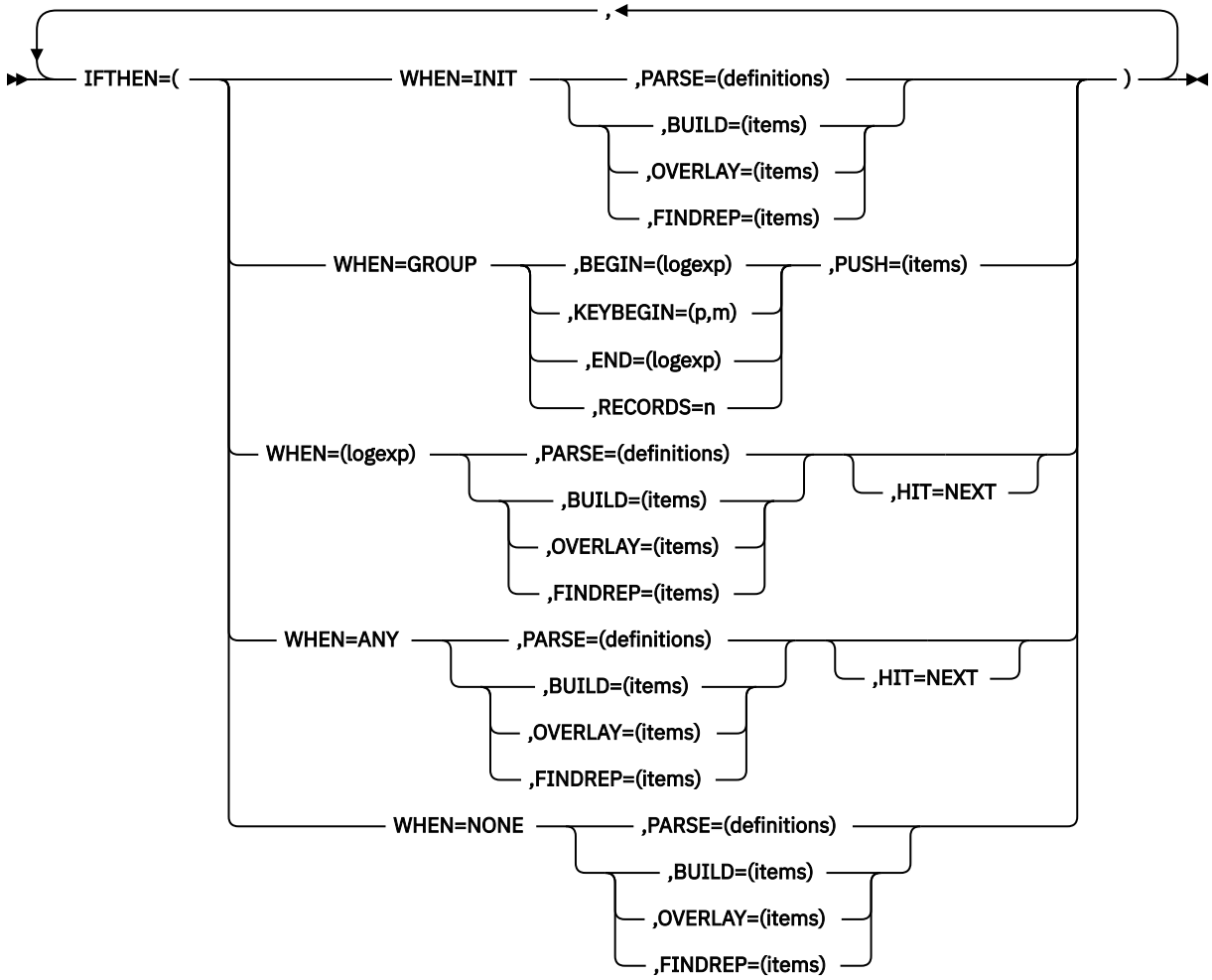
```
  OUTFIL IFTHEN=(WHEN=INIT,BUILD=(1,100)),
         IFTHEN=(WHEN=INIT,
           FINDREP=(IN=C'A',OUT=C'B',MAXLEN=150))
```

the output length will be set to 150, whereas with:

```
  OUTFIL IFTHEN=(WHEN=INIT,BUILD=(1,100)),
         IFTHEN=(WHEN=INIT,
           FINDREP=(IN=C'A',OUT=C'B',MAXLEN=70))
```

the output length will be set to 100.

IFTHEN



IFTHEN clauses allow you to reformat different records in different ways by specifying how build, overlay, find/replace or group operation items are to be applied to records that meet given criteria. IFTHEN clauses let you use simple or complex conditional logic to choose how different record types are reformatted.

If you want to insert, rearrange, or delete fields in the same way for every record, use BUILD or OUTREC rather than IFTHEN. If you want to do find and replace operations in the same way for every record, use FINDREP rather than IFTHEN. Use IFTHEN clauses if you want to insert, rearrange, delete, or overlay fields, or perform find/replace operations in different ways for different records, or if you want to perform operations on groups of records.

You can use five types of IFTHEN clauses as follows:

- **WHEN=INIT:** Use one or more WHEN=INIT clauses to apply build, overlay, or find/replace items to all of your input records. WHEN=INIT clauses and WHEN=GROUP clauses are processed before any of the other IFTHEN clauses.
- **WHEN=GROUP**: Use one or more WHEN=GROUP clauses to propagate fields, identifiers and sequence numbers within specified groups of records. WHEN=INIT clauses and WHEN=GROUP clauses are processed before any of the other IFTHEN clauses.
- **WHEN=(logexp):** Use one or more WHEN=(logexp) clauses to apply build, overlay or find/replace items to your input records that meet specified criteria. A WHEN=(logexp) clause is satisfied when the logical expression evaluates as true.
- **WHEN=ANY:** Use a WHEN=ANY clause after multiple WHEN=(logexp) clauses to apply additional build, overlay or find/replace items to your input records if they satisfied the criteria for any of the preceding WHEN=(logexp) clauses.

- **WHEN=NONE:** Use one or more WHEN=NONE clauses to apply build, overlay or find/replace items to your input records that did not meet the criteria for any of the WHEN=(logexp) clauses. WHEN=NONE clauses are processed after any of the other IFTHEN clauses. If you do not specify a WHEN=NONE clause, only the WHEN=INIT and WHEN=GROUP changes (if any) are applied to input records that do not meet the criteria for any of the WHEN=(logexp) clauses.

IFTHEN clauses are processed in the following order:

- WHEN=INIT clauses and WHEN=GROUP clauses
- WHEN=(logexp) clauses and WHEN=ANY clauses
- WHEN=NONE clauses

Processing of IFTHEN clauses continues unless one of the following occurs:

- A WHEN=(logexp) or WHEN=ANY clause is satisfied, and HIT=NEXT is not specified.
- A WHEN=(logexp), WHEN=ANY or WHEN=NONE clause is satisfied, and BUILD with / is specified (multiple output records).
- There are no more IFTHEN clauses to process. When processing of IFTHEN clauses stops, the IFTHEN record created so far is used as the output record.

  Example:

```
  OUTFIL IFTHEN=(WHEN=(12,1,BI,ALL,X'3F'),OVERLAY=(18:C'Yes')),
         IFTHEN=(WHEN=(35,2,PD,EQ,+14),BUILD=(1,40,45,3,HEX),HIT=NEXT),
         IFTHEN=(WHEN=(35,2,PD,GT,+17),BUILD=(1,40,41,5,HEX),HIT=NEXT),
         IFTHEN=(WHEN=ANY,BUILD=(1,55,C'ABC',70:X)),
         IFTHEN=(WHEN=(35,2,PD,EQ,+8),BUILD=(1,40,2/,45,10)),
         IFTHEN=(WHEN=(63,2,CH,EQ,C'AB'),OVERLAY=(18:C'No')),
         IFTHEN=(WHEN=NONE,BUILD=(1,40,51,8,TRAN=LTOU))
```

  For this example, the IFTHEN clauses are processed as follows:

  – If IFTHEN clause 1 is satisfied, its overlay item is applied and IFTHEN processing stops.
  – If IFTHEN clause 1 is not satisfied, its overlay item is not applied and IFTHEN processing continues.
  – If IFTHEN clause 2 is satisfied, its build items are applied and IFTHEN processing continues.
  – If IFTHEN clause 2 is not satisfied, its build items are not applied and IFTHEN processing continues.
  – If IFTHEN clause 3 is satisfied, its build items are applied and IFTHEN processing continues.
  – If IFTHEN clause 3 is not satisfied, its build items are not applied and IFTHEN processing continues.
  – If IFTHEN clause 4 is satisfied, its build items are applied and IFTHEN processing stops.
  – If IFTHEN clause 4 is not satisfied, its build items are not applied and IFTHEN processing continues.
  – If IFTHEN clause 5 is satisfied, its build items are applied and IFTHEN processing stops.
  – If IFTHEN clause 5 is not satisfied, its build items are not applied and IFTHEN processing continues.
  – If IFTHEN clause 6 is satisfied, its overlay item is applied and IFTHEN processing stops.
  – If IFTHEN clause 6 is not satisfied, its overlay item is not applied and IFTHEN processing continues.
  – If IFTHEN clause 7 is satisfied, its build items are applied and IFTHEN processing stops.
  – If IFTHEN clause 7 is not satisfied, its build items are not applied and IFTHEN processing stops.

All of the IFTHEN clauses operate sequentially on an IFTHEN record. The IFTHEN record is created initially from the input record. Each IFTHEN clause tests and changes the IFTHEN record, as

appropriate. Thus, **changes made by earlier IFTHEN clauses are "seen" by later IFTHEN clauses.** For example, if you have a 40-byte input record and specify:

```
  OUTFIL IFTHEN=(WHEN=INIT,OVERLAY=(8:8,4,ZD,ADD,+1,TO=ZD,LENGTH=4)),
         IFTHEN=(WHEN=(8,4,ZD,EQ,+27),OVERLAY=(28:C'Yes')),
         IFTHEN=(WHEN=NONE,OVERLAY=(28:C'No'))
```

The WHEN=INIT clause adds 1 to the ZD value and stores it in the IFTHEN record. The WHEN=(8,4,ZD,EQ,+27) clause tests the incremented ZD value in the IFTHEN record rather than the original ZD value in the input record.

The IFTHEN record is adjusted as needed for the records created or changed by the IFTHEN clauses. For fixed-length records, blanks are filled in on the left as needed. For variable-length records, the RDW length is adjusted as needed each time the IFTHEN record is changed.

Missing bytes in specified input fields are replaced with blanks so the padded fields can be processed.

DFSORT sets an appropriate LRECL for the OUTFIL output records based on the build, overlay, find/replace and group operation items specified by the IFTHEN clauses. However, DFSORT does not analyze the possible results of WHEN=(logexp) conditions when determining an appropriate LRECL. When you use OUTFIL IFTHEN clauses, you can override the OUTFIL LRECL determined by DFSORT with the OUTFIL IFOUTLEN parameter.

If SEQNUM is used in multiple IFTHEN clauses, the sequence number will be incremented for each record that satisfies the IFTHEN clause, that is, a separate SEQNUM counter will be kept for each IFTHEN clause. For example, if your input is:

```
RECORD A 1
RECORD B 1
RECORD B 2
RECORD C 1
RECORD A 2
RECORD C 2
RECORD B 3
RECORD D 1
```

and you specify:

```
  OUTFIL IFTHEN=(WHEN=(8,1,CH,EQ,C'A'),OVERLAY=(15:SEQNUM,4,ZD)),
         IFTHEN=(WHEN=(8,1,CH,EQ,C'B'),OVERLAY=(16:SEQNUM,4,ZD)),
         IFTHEN=(WHEN=NONE,OVERLAY=(17:SEQNUM,4,ZD))
```

your output will be:

```
RECORD A 1    0001
RECORD B 1     0001
RECORD B 2     0002
RECORD C 1      0001
RECORD A 2    0002
RECORD C 2      0002
RECORD B 3     0003
RECORD D 1      0003
```

Separate SEQNUM counters are kept for the 'A' record, for the 'B' record, and for the NONE records.

You can use IFTHEN clauses with the FTOV parameter to convert fixed-length record data sets to variable-length record data sets.

You can use the VLTRIM parameter with IFTHEN clauses to remove specified trailing bytes from the end of variable-length records.

You can use the VLTRAIL parameter with IFTHEN clauses to insert a string at the end of variable-length records.

You can use IFTHEN clauses with any or all of the report parameters (LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS, BLKCCH1, BLKCCH2, BLKCCT1, and NODETAIL) in the same way as for the BUILD or OUTREC parameter.

The IFTHEN clauses of the OUTREC statement apply to all input records whereas the IFTHEN clauses of the OUTFIL statement only apply to the OUTFIL input records for its OUTFIL group.

WHEN=INIT clause

Specifies build, overlay or find/replace items to be applied to all records. Multiple WHEN=INIT clauses and WHEN=GROUP clauses may be intermixed. They are processed before any other type of IFTHEN clauses and in the order in which they are specified.

WHEN=INIT

Identifies a WHEN=INIT clause.

PARSE=(definitions)

Defines %nn fixed parsed fields into which variable position/length fields are extracted for all records. See OUTFIL PARSE for details. You can use the %nn parsed fields defined in a WHEN=INIT clause in the BUILD or OVERLAY operand of that clause and all subsequent IFTHEN clauses.

Sample Syntax:

```
OUTFIL IFOUTLEN=50,
       IFTHEN=(WHEN=INIT,
        PARSE=(%01=(ABSPOS=11,STARTAT=NONBLANK,
                    ENDBEFR=C',',FIXLEN=12),
               %02=(ENDBEFR=C',',FIXLEN=10),
               %03=(FIXLEN=12))),
       IFTHEN=(WHEN=(5,2,CH,EQ,C'US'),BUILD=(%02,%03)),
       IFTHEN=(WHEN=(5,2,CH,EQ,C'UK'),BUILD=(%01,%02)),
       IFTHEN=(WHEN=NONE,BUILD=(%03,%01))
```

BUILD=(items)

Specifies the build items to be applied to each record. See OUTFIL BUILD for details. You can specify all of the items in the same way that you can specify them for OUTFIL BUILD, except that you cannot specify / to create blank records or new records.

OVERLAY=(items)

Specifies the overlay items to be applied to each record. See OUTFIL OVERLAY for details. You can specify all of the items in the same way that you can specify them for OUTFIL OVERLAY.

Sample Syntax:

```
OUTFIL FNAMES=OUT1,
       IFTHEN=(WHEN=INIT,BUILD=(1,20,21:C'Department',31:3X,21,60)),
       IFTHEN=(WHEN=(5,2,CH,EQ,C'D1'),OVERLAY=(31:8,3)),
       IFTHEN=(WHEN=(5,2,CH,EQ,C'D2'),OVERLAY=(31:12,3))
```

FINDREP=(items)

Specifies find and replace operations to be applied to each record. See OUTFIL FINDREP for details. You can specify all of the items in the same way that you can specify them for OUTFIL FINDREP.

Sample Syntax

```
OUTFIL FNAMES=OUT2,
  IFTHEN=(WHEN=INIT,FINDREP=(IN=X'00',OUT=X'40')),
  IFTHEN=(WHEN=INIT,OVERLAY=(81:SEQNUM,8,ZD))
```

WHEN=GROUP clause

Identifies groups of records in various ways and propagates fields, identifiers and sequence numbers to the records of each group. Fields, identifiers and sequence numbers are not propagated to records before, between or after identified groups. Multiple WHEN=INIT clauses and WHEN=GROUP clauses may be intermixed. They are processed before any other type of IFTHEN clauses and in the order in which they are specified.

You can specify the BEGIN, KEYBEGIN, END and RECORDS operands in any combination to define the groups, but you must specify at least one of these operands.

BEGIN=(logexp)

Specifies the criteria to be tested to determine if a record starts a group. See the INCLUDE statement for details of the logical expressions you can use. You can specify all of the logical expressions in the same way that you can specify them for the INCLUDE statement except that:

- You cannot specify FORMAT=f with BEGIN=(logexp).
- You cannot specify D2 format in BEGIN=(logexp).
- Locale processing is not used for BEGIN=(logexp).
- VLSCMP and VLSHRT are not used with BEGIN=(logexp). Instead, missing bytes in specified input fields are replaced with blanks so the padded fields can be processed.

A new group starts with a record that satisfies the BEGIN condition, that is, when the specified logical expression is true for that record. If BEGIN is specified without END or RECORDS, all of the records from the begin record up to but not including the next begin record belong to a group. Here's an example of groups with

```
BEGIN=(1,1,CH,EQ,C'A'):

H
R
A group 1
B group 1
C group 1
A group 2
A group 3
B group 3
```

Example:

```
  OUTFIL IFTHEN=(WHEN=GROUP,
    BEGIN=(1,40,SS,EQ,C'J82',OR,1,40,SS,EQ,C'M72'),
    PUSH=(41:ID=5))
```

Starts a new group each time C'J82' or 'M72' is found anywhere in positions 1-40 of a record. Overlays positions 41-45 of each record of a group with a 5-byte ZD identifier. The records before the first group are not changed.

KEYBEGIN=(p,m)

Specifies a field to be used to determine if a record starts a group. A new group starts when the binary value in the specified key field (p,m) changes. The first input record will start a group. KEYBEGIN=(p,m) operates in a similar way to BEGIN=(logexp) but uses a key change as a trigger rather than a condition to be satisfied. Use KEYBEGIN=(p,m) if you want to start a new group for each consecutive set of key values in your records.

p is the starting position of the key and can be from 1 to 32752.

m is the length of the key and can be from 1 to 256.

If KEYBEGIN=(p,m) is specified, the first input record will start a group.

Here is an example of groups with

```
KEYBEGIN=(1,1)

A group 1
A group 1
A group 1
B group 2
B group 2
C group 3
```

Example:

```
  OUTFIL IFTHEN=(WHEN=GROUP,
     KEYBEGIN=(11,5),PUSH=(81:21,8))
```

Starts a new group each time the value in positions 11-15 changes. Overlays positions 81-88 of each record of a group with positions 21-28 from the first record of that group.

END=(logexp)

Specifies the criteria to be tested to determine if a record ends a group. See the INCLUDE statement for details of the logical expressions you can use. You can specify all of the logical expressions in the same way that you can specify them for the INCLUDE statement except that:

- You cannot specify FORMAT=f with END=(logexp).
- You cannot specify D2 format in END=(logexp).
- Locale processing is not used for END=(logexp).
- VLSCMP and VLSHRT are not used with END=(logexp). Instead, missing bytes in specified input fields are replaced with blanks so the padded fields can be processed.

A group ends whenever a record satisfies the END condition, that is, whenever the specified logical expression is true for that record. If END is specified without BEGIN or KEYBEGIN, all of the records up to the end record (or the last record of the data set) belong to a group. Here's an example of groups with

```
END=(1,1,CH,EQ,C'T')

A group 1
B group 1
T group 1
T group 2
A group 3
T group 3
M group 4
```

If END is specified with BEGIN or KEYBEGIN, all of the records from the begin record up to the end record (or the last record of the data set) belong to a group. Here's an example of groups with

```
BEGIN=(1,1,CH,EQ,C'H'),END=(1,1,CH,EQ,C'T')

H group 1
B group 1
T group 1
T
H group 2
T group 2
M
N
H group 3
A group 3
```

Example:

```
  OUTFIL IFTHEN=(WHEN=GROUP,
           BEGIN=(1,2,CH,EQ,C'02',AND,8,3,CH,EQ,C'YES'),
           END=(11,5,CH,EQ,C'PAGE:'),PUSH=(61:SEQ=3))
```

Starts a new group each time C'02' is found in positions 1-2 and C'YES' is found in positions 8-10 of a record. Ends the group when C'PAGE:' is found in positions 11-15 of a record. Overlays positions 61-63 of each record of a group with a 3-byte ZD sequence number. The records between the groups are not changed.

RECORDS=n

Specifies the maximum number of records in a group. n can be 1 to 2000000000. If RECORDS is specified without BEGIN, KEYBEGIN or END, a new group starts after n records. Here's an example of groups with

```
RECORDS=3

H group 1
B group 1
```

```
T group 1
H group 2
B group 2
T group 2
M group 3
N group 3
```

If RECORDS is specified with BEGIN or KEYBEGIN, up to n records starting with the begin record belong to a group. Here's an example of groups with

```
BEGIN=(1,1,CH,EQ,C'H'),RECORDS=3

H group 1
B group 1
T group 1
A
H group 2
B group 2
H group 3
M group 3
N group 3
P
```

The records between and after the groups are not changed.

If RECORDS is specified with END, the group ends after up to n records or with the end record (or the last record of the data set), whichever comes first. Here's an example of groups with

```
BEGIN=(1,1,CH,EQ,C'H'),END=(1,1,CH,EQ,C'T'),RECORDS=4

H group 1
B group 1
T group 1
A
H group 2
B group 2
C group 2
D group 2
E
H group 3
M group 3
```

The records between the groups are not changed.

You must specify the PUSH operand to define at least one field, identifier or sequence number to be added to the records of each group. You can specify any combination or number of fields, identifiers and sequence numbers.

PUSH=(c:item,...)

Specifies the position where each field, identifier or sequence number is to be overlaid in the records of each group.

For fixed-length records, the first input and output data byte starts at position 1. For variable-length records, the first input and output data byte starts at position 5, after the RDW in positions 1-4.

You can use the following in PUSH:

c:

Specifies the output position (column) to be overlaid. If you do not specify c: for the first item, it defaults to 1:. If you do not specify c: for any other item, it starts after the previous item. You can specify items in any order and overlap output columns. c can be 1 to 32752.

If you specify an item that extends the output record beyond the end of the input record, the record length is automatically increased to that length, and blanks are filled in on the left as needed. For variable-length records, the RDW length is increased to correspond to the larger record length after all of the items are processed. Missing bytes in specified input fields are replaced with blanks so the padded fields can be processed.

p,m
Specifies a field in the first input record of each group to be propagated to every record of the group. p specifies the starting position of the field in the input record and m specifies its length. A field must not extend beyond position 32752.

ID=n
Specifies a ZD identifier of length n is to be added to every record of each group. The identifier starts at 1 for the first group and is incremented by 1 for each subsequent group. n can be 1 to 15.

SEQ=n
Specifies a ZD sequence number of length n is to be added to every record of each group. The sequence number starts at 1 for the first record of each group and is incremented by 1 for each subsequent record of the group. n can be 1 to 15.

Sample Syntax

```
OUTFIL IFTHEN=(WHEN=GROUP,
         BEGIN=(1,5,CH,EQ,C'DATE:'),
         RECORDS=3,
         PUSH=(15:ID=3,31:21,8))
```

WHEN=(logexp) clause
Specifies build, overlay or find/replace items to be applied to records for which the specified logical expression is true. If multiple WHEN=(logexp) clauses are specified, they are processed in the order in which they are specified.

WHEN=(logexp)
Identifies a WHEN=(logexp) clause and specifies the criteria to be tested to determine if the build, overlay or find/replace items are to be applied to the record. See the INCLUDE statement for details of the logical expressions you can use. You can specify all of the logical expressions in the same way that you can specify them for the INCLUDE statement except that:

- You cannot specify FORMAT=f with WHEN=(logexp).
- You cannot specify D2 format in WHEN=(logexp).
- Locale processing is not used for WHEN=(logexp).
- VLSCMP and VLSHRT are not used with WHEN=(logexp). Instead, missing bytes in specified input fields are replaced with blanks so the padded fields can be processed.

PARSE=(definitions)
Defines %nn fixed parsed fields into which variable position/length fields are extracted for each record for which the logical expression is true. See OUTFIL PARSE for details. You can only use the %nn parsed fields defined in a WHEN=(logexp) clause in the BUILD or OVERLAY operand of that WHEN=(logexp) clause.

Sample Syntax:

```
OUTFIL IFTHEN=(WHEN=(45,2,CH,EQ,C'AL'),
         PARSE=(%01=(ENDBEFR=C',',FIXLEN=12),
                %02=(FIXLEN=10)),
         OVERLAY=(21:%02,51:%01))
```

BUILD=(items)
Specifies the build items to be applied to each record for which the logical expression is true. See OUTFIL BUILD for details. You can specify all of the items in the same way that you can specify them for OUTFIL BUILD.

OVERLAY=(items)
Specifies the overlay items to be applied to each record for which the logical expression is true. See OUTFIL OVERLAY for details. You can specify all of the items in the same way that you can specify them for OUTFIL OVERLAY.

FINDREP=(items)
Specifies find and replace operations to be applied to each record for which the logical expression is true. See OUTFIL FINDREP for details. You can specify all of the items in the same way that you can specify them for OUTFIL FINDREP.

Sample Syntax

```
  OUTFIL IFTHEN=(WHEN=(11,2,CH,EQ,C'**'),
    FINDREP=(IN=C'**',OUT=C''))
```

HIT=NEXT
Specifies that IFTHEN processing should continue even if the logical expression is true. By default (if HIT=NEXT is not specified), IFTHEN processing stops if the logical expression is true.

Sample Syntax:

```
OUTFIL FNAMES=OUT2,
  IFTHEN=(WHEN=(1,3,CH,EQ,C'T01',AND,
     18,4,ZD,LE,+2000),OVERLAY=(42:C'Type1 <= 2000'),HIT=NEXT),
  IFTHEN=(WHEN=(1,3,CH,EQ,C'T01',AND,6,1,BI,BO,X'03'),
      BUILD=(1,21,42,13)),
  IFTHEN=(WHEN=(1,3,CH,EQ,C'T01',AND,
     18,4,ZD,GT,+2000),OVERLAY=(42:C'Type1 > 2000 '),HIT=NEXT),
  IFTHEN=(WHEN=(1,3,CH,EQ,C'T01',AND,6,1,BI,BO,X'01'),
      BUILD=(1,25,42,13))
```

WHEN=ANY clause
Specifies build, overlay or find/replace items to be applied to records for which the logical expression in any "preceding" WHEN=(logexp) clause is true. For the first WHEN=ANY clause, the "preceding" WHEN=(logexp) clauses are those before this WHEN=ANY clause. For the second or subsequent WHEN=ANY clause, the "preceding" WHEN=(logexp) clauses are those between the previous WHEN=ANY clause and this WHEN=ANY clause. At least one WHEN=(logexp) clause must be specified before a WHEN=ANY clause. A WHEN=ANY clause can be used without any build, overlay or find/replace items to just stop IFTHEN processing if any preceding WHEN=(logexp) clause is satisfied.

PARSE=(definitions)
Defines %nn fixed parsed fields into which variable position/length fields are extracted for each record for which the logical expression in any preceding WHEN=(logexp) clause is true. See OUTFIL PARSE for details. You can only use the %nn parsed fields defined in a WHEN=ANY clause in the BUILD or OVERLAY operand of that WHEN=ANY clause.

BUILD=(items)
Specifies the build items to be applied to each record for which the logical expression in any preceding WHEN=(logexp) clause is true. See OUTFIL BUILD for details. You can specify all of the items in the same way that you can specify them for OUTFIL BUILD.

OVERLAY=(items)
Specifies the overlay items to be applied to each record for which the logical expression in any preceding WHEN=(logexp) clause is true. See OUTFIL OVERLAY for details. You can specify all of the items in the same way that you can specify them for OUTFIL OVERLAY.

FINDREP=(items)
Specifies find and replace operations to be applied to each record for which the logical expression in any preceding WHEN=(logexp) clause is true. See OUTFIL FINDREP for details. You can specify all of the items in the same way that you can specify them for OUTFIL FINDREP.

HIT=NEXT
Specifies that IFTHEN processing should continue even if the logical expression in a preceding WHEN=(logexp) clause is true. By default (if HIT=NEXT is not specified), IFTHEN processing stops if the logical expression in a preceding WHEN=(logexp) clause is true.

Sample Syntax:

```
OUTFIL FNAMES=OUT3,
 IFTHEN=(WHEN=(1,3,SS,EQ,C'T01,T02,T03'),
           BUILD=(C'Group A',X,1,80),HIT=NEXT),
 IFTHEN=(WHEN=(1,3,SS,EQ,C'T04,T05,T06'),
           BUILD=(C'Group B',X,1,80),HIT=NEXT),
 IFTHEN=(WHEN=(1,3,SS,EQ,C'T07,T08,T09,T10'),
           BUILD=(C'Group C',X,1,80),HIT=NEXT),
 IFTHEN=(WHEN=ANY,OVERLAY=(16:C'Group Found'))
```

WHEN=NONE clause

Specifies build, overlay or find/replace items to be applied to records for which none of the logical expressions in any WHEN=(logexp) clause is true. If there are no WHEN=(logexp) clauses, the build, overlay or find/replace items are applied to all of the records. If multiple WHEN=NONE clauses are specified, they are processed in the order in which they are specified. WHEN=NONE clauses are processed after any other type of IFTHEN clauses.

PARSE=(definitions)

Defines %nn fixed parsed fields into which variable position/length fields are extracted for each record for which no logical expression was true. See OUTFIL PARSE for details. You can only use the %nn parsed fields defined in a WHEN=NONE clause in the BUILD or OVERLAY operand of that WHEN=NONE clause.

Sample Syntax:

```
OUTFIL IFTHEN=(WHEN=(5,2,ZD,GT,+5),
  PARSE=(%01=(STARTAT=C'<',ENDAT=C'>',FIXLEN=12),
         %02=(STARTAFT=BLANKS,FIXLEN=10)),
  BUILD=(5,2,21:%01,X,%02,HEX)),
 IFTHEN=(WHEN=NONE,
  PARSE=(%03=(STARTAFT=C'(',ENDBEFR=C')',FIXLEN=8)),
  BUILD=(5,2,%03,SFF,M12))
```

BUILD=(items)

Specifies the build items to be applied to each record for which no logical expression was true. See OUTFIL BUILD for details. You can specify all of the items in the same way that you can specify them for OUTFIL BUILD.

OVERLAY=(items)

Specifies the overlay items to be applied to each record for which no logical expression was true. See OUTFIL OVERLAY for details. You can specify all of the items in the same way that you can specify them for OUTFIL OVERLAY.

Sample Syntax:

```
OUTFIL FNAMES=OUT4,
  IFTHEN=(WHEN=INIT,BUILD=(1,20,21:C'Department',31:3X,21,60)),
  IFTHEN=(WHEN=(5,2,CH,EQ,C'D1'),OVERLAY=(31:8,3)),
  IFTHEN=(WHEN=(5,2,CH,EQ,C'D2'),OVERLAY=(31:12,3)),
  IFTHEN=(WHEN=NONE,OVERLAY=(31:C'***'))
```

FINDREP=(items)

Specifies find and replace operations to be applied to each record for which no logical expression was true. See OUTFIL FINDREP for details. You can specify all of the items in the same way that you can specify them for OUTFIL FINDREP.

Default for IFTHEN clauses: None; must be specified.

IFOUTLEN

Overrides the OUTFIL LRECL determined by DFSORT from your OUTFIL IFTHEN clauses. DFSORT sets an appropriate LRECL for the output records based on the build, overlay, find/replace and group operation items specified by the IFTHEN clauses. However, DFSORT does not analyze the possible results of WHEN=(logexp) conditions when determining an appropriate OUTFIL LRECL. When you use

OUTFIL IFTHEN clauses, you can override the OUTFIL LRECL determined by DFSORT with the OUTFIL IFOUTLEN parameter.

Fixed-length records longer than the IFOUTLEN length are truncated to the IFOUTLEN length. Fixed-length records shorter than the IFOUTLEN are padded with blanks to the IFOUTLEN length. Variable-length records longer than the IFOUTLEN length are truncated to the IFOUTLEN length.

n
specifies the length to use for the OUTFIL LRECL. The value for n must be between 1 and 32767, but must not be larger than the maximum LRECL allowed for the RECFM, and must not conflict with the specified or retrieved LRECL for the fixed-length OUTFIL data set.

Sample Syntax:

```
OUTFIL FNAMES=OUT5,IFOUTLEN=70,
  IFTHEN=(WHEN=(5,1,CH,EQ,C'1',AND,8,3,ZD,EQ,+10),
    BUILD=(1,40,C'T01-GROUP-A',65)),
  IFTHEN=(WHEN=(5,1,CH,EQ,C'2',AND,8,3,ZD,EQ,+12),
    BUILD=(1,40,C'T02-GROUP-B',65))
```

Default for IFOUTLEN: The LRECL determined from the IFTHEN clauses.

VTOF or CONVERT

Specifies that variable-length OUTFIL input records are to be converted to fixed-length OUTFIL output records for this OUTFIL group.

You must specify a BUILD or OUTREC parameter. The items you specify produce a reformatted fixed-length OUTFIL output record without an RDW (the data starts at position 1). Any BUILD or OUTREC fields you specify apply to the variable-length OUTFIL input records (the data starts at position 5 after the 4-byte RDW). However, you cannot specify the variable-part of the OUTFIL input records (for example, p or p,HEX). Any BUILD or OUTREC columns you specify apply to the reformatted fixed-length OUTFIL output records.

By default, VTOF or CONVERT automatically uses VLFILL=X'40' (blank fill byte) to allow processing of variable-length input records which are too short to contain all specified BUILD or OUTREC fields. You can specify VLFILL=byte to change the fill byte.

If you do not specify a RECFM for the OUTFIL data set, it will be given a record format of FB.

If you specify a RECFM for the OUTFIL data set, it must have a fixed-length record format (for example, FB).

If VTOF or CONVERT is specified for fixed-length input records, it will not be used.

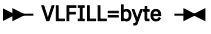
If VTOF or CONVERT is specified with FTOV, IFTRAIL, IFTHEN, FINDREP or OVERLAY, DFSORT will terminate.

Sample Syntax:

```
OUTFIL FNAMES=FIXOUT,VTOF,
  OUTREC=(1:5,14,35:32,8,50:22,6,7c'*')
```

Default for VTOF or CONVERT: None; must be specified.

VLFILL



Allows DFSORT to continue processing if a variable-length OUTFIL input record is found to be too short to contain all specified OUTFIL BUILD or OUTREC fields for this OUTFIL group. Without VLFILL=byte, a short record causes DFSORT to issue message ICE218A and terminate. With

VLFILL=byte, missing bytes in OUTFIL BUILD or OUTREC fields are replaced with fill bytes so the filled fields can be processed.

If VLFILL=byte is specified for fixed-length input records, it will not be used.

If VLFILL=byte is specified with FTOV, IFTRAIL, IFTHEN, FINDREP or OVERLAY, DFSORT will terminate.

byte
specifies the fill byte. Permissible values are C'x' and X'yy'.

C'x'
Character byte: The value x must be one EBCDIC character. If you want to use an apostrophe as the fill byte, you must specify it as C''''.

X'yy'
Hexadecimal byte: The value yy must be one pair of hexadecimal digits (00-FF).

Sample Syntax:

```
OUTFIL FNAMES=FIXOUT,VTOF,OUTREC=(5,20,2X,35,10),VLFILL=C'*'
OUTFIL FNAMES=OUT1,VLFILL=X'FF',OUTREC=(1,4,15,5,52)
```

Default for VLFILL: VLFILL=X'40' (blank fill byte) if VTOF or CONVERT is specified. Otherwise, none; must be specified.

FTOV

Specifies that fixed-length OUTFIL input records are to be converted to variable-length OUTFIL output records for this OUTFIL group.

If you do not specify an OUTREC, BUILD, OVERLAY, FINDREP or IFTHEN parameter, the fixed-length OUTFIL input record is converted to a variable-length OUTFIL output record. A 4-byte RDW is prepended to the fixed-length record before it is written.

If you specify an OUTREC, BUILD, OVERLAY, FINDREP or IFTHEN parameter, the items you specify produce a reformatted fixed-length record that is converted to a variable-length OUTFIL output record. Any OUTREC, BUILD, OVERLAY, FINDREP or IFTHEN fields you specify apply to the fixed-length OUTFIL input records (the data starts at position 1). A 4-byte RDW is prepended to the reformatted fixed-length record before it is written.

If you do not specify a RECFM for the OUTFIL data set, it will be given a record format of VB.

If you specify a RECFM for the OUTFIL data set, it must have a variable-length record format (for example, VB or VBS).

If you do not specify an LRECL for the OUTFIL data set, it will be given an LRECL that can contain the largest variable-length output record to be produced, up to a maximum of 32756 for an unspanned record format (for example, VB) or up to 32767 for a spanned record format (for example, VBS).

If you specify an LRECL for the OUTFIL data set, it must be big enough to contain the largest variable-length output record to be produced.

If your largest variable-length output record is between 32757 and 32767 bytes, you'll need to specify a spanned record format (for example, VBS) for the output data set.

If FTOV is specified for variable-length input records, it will not be used.

If FTOV is specified with VTOF, IFTRAIL, CONVERT or VLFILL=byte, DFSORT will terminate.

Sample Syntax:

```
OUTFIL FNAMES=VAROUT,FTOV
OUTFIL FNAMES=V1,FTOV,OUTREC=(1,20,26:21,10,6C'*')
OUTFIL FNAMES=V2,FTOV,
```

```
     IFTHEN=(WHEN=(12,5,ZD,GT,+20000),OVERLAY=(25:C'Yes')),
     IFTHEN=(WHEN=NONE,OVERLAY=(25:C'No'))
```

Default for FTOV: None; must be specified.

VLTRIM

VLTRIM=byte specifies that the trailing bytes are to be removed from the end of variable-length OUTFIL output records for this OUTFIL group before the records are written.

The trim byte can be any value, such as blank, binary zero, or asterisk. If DFSORT finds one or more trim bytes at the end of a variable-length OUTFIL data record or report record, it will decrease the length of the record accordingly, effectively removing the trailing trim bytes. However, VLTRIM=byte will not remove the RDW, the ANSI carriage control character (if produced), or the first data byte.

For example, say that you have the following 17-byte fixed-length data records that you want to convert to variable-length data records:

```
123456**********
0003************
ABCDEFGHIJ*****22
****************
```

If you use:

```
OUTFIL FTOV
```

the following variable-length output records will be written (4-byte RDW followed by data):

```
Length  |  Data
  21       123456**********
  21       0003************
  21       ABCDEFGHIJ*****22
  21       ****************
```

but if you use:

```
OUTFIL FTOV,VLTRIM=C'*'
```

the following variable-length output records will be written (4-byte RDW followed by data):

```
Length  |  Data
  10       123456
   8       0003
  21       ABCDEFGHIJ*****22
   5       *
```

VLTRIM=C'*' removed the trailing asterisks from the first and second records. The third record did not have any trailing asterisks to remove. The fourth record had all asterisks, so one asterisk was kept.

If VLTRIM=byte is specified for fixed-length output records, it will not be used.

If VLTRIM is specified with IFTRAIL, DFSORT will terminate.

byte
specifies the trim byte. Permissible values are C'x' and X'yy'.

C'x'
Character byte: The value x must be one EBCDIC character. If you want to use an apostrophe as the trim byte, you must specify it as C''''.

X'yy'
Hexadecimal byte: The value yy must be one pair of hexadecimal digits (00-FF)

Sample Syntax:

```
Fixed input:
 OUTFIL FNAMES=TRIM1,FTOV,VLTRIM=C' '

Variable input:
 OUTFIL FNAMES=TRIM2,VLTRIM=X'00'
 OUTFIL FNAMES=TRIM3,VLTRIM=C'*',
    OUTREC=(1,15,5X,16,8,5X,28)
```

Default for VLTRIM: None; must be specified.

VLTRAIL

VLTRAIL=string specifies that the indicated string is to be inserted at the end of variable-length OUTFIL output records for this OUTFIL group before the records are written.

string can be 1 to 50 characters specified using a character string constant (C'xx...x') or a hexadecimal string constant (X'yy...yy'). See "INCLUDE control statement" on page 91 for details of coding character and hexadecimal string constants.

DFSORT will insert the specified string at the end of each variable-length OUTFIL data record or report record, and increase the length of the record accordingly. You must ensure that the LRECL of the OUTFIL data set is large enough to contain the added bytes. If the added bytes increase the length of a record beyond the LRECL for the data set, DFSORT will terminate. To avoid this, you can specify a larger LRECL on the DD statement, when necessary.

For example, say that you have the following variable-length input records:

```
Length | Data
    28   Professional Development
    18   Psychoanalysis
    25   Theory of Computation
```

and you want to add X'0D0A' (CR,LF) to the end of each record. If you use the following OUTFIL statement:

```
  OUTFIL FNAMES=OUT1,VLTRAIL=X'0D0A'
```

X'0D0A' will be inserted at the end of each record and the length of each record will be increased by 2 bytes. If we represent X'0D0A' by CL, the OUT1 records will look like this:

```
Length | Data
    30   Professional DevelopmentCL
    20   PsychoanalysisCL
    27   Theory of ComputationCL
```

Note that if the LRECL of the output data set is less than the maximum output record (30 bytes for this example), we must increase it to at least the maximum (LRECL=30 or greater for this example).

If VLTRIM=byte is used with VLTRAIL=string, the indicated bytes will be trimmed from the end of the record before the indicated string is added. For example, say that you have the following variable-length input records:

```
Length | Data
    30   Abby is a sweet rat*******
    31   Samantha is a curious rat**
    43   Rachel is our oldest rat***************
    29   Cathy is a mischief maker
```

and you want to remove any trailing asterisks, and add '|March 21|' at the end of each record. If you use the following OUTFIL statement:

```
  OUTFIL FNAMES=OUT2,VLTRIM=C'*',VLTRAIL=C'|March 21|'
```

The trailing asterisks will be removed, 'March 21' will be inserted, and the length of each record will be updated appropriately. The OUT2 records will look like this:

```
Length | Data
    33   Abby is a sweet rat|March 21|
    39   Samantha is a curious rat|March 21|
    38   Rachel is our oldest rat|March 21|
    39   Cathy is a mischief maker|March 21|
```

If VLTRAIL=string is specified for fixed-length output records, it will not be used.

If VLTRAIL is specified with IFTRAIL, DFSORT will terminate.

Sample Syntax:

```
Variable input:
  OUTFIL FNAMES=ADD1,VLTRIM=X'00',VLTRAIL=C'$$ADD$$'
  OUTFIL FNAMES=(ADD2A,ADD2B),VLTRAIL=X'FFFF',SPLIT

Fixed input:
   OUTFIL FNAMES=ADD3,FTOV,VLTRIM=C' ',VLTRAIL=C'end'
```

Default for VLTRAIL: None; must be specified.

REPEAT

Specifies the number of times each OUTFIL output record is to be repeated for this OUTFIL group. Each OUTFIL output record is written n times.

If SEQNUM is used in the OUTREC, BUILD, OVERLAY, or IFTHEN parameter for this OUTFIL group, the sequence number will be incremented for each repeated record. For example, if your input is:

```
RECORD A
RECORD B
```

and you specify:

```
  OUTFIL OUTREC=(1,8,X,SEQNUM,5,ZD),REPEAT=2
```

your output will be:

```
RECORD A 00001
RECORD A 00002
RECORD B 00003
RECORD B 00004
```

If SEQNUM is used in multiple IFTHEN clauses for this OUTFIL group, the sequence number will be incremented for each repeated record that satisfies the IFTHEN clause. For example, if your input is:

```
RECORD A 1
RECORD B 1
RECORD C 1
RECORD A 2
RECORD C 2
RECORD B 2
RECORD B 3
```

and you specify:

```
  OUTFIL REPEAT=2,
    IFTHEN=(WHEN=(8,1,CH,EQ,C'A'),OVERLAY=(15:SEQNUM,4,ZD)),
    IFTHEN=(WHEN=(8,1,CH,EQ,C'B'),OVERLAY=(15:SEQNUM,4,ZD)),
    IFTHEN=(WHEN=NONE,OVERLAY=(15:SEQNUM,4,ZD))
```

your output will be:

```
RECORD A 1    0001
RECORD A 1    0002
RECORD B 1    0001
RECORD B 1    0002
RECORD C 1    0001
RECORD C 1    0002
RECORD A 2    0003
RECORD A 2    0004
RECORD C 2    0003
RECORD C 2    0004
RECORD B 2    0003
RECORD B 2    0004
RECORD B 3    0005
RECORD B 3    0006
```

If you specify REPEAT=n with / in the OUTREC, BUILD, or IFTHEN BUILD parameter for this OUTFIL group, the first line is written n times, then the second line is written n times, and so on. (For IFTHEN, this means each record that has the SEQNUM item applied to it.) If SEQNUM is used, all lines for the same record are given the same sequence number. For example, if your input is:

```
RECORD A
RECORD B
```

and you specify:

```
 OUTFIL OUTREC=(C'P1>',X,1,6,X,SEQNUM,4,ZD,/,
   C'P2>',X,8,1,X,SEQNUM,4,ZD),REPEAT=2
```

your output will be:

```
P1> RECORD 0001
P1> RECORD 0002
P2> A 0001
P2> A 0002
P1> RECORD 0003
P1> RECORD 0004
P2> B 0003
P2> B 0004
```

The REPEAT parameter cannot be used with any of the following parameters: IFTRAIL, LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS, and NODETAIL.

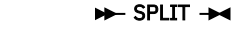
n
specifies the number of times each OUTFIL output record is to be repeated. The value for n starts at 2 (write record twice) and is limited to 28 digits (15 significant digits).

Sample Syntax:

```
* WRITE EACH OUTPUT RECORD 12 TIMES.
  OUTFIL FNAMES=OUT1,REPEAT=12

* WRITE EACH INCLUDED AND REFORMATTED OUTPUT RECORD 50 TIMES.
* (THE SEQUENCE NUMBER WILL BE INCREMENTED FOR EACH REPETITION.)
  OUTFIL FNAMES=OUT2,INCLUDE=(5,2,SS,EQ,C'B2,C5,M3'),
```

Default for REPEAT: None; must be specified.

SPLIT



Splits the output records one record at a time in rotation among the data sets of this OUTFIL group until all of the output records have been written. As a result, the records will be split as evenly as possible among all of the data sets in the group.

As an example, for an OUTFIL group with three data sets:

- the first OUTFIL data set in the group will receive records 1, 4, 7, and so on.

- the second OUTFIL data set in the group will receive records 2, 5, 8, and so on.
- the third OUTFIL data set in the group will receive records 3, 6, 9, and so on.

The records are **not** contiguous in each OUTFIL data set.

SPLIT is equivalent to SPLITBY=1.

The SPLIT parameter cannot be used with any of the following parameters: IFTRAIL, LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS, and NODETAIL.

Sample Syntax:

```
* WRITE RECORD 1 TO PIPE1, RECORD 2 TO PIPE2, RECORD 3 TO PIPE3,
* RECORD 4 TO PIPE4, RECORD 5 TO PIPE1, RECORD 6 TO PIPE2, AND SO ON.
  OUTFIL FNAMES=(PIPE1,PIPE2,PIPE3,PIPE4),SPLIT

* SPLIT THE INCLUDED AND REFORMATTED OUTPUT RECORDS EVENLY BETWEEN
* TAPE1 AND TAPE2.
  OUTFIL FNAMES=(TAPE1,TAPE2),SPLIT,
    INCLUDE=(8,2,ZD,EQ,27),OUTREC=(5X,1,75)
```

Default for SPLIT: None; must be specified.

SPLITBY

Splits the output records n records at a time in rotation among the data sets of this OUTFIL group until all of the output records have been written.

As an example, if SPLITBY=10 is specified for an OUTFIL group with three data sets:

- the first OUTFIL data set in the group will receive records 1-10, 31-40, and so on.
- the second OUTFIL data set in the group will receive records 11-20, 41-50, and so on.
- the third OUTFIL data set in the group will receive records 21-30, 51-60, and so on.

The records are **not** contiguous in each OUTFIL data set.

SPLITBY=1 is equivalent to SPLIT.

The SPLITBY parameter cannot be used with any of the following parameters: IFTRAIL, LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS, and NODETAIL.

n
specifies the number of records to split by. The value for n starts at 1 and is limited to 28 digits (15 significant digits).

Sample Syntax:

```
* WRITE RECORDS 1-5 TO PIPE1, RECORDS 6-10 TO PIPE2, RECORDS 11-15 TO
* PIPE3, RECORDS 16-20 TO PIPE4, RECORDS 21-25 TO PIPE1, RECORDS 26-30
* TO PIPE2, AND SO ON.
   OUTFIL FNAMES=(PIPE1,PIPE2,PIPE3,PIPE4),SPLITBY=5

* SPLIT THE INCLUDED AND REFORMATTED OUTPUT RECORDS 100 AT A TIME
* BETWEEN TAPE1 AND TAPE2.
   OUTFIL FNAMES=(TAPE1,TAPE2),SPLITBY=100,
     INCLUDE=(8,2,ZD,EQ,27),OUTREC=(5X,1,75)
```

Default for SPLITBY: None; must be specified.

SPLIT1R

Splits the output records n records at a time for one rotation among the data sets of this OUTFIL group until all of the output records have been written.

As an example, if SPLIT1R=10 is specified for an input data set with 35 records and an OUTFIL group with three data sets:

- the first OUTFIL data set in the group will receive records 1-10.
- the second OUTFIL data set in the group will receive records 11-20.
- the third OUTFIL data set in the group will receive records 21-35.

The records **are** contiguous in each OUTFIL data set.

The SPLIT1R parameter cannot be used with any of the following parameters: IFTRAIL, LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS, and NODETAIL.

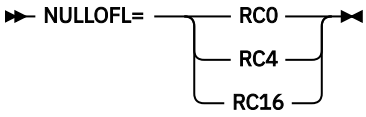
n
specifies the number of records to split by. The value for n starts at 1 and is limited to 28 digits (15 significant digits).

Sample Syntax:

```
  * WRITE RECORDS 1-20 TO PIPE1, RECORDS 21-40 TO PIPE2,
  * RECORDS 41-60 TO PIPE3 AND RECORDS 61-85 TO PIPE4.
     OUTFIL FNAMES=(PIPE1,PIPE2,PIPE3,PIPE4),SPLIT1R=20
  * SPLIT THE INCLUDED AND REFORMATTED OUTPUT RECORDS ONCE
  * CONTIGUOUSLY BETWEEN TAPE1 AND TAPE2.
     OUTFIL FNAMES=(TAPE1,TAPE2),SPLIT1R=100,
      INCLUDE=(8,2,ZD,EQ,27),OUTREC=(5X,1,75)
```

Default for SPLIT1R: None; must be specified.

NULLOFL



specifies the action to be taken by DFSORT when there are no data records for a data set associated with this OUTFIL statement, as indicated by a DATA count of 0 in message ICE227I. OUTFIL report records have no affect on the action taken as a result of this option.

RC0
specifies that DFSORT should issue message ICE174I, set a return code of 0, and continue processing when there are no data records for a data set associated with this OUTFIL statement.

RC4
specifies that DFSORT should issue message ICE174I, set a return code of 4, and continue processing when there are no data records for the a data set associated with this OUTFIL statement.

RC16
specifies that DFSORT should issue message ICE209A, terminate, and give a return code of 16 when there are no data records for a data set associated with this OUTFIL statement.

Default for NULLOFL: The NULLOFL installation default.

Note:

1. The NULLOFL value specified for each OUTFIL statement applies to the data sets associated with that OUTFIL statement. If a NULLOFL value is not specified for an OUTFIL statement, the NULLOFL installation default value applies to the data sets associated with that OUTFIL statement. For example, if the installation default is NULLOFL=RC0 (IBM's shipped default) and the following is specified:

```
OUTFIL FNAMES=OUT1,NULLOFL=RC16,INCLUDE=(5,3,CH,EQ,C'D01')
OUTFIL FNAMES=(OUT2,OUT3),INCLUDE=(5,3,CH,EQ,C'D02')
OUTFIL FNAMES=OUT4,NULLOFL=RC4,SAVE
```

then NULLOFL=RC16 applies to the data set for OUT1, NULLOFL=RC0 (the installation default) applies to the data sets for OUT2 and OUT3, and NULLOFL=RC4 applies to the data set for OUT4.

2. If you receive message ICE174I or ICE209A, a DATA count of 0 in message ICE227I identifies an OUTFIL data set for which there are no data records.
3. The return code of 0 or 4 resulting from NULLOFL=RC0 or NULLOFL=RC4, respectively, can be overridden by a higher return code set for some other reason, including a return code of 16 resulting from NULLOFL=RC16.
4. NULLOUT applies to the SORTOUT data set. NULLOFL applies to OUTFIL data sets

LINES

Specifies the number of lines per page to be used for the reports produced for this OUTFIL group. DFSORT uses ANSI carriage control characters to control page ejects and the placement of the lines in your report, according to your specifications.

The LINES parameter cannot be used with the IFTRAIL parameter.

n
specifies the number of lines per page. The value for n must be between 1 and 255. However, n--or the default for n if LINES is not specified--must be greater than or equal to the number of lines needed for each of the following:

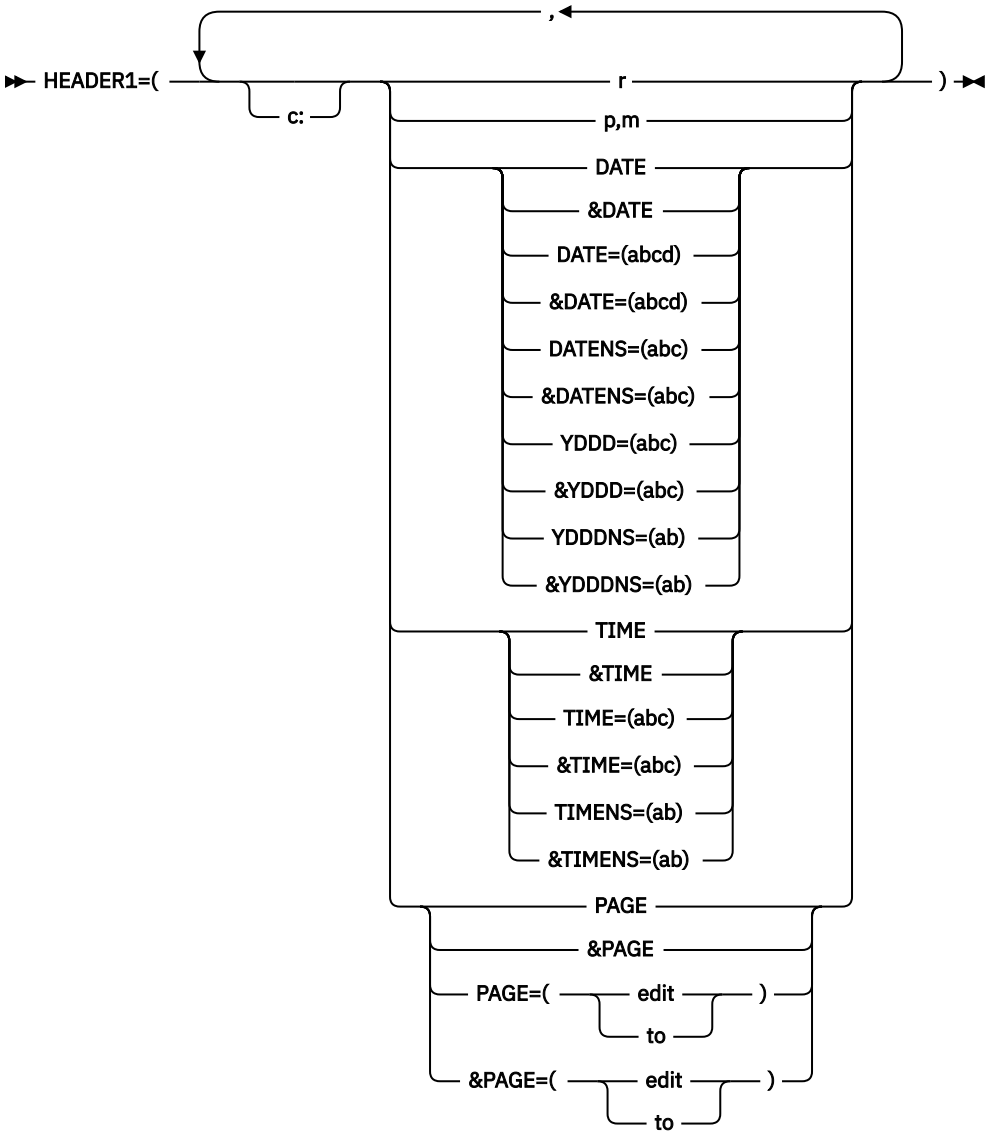
- The HEADER1 lines
- The TRAILER1 lines
- The sum of all lines for HEADER2, TRAILER2, HEADER3s, TRAILER3s, and the data lines and blank lines produced from an input record.

Sample Syntax:

```
OUTFIL FNAMES=RPT1,LINES=50
```

Default for LINES: None; must be specified, unless HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS, or NODETAIL is specified, in which case the default for LINES is 60.

HEADER1



Specifies the report header to be used for the reports produced for this OUTFIL group. The report header appears by itself as the first page of the report. DFSORT uses ANSI carriage control characters to control page ejects and the placement of the lines in your report, according to your specifications. You can use BLKCCH1 to replace the '1' (page eject) for the ANSI carriage control character in the first line of the report header with a blank, thus avoiding forcing a page eject. You can use REMOVECC to remove the ANSI carriage control characters from a report.

You can choose to include any or all of the following report elements in your report header:

- Blanks, character strings, and hexadecimal strings
- Unedited input fields from the first OUTFIL input record
- Current date in a variety of different forms
- Current time in a variety of different forms
- Page number, converted to different numeric formats, or edited to contain signs, decimal points, leading zeros or no leading zeros, and so on.

The HEADER1 parameter cannot be used with the IFTRAIL parameter.

The report header consists of the elements you select, in the order in which you specify them, and in the columns or lines you specify.

c:

specifies the column in which the first position of the associated report element is to appear, relative to the start of the data in the report record. Ignore the RDW (variable-length report records only) and carriage control character when specifying c:. That is, 1: indicates the first byte of the data in the report record for both fixed-length and variable-length report records.

Unused space preceding the specified column is padded with EBCDIC blanks. The following rules apply:

- c must be a number between 1 and 32752.
- c: must be followed by a report element, but must not precede / or n/.
- c must not overlap the previous report element in the report record.
- The colon (:) is treated like the comma (,) or semicolon (;) for continuation to another line.

r

specifies that blanks, a character string, or a hexadecimal string are to appear in the report record, or that a new report record is to be started in the header, with or without intervening blank lines. These report elements can be specified before or after any other report elements. Consecutive report elements can be specified. Permissible values are nX, n'xx...x', nC'xx...x', nX'yy...yy', /.../ and n/.

nX

Blanks. n bytes of EBCDIC blanks (X'40') are to appear in the report record. n can range from 1 to 4095. If n is omitted, 1 is used.

n'xx...x'

Character string. n repetitions of the character string constant ('xx...x') are to appear in the report record. n can range from 1 to 4095. If n is omitted, 1 is used. x can be any EBCDIC character. You can specify 1 to 256 characters.

nC'xx...x' can be used instead of n'xx...x'.

If you want to include a single apostrophe in the character string, you must specify it as two single apostrophes:

```
Required:  O'NEILL     Specify:  'O''NEILL' or C'O''NEILL'
```

nX'yy...yy'

Hexadecimal string. n repetitions of the hexadecimal string constant (X'yy...yy') are to appear in the report record. n can range from 1 to 4095. If n is omitted, 1 is used.

The value yy represents any pair of hexadecimal digits. You can specify from 1 to 256 pairs of hexadecimal digits

/.../ or n/

Blank lines or a new line. A new report record is to be started in the header with or without intervening blank lines. If /.../ or n/ is specified at the beginning or end of the header, n blank lines are to appear in the header. If /.../ or n/ is specified in the middle of the header, n-1 blank lines are to appear in the header (thus, / or 1/ indicates a new line with no intervening blank lines).

Either n/ (for example, 5/) or multiple /'s (for example, /////) can be used. n can range from 1 to 255. If n is omitted, 1 is used.

As an example, if you specify:

```
  OUTFIL HEADER1=(2/,'First line of text',/,
                     'Second line of text',2/,
                     'Third line of text',2/)
```

the report header appears as follows when printed:

```
blank line
blank line
First line of text
Second line of text
blank line
Third line of text
blank line
blank line
```

p,m
specifies that an unedited input field, from the first OUTFIL input record for which a data record appears in the report, is to appear in the report record.

p
specifies the first byte of the input field relative to the beginning of the OUTFIL input record. The first data byte of a fixed-length record has relative position 1. The first data byte of a variable-length record has relative position 5, because the first four bytes are occupied by the RDW. All fields must start on a byte boundary, and no field can extend beyond byte 32752. See "OUTFIL statements notes" on page 360 for special rules concerning variable-length records.

m
specifies the length in bytes of the input field. The value for m must be between 1 and 32752.

DATE
specifies that the current date is to appear in the report record in the form 'mm/dd/yy', where mm represents the month (01-12), dd represents the day (01-31), and yy represents the last two digits of the year (for example, 95).

&DATE
can be used instead of DATE.

DATE=(abcd)
specifies that the current date is to appear in the report record in the form 'adbdc', where a, b, and c indicate the order in which the month, day, and year are to appear and whether the year is to appear as two or four digits, and d is the character to be used to separate the month, day and year.

For a, b, and c, use M to represent the month (01-12), D to represent the day (01-31), Y to represent the last two digits of the year (for example, 05), or 4 to represent the four digits of the year (for example, 2005). M, D, and Y or 4 can each be specified only once. Examples: DATE=(DMY.) would produce a date of the form 'dd.mm.yy', which on March 29, 2005, would appear as '29.03.05'. DATE=(4MD-) would produce a date of the form 'yyyy-mm-dd', which on March 29, 2005, would appear as '2005-03-29'.

a, b, c, and d must be specified.

&DATE=(abcd)
can be used instead of DATE=(abcd).

DATENS=(abc)
specifies that the current date is to appear in the report record in the form 'abc', where a, b and c indicate the order in which the month, day, and year are to appear and whether the year is to appear as two or four digits.

For a, b and c, use M to represent the month (01-12), D to represent the day (01-31), Y to represent the last two digits of the year (for example, 02), or 4 to represent the four digits of the year (for example, 2002). M, D, and Y or 4 can each be specified only once. Examples: DATENS=(DMY) would produce a date of the form 'ddmmyy', which on March 29, 2002, would appear as '290302'. DATENS=(4MD) would produce a date of the form 'yyyymmdd', which on March 29, 2002, would appear as '20020329'.

a, b and c must be specified.

&DATENS=(abc)
can be used instead of DATENS=(abc).

YDDD=(abc)
specifies that the current date is to appear in the report record in the form 'acb', where a and b indicate the order in which the year and day of the year are to appear and whether the year is to appear as two or four digits, and c is the character to be used to separate the year and day of the year.

For a and b, use D to represent the day of the year (001-D366), Y to represent the last two digits of the year (for example, 04), or 4 to represent the four digits of the year (for example, 2004). D, and Y or 4 can each be specified only once. Examples: YDDD=(DY-) would produce a date of the form 'ddd-yy' which on April 7, 2004, would appear as '098-04'. YDDD=(4D/) would produce a date of the form 'yyyy/ddd' which on April 7, 2004, would appear as '2004/098'.

a, b and c must be specified.

&YDDD=(abc)
can be used instead of YDDD=(abc).

YDDDNS=(ab)
specifies that the current date is to appear in the report record in the form 'ab', where a and b indicate the order in which the year and day of the year are to appear and whether the year is to appear as two or four digits.

For a and b, use D to represent the day of the year (001-366), Y to represent the last two digits of the year (for example, 04), or 4 to represent the four digits of the year (for example, 2004). D, and Y or 4 can each be specified only once. Examples: YDDDNS=(DY) would produce a date of the form 'dddyy' which on April 7, 2004, would appear as '09804'. YDDDNS=(4D) would produce a date of the form 'yyyyddd' which on April 7, 2004, would appear as '2004098'.

a and b must be specified.

&YDDDNS=(ab)
can be used instead of YDDDNS=(ab).

TIME
specifies that the current time is to appear in the report record in the form 'hh:mm:ss', where hh represents the hour (00-23), mm represents the minutes (00-59), and ss represents the seconds (00-59).

&TIME
can be used instead of TIME.

TIME=(abc)
specifies that the current time is to appear in the report record in the form 'hhcmmcss' (24-hour time) or 'hhcmmcss xx' (12-hour time).

If ab is 24, the time is to appear in the form 'hhcmmcss' (24-hour time) where hh represents the hour (00-23), mm represents the minutes (00-59), ss represents the seconds (00-59), and c is the character used to separate the hours, minutes, and seconds. Example: TIME=(24.) would produce a time of the form 'hh.mm.ss' which at 08:25:13 pm would appear as '20.25.13'.

If ab is 12, the time is to appear in the form 'hhcmmcss xx' (12-hour time) where hh represents the hour (01-12), mm represents the minutes (00-59), ss represents the seconds (00-59), xx is am or pm, and c is the character used to separate the hours, minutes, and seconds. Example: TIME=(12.) would produce a time of the form 'hh.mm.ss xx' which at 08:25:13 pm would appear as '08.25.13 pm'.

ab and c must be specified.

&TIME=(abc)
can be used instead of TIME=(abc).

TIMENS=(ab)
specifies that the current time is to appear in the report record in the form 'hhmmss' (24-hour time) or 'hhmmss xx' (12-hour time).

If ab is 24, the time is to appear in the form 'hhmmss' (24-hour time) where hh represents the hour (00-23), mm represents the minutes (00-59), and ss represents the seconds (00-59). Example: TIMENS=(24) would produce a time of the form 'hhmmss' which at 08:25:13 pm would appear as '202513'.

If ab is 12, the time is to appear in the form 'hhmmss xx' (12-hour time) where hh represents the hour (01-12), mm represents the minutes (00-59), and ss represents the seconds (00-59). Example: TIMENS=(12) would produce a time of the form 'hhmmss xx' which at 08:25:13 pm would appear as '082513 pm'.

ab must be specified.

&TIMENS=(ab)
can be used instead of TIMENS=(ab).

PAGE
specifies that the page number is to appear in the report record. The page number for the report header appears as ' 1'.

If HEADER1 is specified with PAGE, PAGE for the report header (first page) will be ' 1' and PAGE for the next page (second page) will be ' 2'. If HEADER1 is specified without PAGE, PAGE for the page after the report header (second page) will be ' 1' (typical of a report with a cover sheet).

&PAGE
can be used instead of PAGE.

PAGE=(edit) or PAGE=(to)
same as PAGE except that the 15-digit page number appears edited or converted as specified. See p,m,f,edit under OUTREC for further details on the edit fields you can use. See p,m,f,to under OUTREC for further details on the to fields you can use.

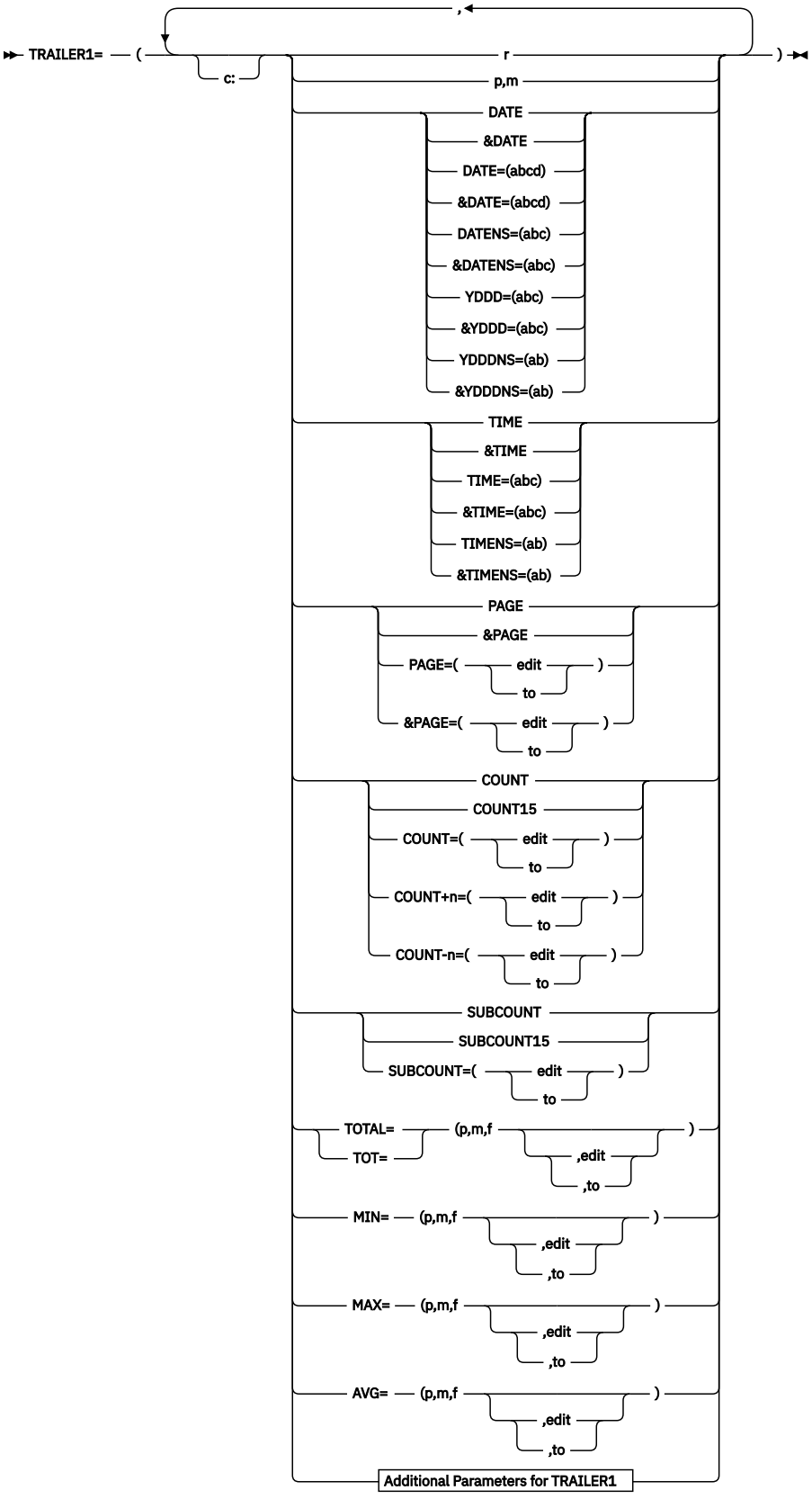
&PAGE=(edit) or &PAGE=(to)
can be used instead of PAGE=(edit) or PAGE=(to).

Sample Syntax:

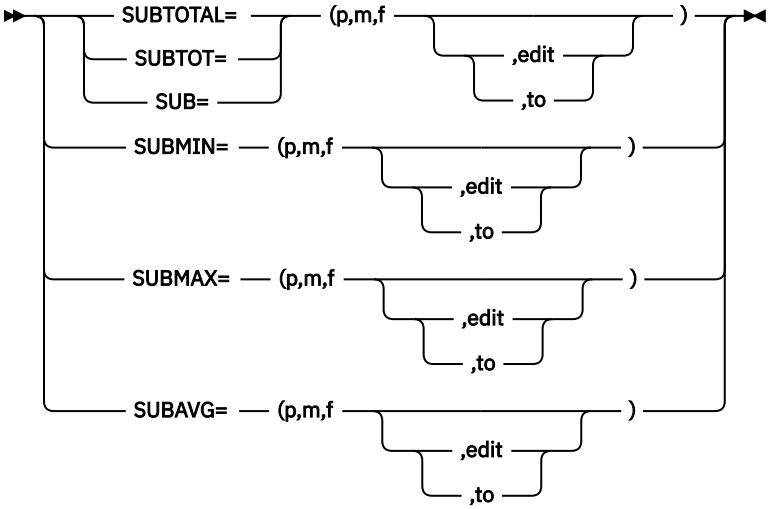
```
OUTFIL FNAMES=(RPT1,RPT2),
   HEADER1=(30:'January Report',4/,
            28:'Prepared on ',DATE,//,
            32:'at ',TIME,//,
            28:'using DFSORT''S OUTFIL',5/,
            10:'Department: ',12,8,50:'Page:',PAGE=(EDIT=(TTT)))
```

Default for HEADER1: None; must be specified.

TRAILER1


TRAILER1=
(
,
c:
r
p,m
DATE
&DATE
DATE=(abcd)
&DATE=(abcd)
DATENS=(abc)
&DATENS=(abc)
YDDD=(abc)
&YDDD=(abc)
YDDDNS=(ab)
&YDDDNS=(ab)
TIME
&TIME
TIME=(abc)
&TIME=(abc)
TIMENS=(ab)
&TIMENS=(ab)
PAGE
&PAGE
PAGE=(
edit
to
)
&PAGE=(
edit
to
)
COUNT
COUNT15
COUNT=(
edit
to
)
COUNT+n=(
edit
to
)
COUNT-n=(
edit
to
)
SUBCOUNT
SUBCOUNT15
SUBCOUNT=(
edit
to
)
TOTAL=
TOT=
(p,m,f
,edit
,to
)
MIN=
(p,m,f
,edit
,to
)
MAX=
(p,m,f
,edit
,to
)
AVG=
(p,m,f
,edit
,to
)
Additional Parameters for TRAILER1
)

Additional Parameters for TRAILER1



Specifies the report trailer to be used for the reports produced for this OUTFIL group. The report trailer appears by itself as the last page of the report. DFSORT uses ANSI carriage control characters to control page ejects and the placement of the lines in your report, according to your specifications. You can use BLKCCT1 to replace the '1' (page eject) for the ANSI carriage control character in the first line of the report trailer with a blank, thus avoiding forcing a page eject. You can use REMOVECC to remove the ANSI carriage control characters from a report.

You can choose to include any or all of the following report elements in your report trailer:

- Blanks, character strings, and hexadecimal strings
- Unedited input fields from the last OUTFIL input record
- Current date in a variety of different forms
- Current time in a variety of different forms
- Page number, converted to different numeric formats, or edited to contain signs, decimal points, leading zeros or no leading zeros, and so on
- Any or all of the following statistics:
  - Count of data records in the report, converted to different numeric formats, or edited to contain signs, decimal points, leading zeros or no leading zeros, and so on. You can add a decimal number to the count before converting or editing it (for example, add 1 to account for writing a trailer record, or add 2 to account for writing a header and trailer record).
  - Total, minimum, maximum, or average for each specified ZD, PD, BI, FI, FL, CSF, FS, UFF, or SFF numeric input field in the data records of the report, converted to different numeric formats, or edited to contain signs, decimal points, leading zeros or no leading zeros, and so on.

The TRAILER1 parameter cannot be used with the IFTRAIL parameter.

The report trailer consists of the elements you select, in the order in which you specify them, and in the columns or lines you specify.

c:
See c: under HEADER1.

r
specifies that blanks, a character string, or a hexadecimal string are to appear in the report record, or that a new report record is to be started in the trailer, with or without intervening blank lines. These report elements can be specified before or after any other report elements. Consecutive report elements can be specified. Permissible values are nX, n'xx...x', nC'xx...x', nX'yy...yy', /.../, and n/.

nX
Blanks. See nX under r for HEADER1.

n'xx...x'
Character string. See n'xx...x' under r for HEADER1. nC'xx...x' can be used instead of n'xx...x'

nX'xx...x'
Hexadecimal string. See nX'xx...x' under r for HEADER1.

/.../ or n/
Blank lines or a new line. A new report record is to be started in the trailer, with or without intervening blank lines. If /.../ or n/ is specified at the beginning or end of the trailer, n blank lines are to appear in the trailer. If /.../ or n/ is specified in the middle of the trailer, n-1 blank lines are to appear in the trailer (thus, / or 1/ indicates a new line with no intervening blank lines).

Either n/ (for example, 5/) or multiple /'s (for example, /////) can be used. n can range from 1 to 255. If n is omitted, 1 is used.

p,m
specifies that an unedited input field, from the last OUTFIL input record for which a data record appears in the report, is to appear in the report record.

p
See p under HEADER1.

m
See m under HEADER1.

DATE
See DATE under HEADER1.

&DATE
can be used instead of DATE.

DATE=(abcd)
See DATE=(abcd) under HEADER1.

&DATE=(abcd)
can be used instead of DATE=(abcd).

DATENS=(abc)
See DATENS=(abc) under HEADER1.

&DATENS=(abc)
can be used instead of DATENS=(abc).

YDDD=(abc)
See YDDD=(abc) under HEADER1.

&YDDD=(abc)
can be used instead of YDDD=(abc).

YDDDNS=(ab)
See YDDDNS=(ab) under HEADER1.

&YDDDNS=(ab)
can be used instead of YDDDNS=(ab).

TIME
See TIME under HEADER1.

&TIME
can be used instead of TIME.

TIME=(abc)
See TIME=(abc) under HEADER1.

&TIME=(abc)
can be used instead of TIME=(abc).

TIMENS=(ab)
See TIMENS=(ab) under HEADER1.

&TIMENS=(ab)
can be used instead of TIMENS=(ab).

PAGE
specifies that the current page number is to appear in the report record. The page number for the trailer appears as 6 digits, right-justified, with leading zeros suppressed. For example, if the page is numbered 12, it appears as '   12'.

&PAGE
can be used instead of PAGE.

PAGE=(edit) or PAGE=(to)
same as PAGE except that the 15-digit page number appears edited or converted as specified. See p,m,f,edit under OUTREC for further details on the edit fields you can use. See p,m,f,to under OUTREC for further details on the to fields you can use.

&PAGE=(edit) or &PAGE=(to)
can be used instead of PAGE=(edit) or PAGE=(to).

COUNT
specifies that the count of data records in the report is to appear in the report record as 8 digits, right-justified, with leading zeros suppressed. For example, if there are 6810 input records in the report, the count appears as '   6810'.

If slash (/) is used in OUTREC or BUILD to produce multiple data records, COUNT only counts the number of data records processed as input to OUTREC or BUILD. For example, if OUTREC processes 3 input records and creates 2 output records for each input record, the count is 3, not 6

COUNT15
same as COUNT except that the count appears as 15 digits.

COUNT=(edit) or COUNT=(to)
same as COUNT except that the 15–digit count appears edited or converted as specified. See p,m,f,edit under OUTREC for further details on the edit fields you can use. See p,m,f,to under OUTREC for further details on the to fields you can use. For example, if there are 6810 input records, COUNT=(M11,LENGTH=6) produces a count of '006810'.

COUNT+n=(edit) or COUNT+n=(to)
same as COUNT=(edit) or COUNT=(to) except that n is added to the 15-digit count before it is edited or converted. n can be 1 to 3 decimal digits. For example, if there are 6810 input records, COUNT+2=(M11,LENGTH=6) produces a count of '006812'. One important use for this parameter is to add 1 for the TRAILER1 record to the count of data records.

COUNT-n=(edit) or COUNT-n=(to)
same as COUNT=(edit) or COUNT=(to) except that n is subtracted from the 15-digit count before it is edited or converted. n can be 1 to 3 decimal digits. For example, if there are 6810 input records, COUNT-1=(M11,LENGTH=6) produces a count of '006809'. One important use for this parameter is to subtract 1 for a header record from the count of all records.

SUBCOUNT
specifies that the running count of input records in the report is to appear in the report record as 8 digits, right-justified, with leading zeros suppressed.

For TRAILER1, the running count is the same as the count, so SUBCOUNT produces the same value as COUNT.

If slash (/) is used in OUTREC or BUILD to produce multiple data records, SUBCOUNT counts only the number of data records processed as input to OUTREC or BUILD. For example, if OUTREC processes 3 input records and creates 2 output records for each input record, the running count is 3, not 6.

SUBCOUNT15
same as SUBCOUNT except that the running count appears as 15 digits.

SUBCOUNT=(edit) or SUBCOUNT=(to)
same as SUBCOUNT except that the 15–digit running count appears edited or converted as specified. See p,m,f,edit under OUTREC for further details on the edit fields you can use. See p,m,f,to under OUTREC for further details on the to fields you can use.

TOTAL
specifies that an edited or converted total, for the values of a numeric input field in all data records of the report, is to appear in the report record.

TOT can be used instead of TOTAL.

p,m,f,edit or p,m,f,to
specifies the numeric input field for which the total is to be produced and how the output field (that is, the total) is to be edited or converted.

See p,m,f,edit under OUTREC for further details. However, note that PD0, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3, and TM4 are not allowed for TOTAL and that for TOTAL, the default number of digits (d) used for editing or conversion is as follows:

Table 63. Digits for TOTAL Fields

Format (f)	Length (m)	Digits (d)
ZD	1-15	15
ZD	16-31	31
PD	1-8	15
PD	9-16	31
BI	1-4	10
BI	5-8	20
FI	1-4	10
FI	5-8	20
FL	4 or 8	20
CSF or FS	1-15	15
CSF or FS	16-32	31
UFF	1-15	15
UFF	16-44	31
SFF	1-15	15
SFF	16-44	31

If EDIT or EDxy is specified, the number of digits in the pattern (I's and T's) is used.

MIN
specifies that an edited or converted minimum, for the values of a numeric input field in all data records of the report, is to appear in the report record.

p,m,f,edit or p,m,f,to
specifies the numeric input field for which the minimum is to be produced and how the output field (that is, the minimum) is to be edited or converted.

See p,m,f,edit or p,m,f,to under OUTREC for further details. However, note that PD0, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3 and TM4 are not allowed for MIN.

MAX
specifies that an edited or converted maximum, for the values of a numeric input field in all data records of the report, is to appear in the report record.

p,m,f,edit or p,m,f,to
specifies the numeric input field for which the maximum is to be produced and how the output field (that is, the maximum) is to be edited or converted.

See p,m,f,edit or p,m,f,to under OUTREC for further details. However, note that PD0, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3 and TM4 are not allowed for MAX.

AVG
specifies that an edited or converted average, for the values of a numeric input field in all data records of the report, is to appear in the report record. The average (or mean) is calculated by dividing the total by the count and rounding down to the nearest integer. For example:

```
+2305 / 152 = +15
-2305 / 152 = -15
```

p,m,f,edit or p,m,f,to
specifies the numeric input field for which the average is to be produced and how the output field (that is, the average) is to be edited or converted.

See p,m,f,edit or p,m,f,to under OUTREC for further details. However, note that PD0, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3 and TM4 are not allowed for AVG.

SUBTOTAL
specifies that an edited or converted running total, for the values of a numeric input field in all data records of the report, is to appear in the report record.

SUBTOT or SUB can be used instead of SUBTOTAL.

For TRAILER1, the running total is the same as the total, so SUBTOTAL produces the same value as TOTAL.

p,m,f,edit or p,m,f,to
specifies the numeric input field for which the running total is to be produced and how the output field (that is, the running total) is to be edited or converted.

See p,m,f,edit or p,m,f,to under TOTAL for further details.

SUBMIN
specifies that an edited or converted running minimum, for the values of a numeric input field in all data records of the report, is to appear in the report record.

For TRAILER1, the running minimum is the same as the minimum, so SUBMIN produces the same value as MIN.

p,m,f,edit or p,m,f,to
specifies the numeric input field for which the running minimum is to be produced and how the output field (that is, the running minimum) is to be edited or converted.

See p,m,f,edit or p,m,f,to under OUTREC for further details. However, note that PD0, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3 and TM4 are not allowed for SUBMIN.

SUBMAX
specifies that an edited or converted running maximum, for the values of a numeric input field in all data records of the report, is to appear in the report record.

For TRAILER1, the running maximum is the same as the maximum, so SUBMAX produces the same value as MAX.

p,m,f,edit or p,m,f,to
specifies the numeric input field for which the running maximum is to be produced and how the output field (that is, the running maximum) is to be edited or converted.

See p,m,f,edit or p,m,f,to under OUTREC for further details. However, note that PD0, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3 and TM4 are not allowed for SUBMAX.

SUBAVG
specifies that an edited or converted running average, for the values of a numeric input field in all data records of the report, is to appear in the report record.

For TRAILER1, the running average is the same as the average, so SUBAVG produces the same value as AVG.

p,m,f,edit or p,m,f,to
specifies the numeric input field for which the running average is to be produced and how the output field (that is, the running average) is to be edited or converted.

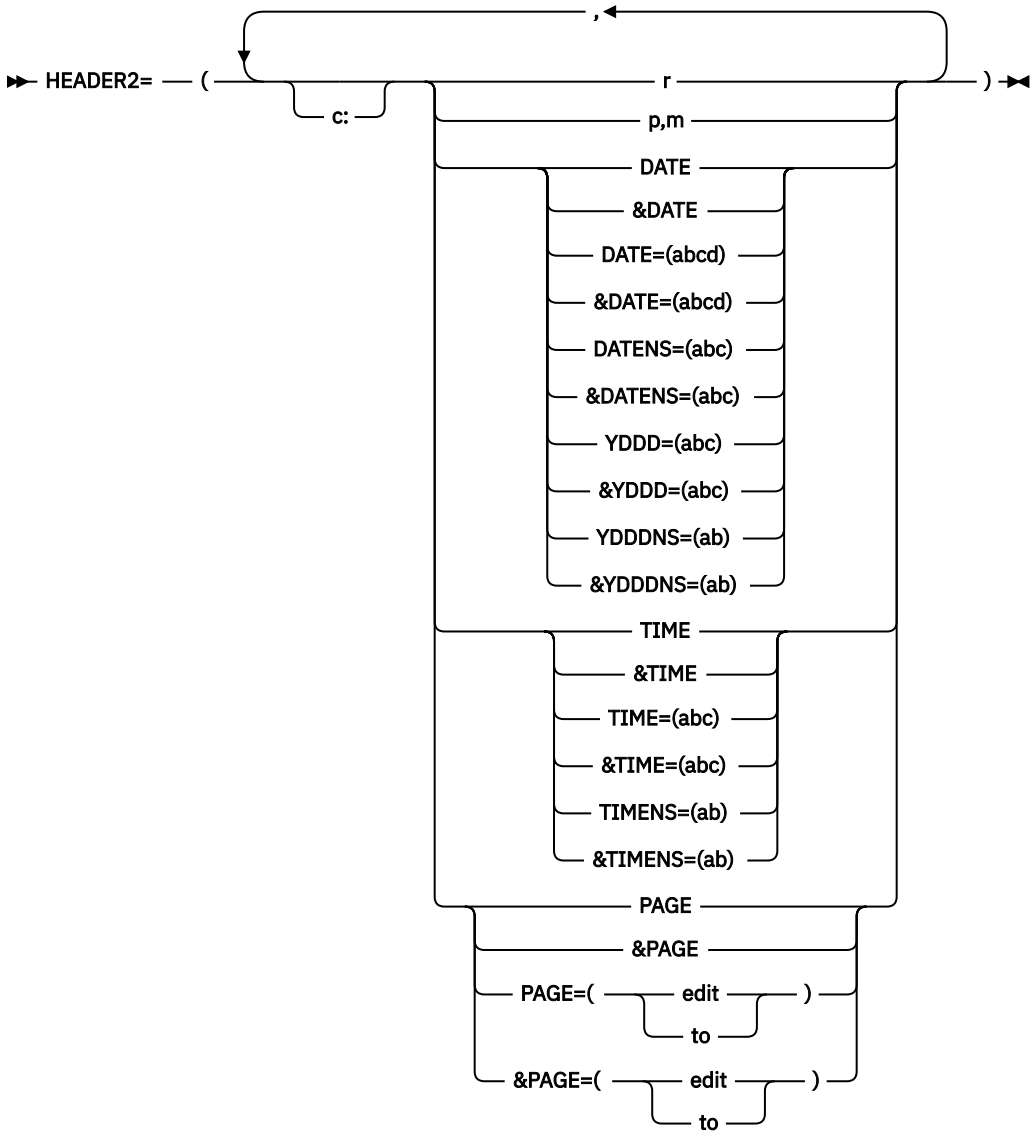
See p,m,f,edit or p,m,f,to under OUTREC for further details. However, note that PD0, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3 and TM4 are not allowed for SUBAVG.

Sample Syntax:

```
OUTFIL FNAMES=RPT1,
  TRAILER1=(5/,
   10:'Summary of Report for Division Revenues',3/,
   10:'Number of divisions reporting: ',COUNT,2/,
   10:'Total revenue: ',TOTAL=(25,5,PD,M5),2/,
   10:'Lowest revenue: ',MIN=(25,5,PD,M5),2/,
   10:'Highest revenue: ',MAX=(25,5,PD,M5),2/,
   10:'Average revenue: ',AVG=(25,5,PD,M5))

OUTFIL FNAMES=RPT2,BLKCCT1,
       TRAILER1=(/,10:'Grand Total: ',TOTAL=(25,5,PD,M5))
```

Default for TRAILER1: None; must be specified.

HEADER2



Specifies the page header to be used for the reports produced for this OUTFIL group. The page header appears at the top of each page of the report, except for the report header page (if any) and report trailer page (if any). DFSORT uses ANSI carriage control characters to control page ejects and the placement of the lines in your report, according to your specifications. You can use BLKCCH2 to replace the '1' (page eject) for the ANSI carriage control character in the first line of the first page header with a blank, thus avoiding forcing a page eject. You can use REMOVECC to remove the ANSI carriage control characters from a report.

You can choose to include any or all of the following report elements in your page header:

- Blanks, character strings, and hexadecimal strings
- Unedited input fields from the first OUTFIL input record for which a data record appears on the page
- Current date in a variety of different forms
- Current time in a variety of different forms
- Page number, converted to different numeric formats, or edited to contain signs, decimal points, leading zeros or no leading zeros, and so on.

The HEADER2 parameter cannot be used with the IFTRAIL parameter.

The page header consists of the elements you select, in the order in which you specify them, and in the columns or lines you specify.

c:
See c: under HEADER1.

r
See r under HEADER1.

p,m
specifies that an unedited input field, from the first OUTFIL input record for which a data record appears on the page, is to appear in the report record. See p,m under HEADER1 for further details.

DATE
See DATE under HEADER1.

&DATE
can be used instead of DATE.

DATE=(abcd)
See DATE=(abcd) under HEADER1.

&DATE=(abcd)
can be used instead of DATE=(abcd).

DATENS=(abc)
See DATENS=(abc) under HEADER1.

&DATENS=(abc)
can be used instead of DATENS=(abc).

YDDD=(abc)
See YDDD=(abc) under HEADER1.

&YDDD=(abc)
can be used instead of YDDD=(abc).

YDDDNS=(ab)
See YDDDNS=(ab) under HEADER1.

&YDDDNS=(ab)
can be used instead of YDDDNS=(ab).

TIME
See TIME under HEADER1.

&TIME
can be used instead of TIME.

TIME=(abc)
See TIME=(abc) under HEADER1.

&TIME=(abc)
can be used instead of TIME=(abc).

TIMENS=(ab)
See TIMENS=(ab) under HEADER1.

&TIMENS=(ab)
can be used instead of TIMENS=(ab).

PAGE
specifies that the current page number is to appear in the OUTFIL report record. The page number for the header appears as 6 digits, right-justified, with leading zeros suppressed. For example, if the page is numbered 3, it appears as '   3'.

If HEADER1 is specified with PAGE and HEADER2 is specified with PAGE, the page number for the first page header will be '   2'. If HEADER1 is not specified or is specified without PAGE and HEADER2 is specified with PAGE, the page number for the first page header will be '   1'.

&PAGE
can be used instead of PAGE.

PAGE=(edit) or PAGE=(to)
same as PAGE except that the 15-digit page number appears edited or converted as specified. See p,m,f,edit under OUTREC for further details on the edit fields you can use. See p,m,f,to under OUTREC for further details on the to fields you can use.

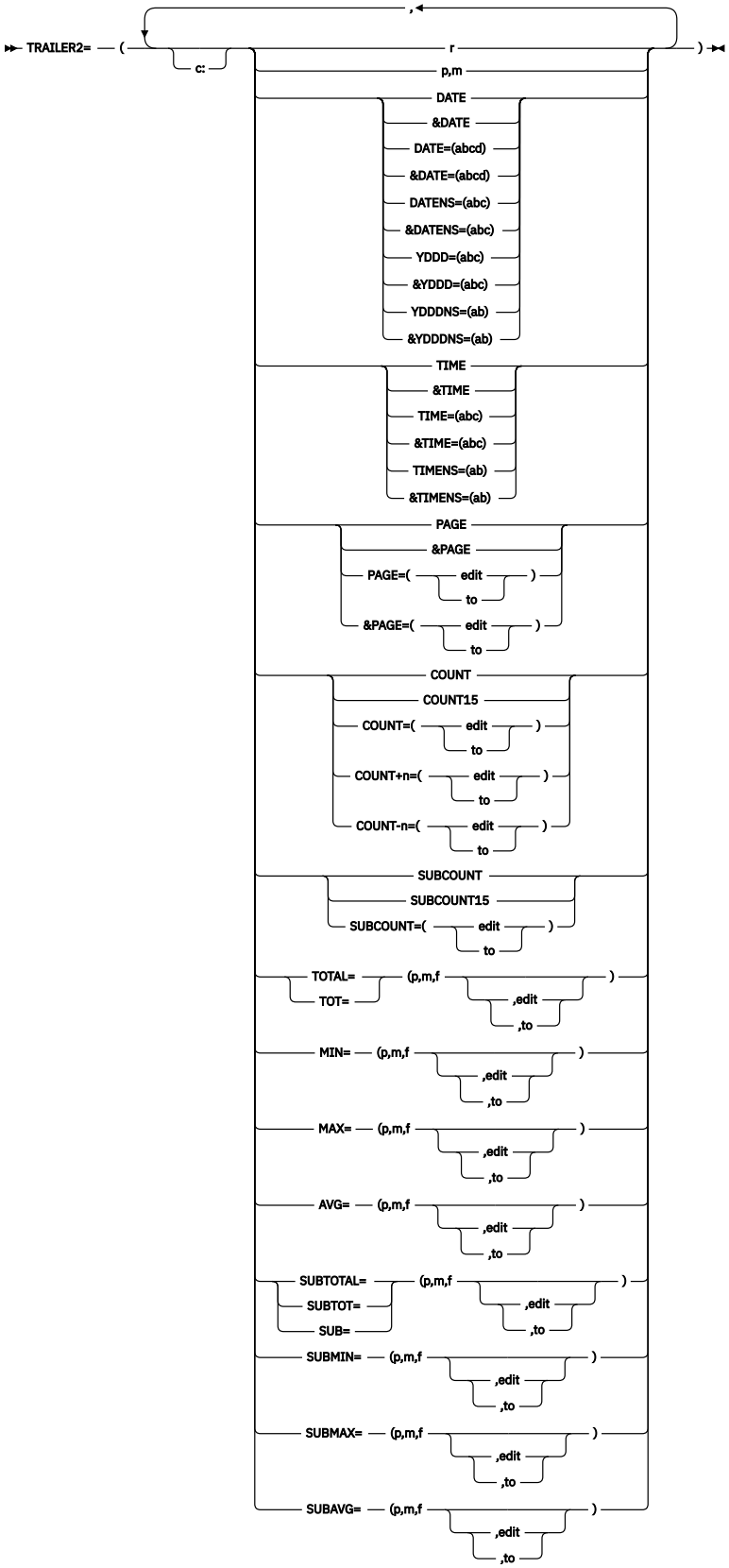
&PAGE=(edit) or &PAGE=(to)
can be used instead of PAGE=(edit) or PAGE=(to).

Sample Syntax:

```
OUTFIL FNAMES=STATUS,
  HEADER2=(5:'Page ',PAGE,' of Status Report for ',DATE=(MD4/),
           ' at ',TIME=(12:),2/,
           10:'Item ',20:'Status      ',35:'Count',/,
           10:'-----',20:'------------',35:'-----'),
   OUTREC=(10:6,5,
           20:14,1,CHANGE=(12,
                           C'S',C'Ship',
                           C'H',C'Hold',
                           C'T',C'Transfer'),
                   NOMATCH=(C'*Check Code*'),
           36:39,4,ZD,M10,
           132:X)
```

Default for HEADER2: None; must be specified.

TRAILER2



Specifies the page trailer to be used for the reports produced for this OUTFIL group. The page trailer appears at the very bottom of each page of the report (as specified or defaulted by the LINES value), except for the report header page (if any) and report trailer page (if any). DFSORT uses ANSI carriage

control characters to control page ejects and the placement of the lines in your report, according to your specifications.

You can choose to include any or all of the following report elements in your page trailer:

- Blanks, character strings, and hexadecimal strings
- Unedited input fields from the last OUTFIL input record for which a data record appears on the page
- Current date in a variety of different forms
- Current time in a variety of different forms
- Page number, converted to different numeric formats, or edited to contain signs, decimal points, leading zeros or no leading zeros, and so on
- Any or all of the following statistics:
  - Count of data records on the page, converted to different numeric formats, or edited to contain signs, decimal points, leading zeros or no leading zeros, and so on. You can add a decimal number to the count before converting or editing it (for example, add 1 to account for writing a trailer record, or add 2 to account for writing a header and trailer record).
  - Total, minimum, maximum, or average for each specified ZD, PD, BI, FI, FL, CSF, FS, UFF, or SFF numeric input field in the data records on the page, converted to different numeric formats, or edited to contain signs, decimal points, leading zeros or no leading zeros, and so on.
  - Running total, minimum, maximum, or average for each specified ZD, PD, BI, FI, FL, CSF, FS, UFF, or SFF numeric input field in the data records up to this point, converted to different numeric formats, or edited to contain signs, decimal points, leading zeros or no leading zeros, and so on.

The TRAILER2 parameter cannot be used with the IFTRAIL parameter.

The page trailer consists of the elements you select, in the order in which you specify them, and in the columns or lines you specify.

c:
See c: under HEADER1.

r
See r under TRAILER1.

p,m
specifies that an unedited input field, from the last OUTFIL input record for which a data record appears on the page, is to appear in the report record. See p,m under TRAILER1 for further details.

DATE
See DATE under HEADER1.

&DATE
can be used instead of DATE.

DATE=(abcd)
See DATE=(abcd) under HEADER1.

&DATE=(abcd)
can be used instead of DATE=(abcd).

DATENS=(abc)
See DATENS=(abc) under HEADER1.

&DATENS=(abc)
can be used instead of DATENS=(abc).

YDDD=(abc)
See YDDD=(abc) under HEADER1.

&YDDD=(abc)
can be used instead of YDDD=(abc).

YDDDNS=(ab)
See YDDDNS=(ab) under HEADER1.

&YDDDNS=(ab)
can be used instead of YDDDNS=(ab).

TIME
See TIME under HEADER1.

&TIME
can be used instead of TIME.

TIME=(abc)
See TIME=(abc) under HEADER1.

&TIME=(abc)
can be used instead of TIME=(abc).

TIMENS=(ab)
See TIMENS=(ab) under HEADER1.

&TIMENS=(ab)
can be used instead of TIMENS=(ab).

PAGE
See PAGE under TRAILER1.

&PAGE
can be used instead of PAGE.

PAGE=(edit) or PAGE=(to)
See PAGE=(edit) or PAGE=(to) under TRAILER1.

&PAGE=(edit) or &PAGE=(to)
can be used instead of PAGE=(edit) or PAGE=(to).

COUNT
specifies that the count of data records on the page is to appear in the report record as 8 digits, right-justified, with leading zeros suppressed. For example, if page 1 has 40 input records, page 2 has 40 input records, and page 3 has 26 input records, COUNT will show ' 40' for page 1, ' 40' for page 2, and ' 26' for page 3.

If slash (/) is used in OUTREC or BUILD to produce multiple data records, COUNT counts only the number of data records processed as input to OUTREC or BUILD. For example, if OUTREC processes 3 input records and creates 2 output records for each input record, the count is 3, not 6.

COUNT15
same as COUNT except that the count appears as 15 digits.

COUNT=(edit) or COUNT=(to)
same as COUNT except that the 15–digit count appears edited or converted as specified. See p,m,f,edit under OUTREC for further details on the edit fields you can use. See p,m,f,to under OUTREC for further details on the to fields you can use.

COUNT+n=(edit) or COUNT+n=(to)
same as COUNT=(edit) or COUNT=(to) except that n is added to the 15-digit count before it is edited or converted. n can be 1 to 3 decimal digits.

COUNT-n=(edit) or COUNT-n=(to)
same as COUNT=(edit) or COUNT=(to) except that n is subtracted from the 15-digit count before it is edited or converted. n can be 1 to 3 decimal digits.

SUBCOUNT
specifies that the count of input records up to this point in the report is to appear in the report record as 8 digits, right-justified, with leading zeros suppressed. The running count accumulates the count for all pages up to and including the current page. For example, if page 1 has 40 input records, page 2 has 40 input records, and page 3 has 26 input records, SUBCOUNT will show ' 40' for page 1, ' 80' for page 2, and ' 106' for page 3.

If slash (/) is used in OUTREC or BUILD to produce multiple data records, SUBCOUNT counts only the number of data records processed as input to OUTREC or BUILD. For example, if OUTREC

processes 3 input records and creates 2 output records for each input record, the running count is 3, not 6.

SUBCOUNT15
same as SUBCOUNT except that the running count appears as 15 digits.

SUBCOUNT=(edit) or SUBCOUNT=(to)
same as SUBCOUNT except that the 15–digit running count appears edited or converted as specified. See p,m,f,edit under OUTREC for further details on the edit fields you can use. See p,m,f,to under OUTREC for further details on the to fields you can use.

TOTAL
specifies that an edited or converted total, for the values of a numeric input field in the data records on the page, is to appear in the report record.

TOT can be used instead of TOTAL.

p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,to under TOTAL for TRAILER1.

MIN
specifies that an edited or converted minimum, for the values of a numeric input field in the data records on the page, is to appear in the report record.

p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,to under MIN for TRAILER1.

MAX
specifies that an edited or converted maximum, for the values of a numeric input field in the data records on the page, is to appear in the report record.

p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,to under MAX for TRAILER1.

AVG
specifies that an edited or converted average, for the values of a numeric input field in the data records on the page, is to appear in the report record.

p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,to under AVG for TRAILER1.

SUBTOTAL
specifies that an edited or converted running total, for the values of a numeric input field in the data records up to this point in the report, is to appear in the report record. The running total accumulates the total for all pages up to and including the current page. For example, if the total for a selected numeric field is +200 for page 1, -250 for page 2, and +90 for page 3, SUBTOTAL will be +200 for page 1, -50 for page 2, and +40 for page 3.

SUBTOT or SUB can be used instead of SUBTOTAL.

p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,to under SUBTOTAL for TRAILER1.

SUBMIN
specifies that an edited or converted running minimum, for the values of a numeric input field in the data records up to this point in the report, is to appear in the report record. The running minimum selects the minimum from all pages up to and including the current page. For example, if the minimum for a selected numeric field is +200 for page 1, -250 for page 2, and +90 for page 3, SUBMIN will be +200 for page 1, -250 for page 2, and -250 for page 3.

p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,to under SUBMIN for TRAILER1.

SUBMAX
specifies that an edited or converted running maximum, for the values of a numeric input field in the data records up to this point in the report, is to appear in the report record. The running maximum selects the maximum from all pages up to and including the current page. For example,

if the maximum for a selected numeric field is -100 for page 1, +250 for page 2, and +90 for page 3, SUBMAX will be -100 for page 1, +250 for page 2, and +250 for page 3.

p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,to under SUBMAX for TRAILER1.

SUBAVG
specifies that an edited or converted running average, for the values of a numeric input field in the data records up to this point in the report, is to appear in the report record. The running average computes the average for all pages up to and including the current page. For example, if the count of data records and total for a selected numeric field are 60 and +2205 for page 1, respectively, 60 and -6252 for page 2, respectively, and 23 and -320 for page 3, respectively, SUBAVG will be +36 for page 1, -33 for page 2, and -30 for page 3.

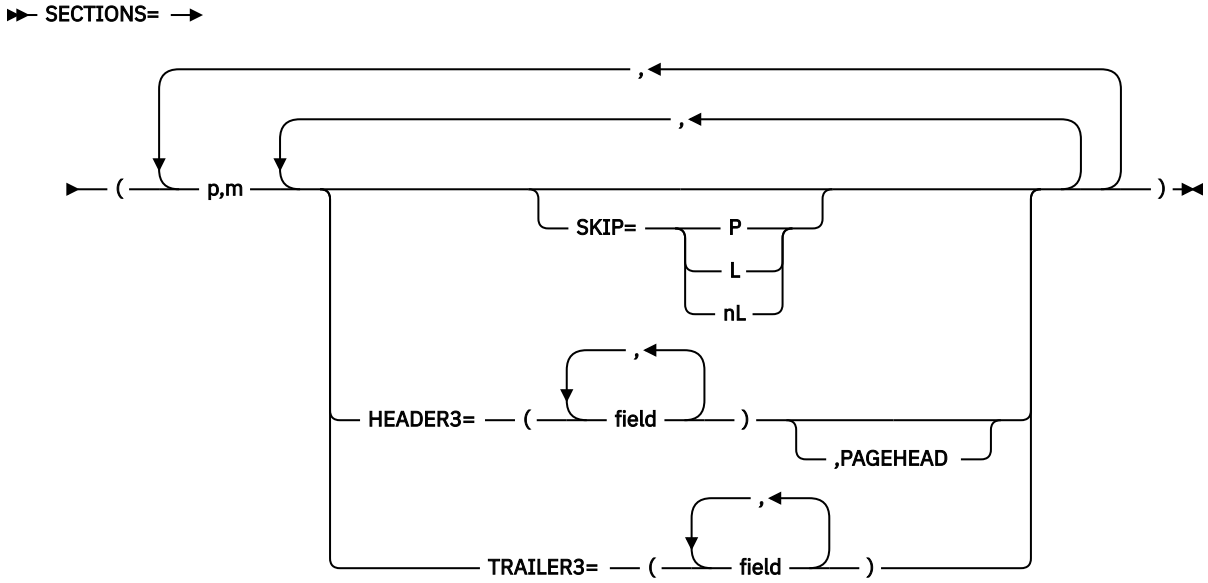
p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,to under SUBAVG for TRAILER1.

Sample Syntax:

```
OUTFIL FNAMES=STATS,
   STARTREC=3,
    OUTREC=(20:23,3,PD,M16,
            30:40,3,PD,M16,
            80:X),
   TRAILER2=(/,2:'Average on page:',
             20:AVG=(23,3,PD,M16),
             30:AVG=(40,3,PD,M16),/,
             2:'Average so far:',
             20:SUBAVG=(23,3,PD,M16),
             30:SUBAVG=(40,3,PD,M16))
```

Default for TRAILER2: None; must be specified.

SECTIONS



Specifies the section break processing to be used for the reports produced for this OUTFIL group. A section break field divides the report into sets of sequential OUTFIL input records with the same binary value for that field, which result in corresponding sets of data records (that is, sections) in the report. A break is said to occur when the binary value changes. Of course, because a break can occur in any record, the data records of a section can be split across pages in your report.

For each section break field you specify, you can choose to include any or all of the following:

- A page eject between sections.

- Zero, one or more blank lines to appear between sections on the same page.
- A section header to appear before the first data record of each section and optionally, at the top of each page. When a page header and section header are both to appear at the top of a page, the section header will follow the page header.
- A section trailer to appear after the last data record of each section. When a page trailer and section trailer are both to appear at the bottom of a page, the page trailer will follow the section trailer.

The SECTIONS parameter cannot be used with the IFTRAIL parameter.

DFSORT uses ANSI carriage control characters to control page ejects and the placement of the lines in your report, according to your specifications.

If multiple section break fields are used, they are processed in first-to-last order, in the same way they would be sorted by these fields. In fact, the input data set is generally sorted by the section break fields, to group the records with the same section break values together for the report. This sorting can be done by the same application that produces the report or by a previous application.

A break in section break field 1 results in a break in section break fields 2 through n. A break in section break 2 results in a break in section break fields 3 through n, and so on. The section headers appear before each section in first-to-last order, whereas the section trailers appear in last-to-first order. For example, if section break fields represented by B1 with header H3A and trailer T3A, B2 with header H3B and trailer T3B, and B3 with header H3C and trailer T3C are specified in order, the following can appear:

```
H3A (header for B1=1 section)
  H3B (header for B2=1 section)
    H3C (header for B3=1 section)
      data records for B1=1, B2=1, B3=1 (new B1, B2, and B3 section)
    T3C (trailer for B3=1 section)
    H3C (header for B3=2 section)
      data records for B1=1, B2=1, B3=2 (new B3 section)
    T3C (trailer for B3=2 section)
  T3B (trailer for B2=1 section)
  H3B (header for B2=2 section)
    H3C (header for B3=1 section)
      data records for B1=1, B2=2, B3=1 (new B2 and B3 section)
    T3C (trailer for B3=1 section)
  T3B (trailer for B2=2 section)
T3A (trailer for B1=1 section)
H3A (header for B1=2 section)
  H3B (header for B2=2 section)
    H3C (header for B3=0 section)
      data records for B1=2, B2=2, B3=0 (new B1, B2, and B3 section)
    T3C (trailer for B3=0 section)
    H3C (header for B3=1 section)
      data records for B1=2, B2=2, B3=1 (new B3 section)
    T3C (trailer for B3=1 section)
  T3B (trailer for B2=2 section)
T3A (trailer for B1=2 section)
```

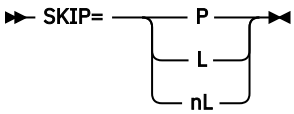
p,m
specifies a section break field in the OUTFIL input records to be used to divide the report into sections. Each set of sequential OUTFIL input records, with the same binary value for the section break field, results in a corresponding set of data records. Each such set of data records is treated as a section in the report. A break is said to occur when the binary value changes.

p
specifies the first byte of the input field relative to the beginning of the OUTFIL input record. The first data byte of a fixed-length record has relative position 1. The first data byte of a variable-length record has relative position 5, because the first four bytes are occupied by the RDW. All fields must start on a byte boundary, and no field can extend beyond byte 32752. See “OUTFIL statements notes” on page 360 for special rules concerning variable-length records.

m
specifies the length in bytes of the input field. The value for m must be between 1 and 256.

You can specify any, all or none of SKIP, HEADER3 and TRAILER3 after p,m. If you do not specify SKIP, HEADER3 or TRAILER3 after p,m, SKIP=0L is used as the default.

SKIP



Specifies, for reports produced for this OUTFIL group, that either:

- Each section for the associated section break field is to appear on a new page, or
- Zero, one or more blank lines to appear after each section associated with this section break field, when it is followed by another section on the same page.

Thus, you can use SKIP to specify how sections will be separated from each other.

P
specifies that each section is to appear on a new page.

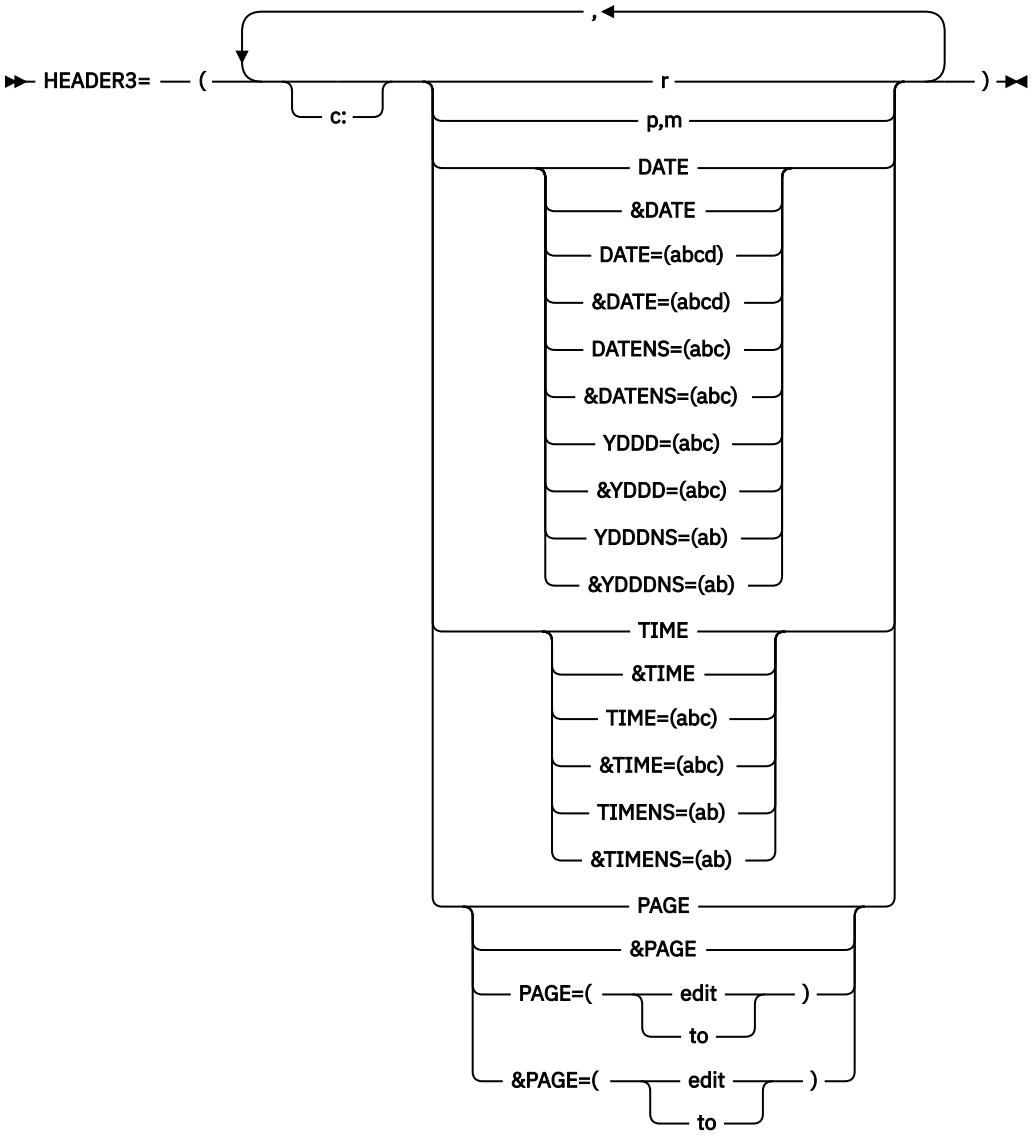
L
specifies that one blank line is to appear between sections on the same page. L is the same as 1L.

nL
specifies that n blank lines are to appear between sections on the same page. You can specify from 0 to 255 for n.

Sample Syntax:

```
OUTFIL FNAMES=(PRINT,TAPE),
   SECTIONS=(10,20,SKIP=P,
             42,10,SKIP=3L)
```

HEADER3



Specifies the section header to be used with the associated section break field for the reports produced for this OUTFIL group. The section header appears before the first data record of each section. DFSORT uses ANSI carriage control characters to control page ejects and the placement of the lines in your report, according to your specifications.

You can choose to include any or all of the following report elements in your section header:

- Blanks, character strings, and hexadecimal strings
- Unedited input fields from the first OUTFIL input record for which a data record appears in the section
- Current date in a variety of different forms
- Current time in a variety of different forms
- Page number, converted to different numeric formats, or edited to contain signs, decimal points, leading zeros or no leading zeros, and so on.

The section header consists of the elements you select, in the order in which you specify them, and in the columns or lines you specify.

c:
See c: under HEADER1.

r
See r under HEADER1.

p,m
specifies that an unedited input field, from the first OUTFIL input record for which a data record appears in the section, is to appear in the report record. See p,m under HEADER1 for further details.

DATE
See DATE under HEADER1.

&DATE
can be used instead of DATE.

DATE=(abcd)
See DATE=(abcd) under HEADER1.

&DATE=(abcd)
can be used instead of DATE=(abcd).

DATENS=(abc)
See DATENS=(abc) under HEADER1.

&DATENS=(abc)
can be used instead of DATENS=(abc).

YDDD=(abc)
See YDDD=(abc) under HEADER1.

&YDDD=(abc)
can be used instead of YDDD=(abc).

YDDDNS=(ab)
See YDDDNS=(ab) under HEADER1.

&YDDDNS=(ab)
can be used instead of YDDD=(ab).

TIME
See TIME under HEADER1.

&TIME
can be used instead of TIME.

TIME=(abc)
See TIME=(abc) under HEADER1.

&TIME=(abc)
can be used instead of TIME=(abc).

TIMENS=(ab)
See TIMENS=(ab) under HEADER1.

&TIMENS=(ab)
can be used instead of TIMENS=(ab).

PAGE
specifies that the current page number is to appear in the OUTFIL report record. The page number for the header appears as 6 digits, right-justified, with leading zeros suppressed. For example, if the page is numbered 3, it appears as '   3'.

&PAGE
can be used instead of PAGE.

PAGE=(edit) or PAGE=(to)
same as PAGE except that the 15-digit page number appears edited or converted as specified. See p,m,f,edit under OUTREC for further details on the edit fields you can use. See p,m,f,to under OUTREC for further details on the to fields you can use.

&PAGE=(edit) or &PAGE=(to)
can be used instead of PAGE=(edit) or PAGE=(to).

Sample Syntax:

```
OUTFIL FNAMES=STATUS1,
         HEADER2=(10:'Status Report for all departments',5X,
                        '- ',&PAGE,' -'),
         SECTIONS=(10,8,
          HEADER3=(2/,10:'Report for department ',10,8,' on ',&DATE,2/,
                    10:'   Number',25:'Average Time',/,
                    10:'Completed',25:'   (in days)',/,
                    10:'---------',25:'------------')),
            OUTREC=(10:21,5,ZD,M10,LENGTH=9,
                       25:38,4,ZD,EDIT=(III.T),LENGTH=12,
                       132:X)
```

PAGEHEAD

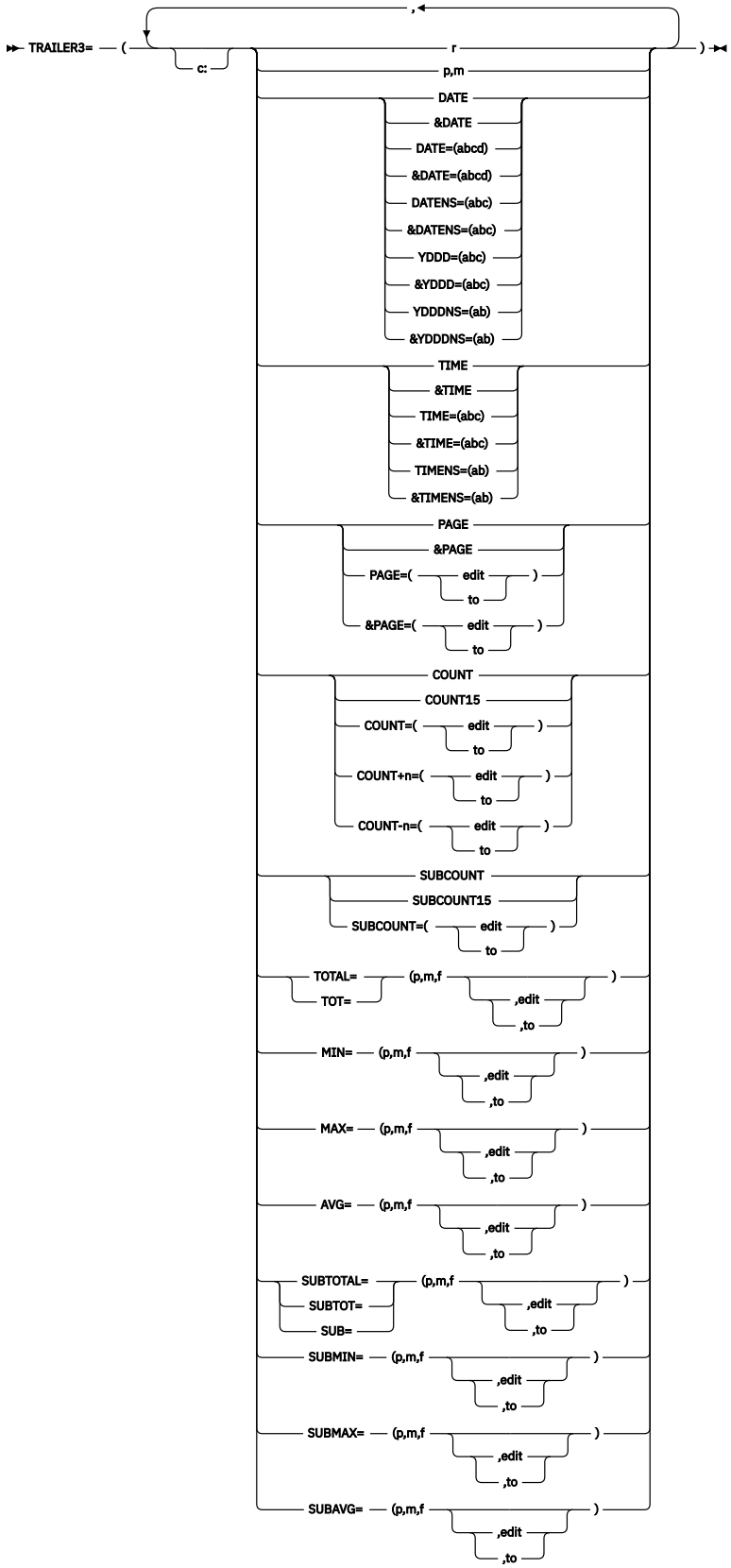
Specifies that the section header to be used with the associated section break field is to appear at the top of each page of the report, except for the report header page (if any) and report trailer page (if any), as well as before each section. If you do not specify PAGEHEAD, the section header appears only before each section; so if a section is split between pages, the section header appears only in the middle of the page. PAGEHEAD can be used when you want HEADER3 to be used as a page header as well as a section header.

If PAGEHEAD is specified for a section break field for which HEADER3 is not also specified, PAGEHEAD will not be used.

Sample Syntax:

```
  OUTFIL FNAMES=STATUS2,
     HEADER2=(10:'Status Report for all departments',5X,
                  '- ',&PAGE,' -'),
     SECTIONS=(10,8,
      HEADER3=(2/,10:'Report for department ',10,8,' on ',&DATE,2/,
               10:'   Number',25:'Average Time',/,
               10:'Completed',25:'   (in days)',/,
               10:'---------',25:'------------'),
          PAGEHEAD,SKIP=P),
       OUTREC=(10:21,5,ZD,M10,LENGTH=9,
               25:38,4,ZD,EDIT=(III.T),LENGTH=12,
               132:X)
```

TRAILER3



Specifies the section trailer to be used with the associated section break field for the reports produced for this OUTFIL group. The section trailer appears after the last data record of each section. DFSORT uses ANSI carriage control characters to control page ejects and the placement of the lines in your report, according to your specifications.

You can choose to include any or all of the following report elements in your section trailer:

- Blanks, character strings, and hexadecimal strings
- Unedited input fields from the last OUTFIL input record for which a data record appears in the section
- Current date in a variety of different forms
- Current time in a variety of different forms
- Page number, converted to different numeric formats, or edited to contain signs, decimal points, leading zeros or no leading zeros, and so on
- Any or all of the following statistics:
  - Count of data records in the section, converted to different numeric formats, or edited to contain signs, decimal points, leading zeros or no leading zeros, and so on. You can add a decimal number to the count before converting or editing it (for example, add 1 to account for writing a trailer record, or add 2 to account for writing a header and trailer record).
  - Total, minimum, maximum, or average for each specified ZD, PD, BI, FI, FL, CSF, FS, UFF, or SFF numeric input field in the data records in the section, converted to different numeric formats, or edited to contain signs, decimal points, leading zeros or no leading zeros, and so on
  - Running total, minimum, maximum, or average for each specified ZD, PD, BI, FI, FL, CSF, FS, UFF, or SFF numeric input field in the data records up to this point, converted to different numeric formats, or edited to contain signs, decimal points, leading zeros or no leading zeros, and so on.

The section trailer consists of the elements you select, in the order in which you specify them, and in the columns or lines you specify.

c:
See c: under HEADER1.

r
See r under TRAILER1.

p,m
specifies that an unedited input field, from the last OUTFIL input record for which a data record appears in the section, is to appear in the report record. See p,m under TRAILER1 for further details.

DATE
See DATE under HEADER1.

&DATE
can be used instead of DATE.

DATE=(abcd)
See DATE=(abcd) under HEADER1.

&DATE=(abcd)
can be used instead of DATE=(abcd).

DATENS=(abc)
See DATENS=(abc) under HEADER1.

&DATENS=(abc)
can be used instead of DATENS=(abc).

YDDD=(abc)
See YDDD=(abc) under HEADER1.

&YDDD=(abc)
can be used instead of YDDD=(abc).

YDDDNS=(ab)
See YDDDNS=(ab) under HEADER1.

&YDDDNS=(ab)
can be used instead of YDDDNS=(ab).

TIME
See TIME under HEADER1.

&TIME
can be used instead of TIME.

TIME=(abc)
See TIME=(abc) under HEADER1.

&TIME=(abc)
can be used instead of TIME=(abc).

TIMENS=(ab)
See TIMENS=(ab) under HEADER1.

&TIMENS=(ab)
can be used instead of TIMENS=(ab).

PAGE
See PAGE under TRAILER1.

&PAGE
can be used instead of PAGE.

PAGE=(edit) or PAGE=(to)
See PAGE=(edit) or PAGE=(to) under TRAILER1.

&PAGE=(edit) or &PAGE=(to)
can be used instead of PAGE=(edit) or PAGE=(to).

COUNT
specifies that the count of data records in the section is to appear in the report record as 8 digits, right-justified, with leading zeros suppressed. For example, if the first section has 40 input records, the second section has 40 input records, and the third section has 26 input records, COUNT will show '   40' for the first section, '   40' for the second section, and '   26' for the third section.

If slash (/) is used in OUTREC or BUILD to produce multiple data records, COUNT counts only the number of data records processed as input to OUTREC or BUILD. For example, if OUTREC or BUILD processes 3 input records and creates 2 output records for each input record, the count is 3, not 6.

COUNT15
same as COUNT except that the count appears as 15 digits.

COUNT=(edit) or COUNT=(to)
same as COUNT except that the 15–digit count appears edited or converted as specified. See p,m,f,edit under OUTREC for further details on the edit fields you can use. See p,m,f,to under OUTREC for further details on the to fields you can use.

COUNT+n=(edit) or COUNT+n=(to)
same as COUNT=(edit) or COUNT=(to) except that n is added to the 15-digit count before it is edited or converted. n can be 1 to 3 decimal digits.

COUNT-n=(edit) or COUNT-n=(to)
same as COUNT=(edit) or COUNT=(to) except that n is subtracted from the 15-digit count before it is edited or converted. n can be 1 to 3 decimal digits.

SUBCOUNT
specifies that the running count of input records up to this point in the report is to appear in the report record 8 digits, right-justified, with leading zeros suppressed. The running count accumulates the count for all sections up to and including the current section. For example, if the first section has 40 input records, the second section has 40 input records, and the third section has 26 input records, SUBCOUNT will show '   40' for the first section, '   80' for the second section, and '   106' for the third section.

If slash (/) is used in OUTREC or BUILD to produce multiple data records, SUBCOUNT counts only the number of data records processed as input to OUTREC or BUILD. For example, if OUTREC or BUILD processes 3 input records and creates 2 output records for each input record, the running count is 3, not 6

SUBCOUNT15
same as SUBCOUNT except that the running count appears as 15 digits.

SUBCOUNT=(edit) or SUBCOUNT=(to)
same as SUBCOUNT except that the 15–digit running count appears edited or converted as specified. See p,m,f,edit under OUTREC for further details on the edit fields you can use. See p,m,f,to under OUTREC for further details on the to fields you can use.

TOTAL
specifies that an edited or converted total, for the values of a numeric input field in the data records in the section, is to appear in the report record.

TOT can be used instead of TOTAL.

p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,to under TOTAL for TRAILER1.

MIN
specifies that an edited or converted minimum, for the values of a numeric input field in the data records in the section, is to appear in the report record.

p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,tounder MIN for TRAILER1.

MAX
specifies that an edited or converted maximum, for the values of a numeric input field in the data records in the section, is to appear in the report record.

p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,to under MAX for TRAILER1.

AVG
specifies that an edited or converted average, for the values of a numeric input field in the data records in the section, is to appear in the report record.

p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,to under AVG for TRAILER1.

SUBTOTAL
specifies that an edited or converted running total, for the values of a numeric input field in the data records up to this point in the report, is to appear in the report record. The running total accumulates the total for all sections up to and including the current section. For example, if the total for a selected numeric field is +200 for the first section, -250 for the second section and +90 for the third section, SUBTOTAL will be +200 for the first section, -50 for the second section and +40 for the third section.

SUBTOT or SUB can be used instead of SUBTOTAL.

p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,to under SUBTOTAL for TRAILER1.

SUBMIN
specifies that an edited or converted running minimum, for the values of a numeric input field in the data records up to this point in the report, is to appear in the report record. The running minimum selects the minimum from all sections up to and including the current section. For example, if the minimum for a selected numeric field is +200 for the first section, -250 for the second section and +90 for the third section, SUBMIN will be +200 for the first section, -250 for the second section and -250 for the third section.

p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,to under SUBMIN for TRAILER1.

SUBMAX
specifies that an edited or converted running maximum, for the values of a numeric input field in the data records up to this point in the report, is to appear in the report record. The running maximum selects the maximum from all sections up to and including the current section. For example, if the maximum for a selected numeric field is -100 for the first section, +250 for the second section and +90 for the third section, SUBMAX will be -100 for the first section, +250 for the second section and +250 for the third section.

p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,to under SUBMAX for TRAILER1.

SUBAVG
specifies that an edited or converted running average, for the values of a numeric input field in the data records up to this point in the report, is to appear in the report record. The running average computes the average for all sections up to and including the current section. For example, if the count of data records and total for a selected numeric field are 60 and +2205 for the first section, respectively, 60 and -6252 for the second section, respectively, and 23 and -320 for the third section, respectively, SUBAVG will be +36 for the first section, -33 for the second section and -30 for the third section.

p,m,f,edit or p,m,f,to
See p,m,f,edit or p,m,f,to under SUBAVG for TRAILER1.

Sample Syntax:

```
OUTFIL FNAMES=SECRPT,
  INCLUDE=(11,4,CH,EQ,C'SSD'),
  SECTIONS=(3,5,SKIP=P,
   HEADER3=(2:'Department: ',3,5,4X,'Date: ',&DATE,2/),
   TRAILER3=(2/,2:'The average for ',3,5,' is ',
                  AVG=(40,3,PD,M12),/,
               2:'The overall average so far is ',
                  SUBAVG=(40,3,PD,M12)),
  45,8,SKIP=3L,
   HEADER3=(4:'Week Ending ',45,8,2/,
              4:'Item Number',20:'Completed',/,
              4:'-',20:'-'),
   TRAILER3=(4:'The item count for week ending ',45,8,
      ' is ',COUNT=(EDIT=(II,IIT)))),
   OUTREC=(11:16,4,22:40,3,PD,M12,120:X)
```

Default for SECTIONS: None; must be specified.

NODETAIL

Specifies that data records are not to be output for the reports produced for this OUTFIL group. With NODETAIL, the data records are completely processed with respect to input fields, statistics, counts, sections breaks, and so on, but are not written to the OUTFIL data set and are not included in line counts for determining the end of a page. You can use NODETAIL to summarize the data records without actually showing them.

The NODETAIL parameter cannot be used with the IFTRAIL parameter.

Sample Syntax:

```
OUTFIL FNAMES=SUMMARY,NODETAIL,
   HEADER2=(' Date: ',DATENS=(DMY.),4X,'Page: ',PAGE,2/,
             10:'Division',25:'    Total Revenue',/,
             10:'--------',25:'-----------------'),
   SECTIONS=(3,5,
    TRAILER3=(10:3,5,
              25:TOTAL=(25,4,FI,M19,
                 LENGTH=17))),
    TRAILER1=(5/,10:'Summary of Revenue ',4/,
                 12:'Number of divisions reporting is ',
                     COUNT,/,
```

```
                    12:'Total revenue is ',
                       TOTAL=(25,4,FI,M19))
```

Default for NODETAIL: None; must be specified.

Default for OUTFIL Statements: None; must be specified. Multiple OUTFIL statements can be specified in the same and different sources; override is at the ddname level.

Applicable Functions for OUTFIL Statements: Sort, merge, and copy.

BLKCCH1

specifies that the ANSI carriage control character of '1' (page eject) in the first line of the report header (HEADER1) is to be replaced with a blank. You can use BLKCCH1 to avoid forcing a page eject at the start of the report header. If BLKCCH1 is specified without HEADER1, it will not be used.

Sample Syntax:

```
  OUTFIL FNAMES=RPT1,BLKCCH1,
         HEADER1=(30:'January Report',4/)
```

Default for BLKCCH1: None; must be specified.

BLKCCH2

specifies that the ANSI carriage control character of '1' (page eject) in the first line of the first page header (HEADER2) is to replaced with a blank. You can use BLKCCH2 to avoid forcing a page eject at the start of the first page header. BLKCHH2 does not prevent a page eject for the second and subsequent page headers. If BLKCCH2 is specified without HEADER2, it will not be used.

Sample Syntax:

```
  OUTFIL FNAMES=RPT2,
* Do page eject for HEADER1, but not for first HEADER2.
     HEADER1=(30:'January Report',2/),
     BLKCCH2,HEADER2=(5:'Account Number',25:'Name')
```

Default for BLKCCH2: None; must be specified.

BLKCCT1

specifies that the ANSI carriage control character of '1' (page eject) in the first line of the report trailer (TRAILER1) is to be replaced with a blank. You can use BLKCCT1 to avoid forcing a page eject at the start of the report trailer. If BLKCCT1 is specified without TRAILER1, it will not be used.

Sample Syntax:

```
  OUTFIL FNAMES=RPT3,BLKCCT1
,        TRAILER1=(5:'Grand Total is ',TOT=(21,10,ZD,M5))
```

Default for BLKCCT1: None; must be specified.

REMOVECC

Specifies that the ANSI carriage control character is to be removed from OUTFIL output records for this OUTFIL group before the records are written. In addition, blank lines are not used to position the page trailer (TRAILER2) at the bottom of the page.

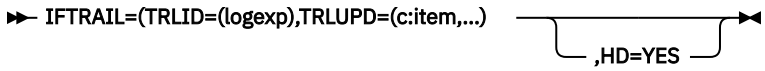
If REMOVECC is specified without any of the following report parameters, it will not be used: LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS, or NODETAIL.

Sample Syntax:

```
OUTFIL FNAMES=RPTWOCC,
  TRAILER1=(3/,'Number of records is ',
     COUNT=(TO=ZD,LENGTH=6)),
  REMOVECC
```

Default for REMOVECC: None; must be specified.

IFTRAIL



IFTRAIL allows you to update count and total values in an existing trailer (last) record to reflect the actual data records in an OUTFIL data set. Whereas TRAILER1 lets you build a new trailer record, IFTRAIL lets you update an existing trailer record using the COUNT and TOTAL features of TRAILER1. When you add, delete or modify input data records to create an OUTFIL data set, you can use IFTRAIL to ensure that the trailer record has accurate updated count and total values.

The count and total values are determined using the data records processed as input to OUTFIL. If slash (/) is used in OUTFIL BUILD to produce multiple output data records, the count and total values are determined using the data records processed as input to OUTFIL. For example, if you use OUTFIL BUILD=(1,80,/,1,80) to create 6 OUTFIL output records from 3 OUTFIL input records, the count is 3, not 6.

You must identify the trailer record using a unique condition (for example, 'TRL' in positions 1-3). You then specify how count and total values in the identified trailer record are to be updated. The trailer record will NOT be treated as a data record, that is, no other OUTFIL processing will be performed against the trailer record. You can optionally specify that the header (first) record will NOT be treated as a data record.

As a simple example, you could use the following OUTFIL statement to write the header (first) record, data records with 'D1', and trailer record (identified by '9' in position 1) in the OUTFIL data set, and set an accurate count and total in the trailer record for the included input data records.

```
OUTFIL FNAMES=OUT1,INCLUDE=(10,2,CH,EQ,C'D1'),
  IFTRAIL=(HD=YES,TRLID=(1,1,CH,EQ,C'9'),
    TRLUPD=(11:COUNT=(M11,LENGTH=8),
     25:TOTAL=(15,6,ZD,EDIT=(IIIIIT))))
```

The trailer record **will not** be treated as a data record; it will not be subject to INCLUDE processing, and will not be used for the updated count or total values.

Since HD=YES is specified, the header record **will not** be treated as a data record; it will not be subject to INCLUDE processing, and will not be used for the updated count or total values.

IFTRAIL
Allows you to update count and total fields in the trailer record of the data sets for an OUTFIL group. You can use IFTRAIL to identify a trailer record and indicate count and total fields to be updated in that record. The identified trailer record will not be used to determine the count or totals. If you specify HD=YES, the header (first) record will not be used to determine the count or totals.

You cannot specify IFTRAIL with any of the following operands in an OUTFIL statement: HEADER1, TRAILER1, HEADER2, TRAILER2, NODETAIL, SECTIONS, LINES, SPLIT, SPLITBY, SPLIT1R, REPEAT, FTOV, VTOF, VLTRIM, VLTRAIL or VLFILL.

You must specify the TRLID and TRLUPD operands. The HD=YES operand is optional.

TRLID=(logexp)
Specifies the criteria to be tested to determine if a record is the trailer record. When a record is found that meets the criteria, it will be treated as the last record, and subsequent records will not

be processed for the OUTFIL group. Thus if you use TRLID=(1,1,CH,EQ,C'9') to identify the trailer record and you have three records with '9' in position 1, the first record with '9' in position 1 will be treated as the trailer record, and records after that will not appear in the OUTFIL data sets.

See the discussion of INCLUDE statement in "INCLUDE control statement" on page 91 for details of the logical expressions that you can use. You can specify all of the logical expressions for TRLID in the same way that you can specify them for the INCLUDE statement, except that:

- You cannot specify FORMAT=f
- You cannot specify D2 format
- Locale processing is not used
- VLSCMP and VLSHRT are not used. The trailer record must be long enough to contain all fields in the logical expression. Trailer record checking will be skipped for any variable-length record that is too short to contain a field in the TRLID logical expression.

The trailer record will not be treated as a data record; all other OUTFIL processing (for example, INCLUDE, OMIT, BUILD, OVERLAY, and so on) will be bypassed for the trailer record. The updated count and total values will not include the trailer record.

TRLUPD=(c:item,...)

Specifies the position where each count or total is to be updated in the trailer record.

For fixed-length records, the first input and output data byte starts at position 1. The trailer record will be truncated or padded with blanks to the length of the data records, if appropriate. You cannot change the length of the trailer record using TRLUPD.

For variable-length records, the first input and output data byte starts at position 5, after the RDW in positions 1-4. The original length of the trailer record will not be changed for output. You cannot change the length of the trailer record using TRLUPD. If you specify an item beyond the end of the trailer record, it will not be updated.

c:

specifies the output position (column) in the trailer record to be updated. If you do not specify c: for the first item, it defaults to 1:. If you do not specify c: for any other item, it starts after the previous item. You must specify items in ascending output column order, and an item must not overlap a previous item in the trailer record. For a variable-length trailer record, you must not specify columns 1-4.

item

specifies the count or total you want to update. Any of the following count and total items can be used as described for TRAILER1:

- COUNT=(edit)
- COUNT=(to)
- COUNT+n=(edit)
- COUNT+n=(to)
- COUNT-n=(edit)
- COUNT-n=(to)
- TOTAL=(p,m,f,edit)
- TOTAL=(p,m,f,to)
- TOT=(p,m,f,edit)
- TOT=(p,m,f,to)

The count and total values are determined using the data records processed as input to OUTFIL.

If TOT or TOTAL is used, the specified p,m,f field from each data record will be totalled, even if the trailer record is not found. OUTFIL BUILD, OVERLAY or IFTHEN processing does NOT affect

the values that are totalled; the values in the original OUTFIL input records are used for the totals.

An invalid field amount can result in a data exception (0C7 ABEND) or incorrect numeric output. Be careful to use HD=YES if the header record does not have valid total fields. For example, if the input file has these records:

```
H***
0001
0002
T***
```

and this IFTRAIL operand is specified:

```
   IFTRAIL=(TRLID=(1,1,CH,EQ,C'T'),
     TRLUPD=(5:TOT=(1,4,ZD,TO=ZD,LENGTH=5))
```

an 0C7 ABEND will result when DFSORT attempts to total the invalid 1,4,ZD field in the header (H***) record. The use of HD=YES would prevent this.

An invalid total field can also be encountered if the trailer record is not found due to erroneous TRLID conditions. For example, if the input file has these records:

```
H***
0001
0002
T***
```

and this IFTRAIL operand is specified:

```
   IFTRAIL=(TRLID=(1,2,CH,EQ,C'TR'),
     TRLUPD=(5:TOT=(1,4,ZD,TO=ZD,LENGTH=5))
```

the last (T***) record will not be identified as the trailer record since it does not have 'TR' in positions 1-2. Thus, an 0C7 ABEND will result when DFSORT attempts to total the invalid 1,4,ZD field in the last (T***) record. The use of the correct condition for identifying the trailer (T***) record would prevent this.

If slash (/) is used in OUTFIL BUILD to produce multiple output data records, the count and total values are determined using the data records processed as input to OUTFIL. For example, if you use OUTFIL BUILD=(1,80,/,1,80) to create 6 OUTFIL output records from 3 OUTFIL input records, the count is 3, not 6.

HD=YES

Specify HD=YES if you want the first record treated as a header record rather than as a data record. With HD=YES, OUTFIL processing (for example, INCLUDE, OMIT, BUILD, OVERLAY, and so on) will be bypassed for the header record. The updated count and total values will not include the header record.

If HD=YES is specified, the TRLID test will not be applied to the first record.

Sample Syntax:

```
OUTFIL IFTRAIL=(HD=YES,
   TRLID=(1,3,CH,EQ,C'TRL'),
   TRLUPD=(11:COUNT=(M11,LENGTH=4),
           21:TOT=(M11,LENGTH=6)))
```

Default for IFTRAIL: None; must be specified.

OUTFIL statements notes

- OUTFIL processing is supported for sort, merge, and copy applications, but only by the Blockset technique.
-

- The ODMAXBF value in effect specifies the maximum buffer space to be used for each OUTFIL data set. ODMAXBF can be specified as an installation or run-time option, or in an ICEIEXIT routine. The default value of 2M is recommended for the ODMAXBF option in effect. Lowering ODMAXBF can cause performance degradation for the application, but might be necessary if you consider the amount of storage used for OUTFIL processing to be a problem. Raising ODMAXBF can improve EXCPs for the application, but can also increase the amount of storage needed.
-
- The storage used for OUTFIL processing will be adjusted automatically according to the total storage available, the storage needed for non-OUTFIL processing, and the number of OUTFIL data sets and their attributes (for example, block size). OUTFIL processing will be subject to the ODMAXBF limit in effect and the system storage limits (for example, IEFUSI), but not to the DFSORT storage limits (that is, SIZE, MAXLIM, and TMAXLIM). DFSORT attempts to use storage above 16MB virtual for OUTFIL processing whenever possible.
- The VSAMBSP option applies to SORTOUT data sets, but not to OUTFIL data sets. The NOBLKSET option will be ignored if OUTFIL data sets are being processed. An E39 exit routine is entered for the SORTOUT data set, but not for OUTFIL data sets.
- For fixed-format OUTFIL data sets: DFSORT will determine each OUTFIL data set LRECL based on the length of the reformatted records for the group, or the length of the OUTFIL input records (if OUTREC, BUILD, OVERLAY, FINDREP and IFTHEN are not specified for the group). For VSAM data sets, the maximum record size defined in the cluster is equivalent to the LRECL.

  If an OUTFIL data set LRECL is not specified or available, DFSORT will set it to the determined LRECL. If an OUTFIL data set LRECL is specified or available, it must not be less than the determined LRECL, or more than the determined LRECL if the OUTREC, BUILD, OVERLAY, FINDREP or IFTHEN parameter is specified. In other words, the LRECL value cannot be used to pad the output records, or to truncate the records produced by OUTREC, BUILD, OVERLAY, FINDREP or IFTHEN parameter processing.

  In general, OUTREC, BUILD, OVERLAY, or IFTHEN processing should be used to pad the records and OUTREC, BUILD, or IFTHEN processing should be used to truncate the records. In either case, the LRECL should either not be specified or set to the length of the reformatted records.
- For variable-format OUTFIL data sets: DFSORT will determine each OUTFIL data set maximum LRECL based on the length of the reformatted records for the group, or the length of the OUTFIL input records (if OUTREC, BUILD, OVERLAY, FINDREP and IFTHEN are not specified for the group). If an OUTFIL data set maximum LRECL is not specified or available, DFSORT will set it to the determined maximum LRECL. For VSAM data sets, the maximum record size defined in the cluster is four bytes more than the maximum LRECL.
- When you create an OUTFIL report, the length for the longest or only data record must be equal to or greater than the maximum report record length. You can use the OUTREC, BUILD, OVERLAY, FINDREP or IFTHEN parameter to force a length for the data records that is longer than any report record; you can then either let DFSORT compute and set the LRECL, or ensure that the computed LRECL is equal to the existing or specified LRECL. Remember to allow an extra byte in the LRECL for the ANSI carriage control character.

  For example, if your data records are 40 bytes, but your longest report record is 60 bytes, you can use an OUTREC parameter such as:

  ```
  OUTREC=(1,40,80:X)
  ```

  DFSORT will then set the LRECL to 81 (1 byte for the ANSI carriage control character plus 80 bytes for the length of the data records), and pad the data records with blanks on the right.

  If you don't want the ANSI carriage control characters to appear in the output data set, use the REMOVECC parameter to remove them. For example, if you specify:

  ```
  OUTREC=(1,40,80:X),REMOVECC
  ```

  DFSORT will set the LRECL to 80 instead of 81 and remove the ANSI carriage control character from each record before it is written.

System errors can result if you print an OUTFIL report containing records longer than your printer can handle.

- DFSORT uses appropriate ANSI carriage controls (for example, C'-' for triple space) in header and trailer records when possible to reduce the number of report records written. DFSORT always uses the single space carriage control (C' ') in data records. Although these carriage control characters may not be shown when you view an OUTFIL data set (depending on how you view it), they will be used if you print the report. If you are creating a report for viewing and want blank lines to appear in headers and trailers, specify a line of blanks instead of using n/. For example, instead of specifying:

```
OUTFIL FNAMES=RPT,
 HEADER2=(2/,'start of header',2/,'next line')
```

  which will result in blank lines for the printer, but not for viewing, specify:

```
OUTFIL FNAMES=RPT,
 HEADER2=(X,/,X,/,'start of header',/,X,/,'next line')
```

  If you don't want the ANSI carriage control characters to appear in the output data set, use the REMOVECC parameter to remove them.
- For variable-length records, the first entry in the OUTREC, BUILD or IFTHEN BUILD parameter must specify or include the unedited 4-byte record descriptor word (RDW), that is, the first field must be 1,4 or 1,m with m greater than 4. DFSORT sets the length of the reformatted record in the RDW.

  If the first field in the data portion of the input record is to appear unedited in the reformatted record immediately following the RDW, the entry in the OUTREC, BUILD, or IFTHEN BUILD parameter can specify both RDW and data field in one (1,m,...). Otherwise, the RDW must be specifically included in the reformatted record (for example, 1,4,1,4,HEX).

  For variable-length OUTFIL header or trailer records, you must not specify the 4-byte RDW at the beginning of the record.
- For variable-length records, OVERLAY, IFTHEN OVERLAY or IFTHEN PUSH items must not overlay the RDW in bytes 1-4. You must ensure that 1:, 2:, 3: or 4: is not specified or defaulted for any OVERLAY or PUSH item. Note that the default for the first OVERLAY or PUSH item is 1:, so you must override it.
- With FIELDS, BUILD or IFTHEN BUILD, the variable part of the input record (that part beyond the minimum record length) can be included in the reformatted record and, if included, must be the last part. For example:

```
OUTFIL BUILD=(1,8,20C'*',9)
```

  With OVERLAY, the variable part of the input record must not be included in the reformatted record.
- For variable-length records, the FIELDS or BUILD parameter of the INREC and OUTREC statement and the OUTREC or FIELDS parameter of the OUTFIL statement must all specify position-only for the last part, or all not specify position-only for the last part. OVERLAY or IFTHEN, and FIELDS, OUTREC or BUILD, can differ with respect to position-only. See “INREC statement notes” on page 150 for more details.
- If there are no OUTFIL input records for an OUTFIL group, the headers and trailers appear without any data records. Blanks will be used for any specified unedited input fields, and zero values will be used for any specified statistics fields.
- If a variable-length OUTFIL input record is too short to contain a specified unedited input field for a report header or trailer, blanks will be used for the missing bytes. If a variable-length OUTFIL input record is too short to contain a specified section break field or statistics field, zeros will be used for the missing bytes, intentionally or unintentionally.
- If a variable-length OUTFIL input record is too short to contain an OUTFIL INCLUDE or OMIT compare field, the action DFSORT takes depends on the settings for VLSCMP/NOVLSCMP and VLSHRT/NOVLSHRT. For details, see the discussion of the VLSCMP and NOVLSCMP options in “OPTION control statement” on page 173.
- If a variable-length OUTFIL input record is too short to contain an OUTFIL OUTREC or BUILD field, DFSORT will terminate unless the VLFILL=byte parameter is specified.

- If a variable-length OUTFIL input record is too short to contain an OUTFIL OVERLAY or IFTHEN field, blanks will be used for the missing bytes.
- If a variable-length OUTFIL output data record is longer than the LRECL of its OUTFIL data set, the action DFSORT takes depends on the settings for VLLONG/NOVLLONG. For details, see the discussion of the VLLONG and NOVLLONG options in "OPTION control statement" on page 173. Note that VLLONG can be used to truncate long OUTFIL data records, but has no effect on long OUTFIL header or trailer records.
- If an unedited page number overflows 6 digits, an edited page number overflows 15 digits, a count or running count overflows 15 digits, or a total or running total overflows 31 digits, the overflowing value will be truncated to the number of digits allowed, intentionally or unintentionally.
- Multiple OUTFIL statements can be specified in the same and different sources. If a ddname occurs more than once in the same source, the ddname is associated with the first OUTFIL group in which it appears. For example, if the following is specified in SYSIN:

```
  OUTFIL FNAMES=(OUT1,OUT2),INCLUDE=(1,1,CH,EQ,C'A')
  OUTFIL FNAMES=(OUT3,OUT1),SAVE
```

  OUT1 and OUT2 are processed as part of the first OUTFIL group, that is, with INCLUDE. OUT3 is processed as part of the second OUTFIL group, that is, with SAVE; but OUT1 is not because it is a duplicate ddname.

  If a ddname occurs in more than one source, the ddname is associated with the highest source OUTFIL group in which it appears. For example, if the following is specified in DFSPARM:

```
  OUTFIL FNAMES=(OUT1,OUT2),INCLUDE=(1,1,CH,EQ,C'A')
```

  and the following is specified in SYSIN:

```
  OUTFIL FNAMES=(OUT3,OUT1),SAVE
```

  OUT1 and OUT2 are processed as part of the DFSPARM OUTFIL group, that is, with INCLUDE. OUT3 is processed as part of the SYSIN OUTFIL group, that is, with SAVE; but OUT1 is not because it is an overridden ddname.
- OUTFIL statements cannot be passed to or returned from an EFS program. The D2 format cannot be specified in the INCLUDE or OMIT parameter of an OUTFIL statement.
- If SZERO is in effect, -0 is treated as negative and +0 is treated as positive for edited or converted input fields, minimums, maximums, decimal constants and the results of arithmetic expressions. If NOSZERO is in effect, -0 and +0 are treated as positive for edited or converted input fields, minimums, maximums, decimal constants, and the results of arithmetic expressions.
- If SZERO is in effect, -0 compares as less than +0 when numeric fields and constants are used. If NOSZERO is in effect, -0 compares as equal to +0 when numeric fields and constants are used.

  Note: OPTION SZERO or OPTION NOSZERO is ignored for OUTFIL INCLUDE=(logexp) or OMIT=(logexp), or for OUTFIL IFTHEN=(WHEN=(logexp),...) unless the OPTION statement is found in a higher source (for example, DFSPARM is a higher source than SYSIN) or **before** the OUTFIL statement in the same source. For example, NOSZERO will be used in both of the following cases:

  Case 1:

```
//DFSPARM DD *
  OPTION COPY,NOSZERO
/*
//SYSIN DD *
  OUTFIL INCLUDE=(1,2,FS,EQ,+0)
/*
```

  Case 2:

```
//SYSIN DD *
  OPTION COPY,NOSZERO
  OUTFIL IFTHEN=(WHEN=(1,2,FS,EQ,+0),OVERLAY=(28:C'A')),
```

```
         IFTHEN=(WHEN=NONE,OVERLAY=(28:C'B'))
/*
```

OUTFIL features—examples

Example 1

```
OPTION COPY
OUTFIL INCLUDE=(15,6,CH,EQ,C'MSG005'),FNAMES=M005
OUTFIL INCLUDE=(15,6,CH,EQ,C'MSG022'),FNAMES=M022
OUTFIL INCLUDE=(15,6,CH,EQ,C'MSG028'),FNAMES=M028
OUTFIL INCLUDE=(15,6,CH,EQ,C'MSG115'),FNAMES=M115
OUTFIL SAVE,FNAMES=UNKNOWN
```

This example illustrates how records can be distributed to different OUTFIL data sets based on criteria you specify:

- Input records with MSG005 in bytes 15 through 20 will be written to the OUTFIL data set associated with ddname M005.
- Input records with MSG022 in bytes 15 through 20 will be written to the OUTFIL data set associated with ddname M022.
- Input records with MSG028 in bytes 15 through 20 will be written to the OUTFIL data set associated with ddname M028.
- Input records with MSG115 in bytes 15 through 20 will be written to the OUTFIL data set associated with ddname M115.
- Input records with anything else in bytes 15 through 20 will be written to the OUTFIL data set associated with ddname UNKNOWN

Example 2

```
SORT FIELDS=(18,5,ZD,D)
OUTFIL FNAMES=(V,VBU1,VBU2)
OUTFIL FNAMES=(F,FBU1),
  CONVERT,OUTREC=(11,3,X,18,5,X,X'0000000F')
OUTFIL FNAMES=VINF,OUTREC=(1,4,C'*',5,20,C'*',25)
```

This example illustrates how multiple sorted output data sets can be created and how a variable-length record data set can be converted to a fixed-length record data set:

- The first OUTFIL statement writes the variable-length input records to the variable-length OUTFIL data sets associated with ddnames V, VBU1, and VBU2.
- The second OUTFIL statement reformats the variable-length input records to fixed-length output records and writes them to the fixed-length OUTFIL data sets associated with ddnames F and FBU1. CONVERT is used to indicate that a variable-length data set is to be converted to a fixed-length data set; OUTREC is used to describe how the variable-length input records are to be reformatted as fixed-length output records.
- The third OUTFIL statement reformats the variable-length input records and writes them to the variable-length OUTFIL data set associated with ddname VINF. OUTREC is used to insert asterisks between fields. 1,4 represents the RDW. 25 represents the variable part at the end of the input record.

Example 3

```
SORT FIELDS=(15,6,ZD,A)
OUTFIL FNAMES=USA,
   HEADER2=(5:'Parts Completion Report for USA',2/,
            5:'Printed on ',DATE,
              ' at ',TIME=(12:),3/,
            5:'Part  ',20:'Completed',35:'   Value ($)',/,
            5:'------',20:'---------',35:'------------'),
   OUTREC=(5:15,6,ZD,M11,
           20:3,4,ZD,M12,LENGTH=9,
           35:38,8,ZD,M18,LENGTH=12,
```

```
            132:X)
  OUTFIL FNAMES=FRANCE,
     HEADER2=(5:'Parts Completion Report for France',2/,
              5:'Printed on ',DATE=(DM4/),
                ' at ',TIME,3/,
              5:'Part  ',20:'Completed',35:'   Value (F)',/,
              5:'------',20:'---------',35:'------------'),
     OUTREC=(5:15,6,ZD,M11,
             20:3,4,ZD,M16,LENGTH=9,
             35:38,8,ZD,M22,LENGTH=12,
             132:X)
  OUTFIL FNAMES=DENMARK,
     HEADER2=(5:'Parts Completion Report for Denmark',2/,
              5:'Printed on ',DATE=(DMY-),
                ' at ',TIME=(24.),3/,
              5:'Part  ',20:'Completed',35:'  Value (kr)',/,
              5:'------',20:'---------',35:'------------'),
     OUTREC=(5:15,6,ZD,M11,
             20:3,4,ZD,M13,LENGTH=9,
             35:38,8,ZD,M19,LENGTH=12,
             132:X)
```

This example illustrates how reports for three different countries can be produced from sorted fixed-length input records. The reports differ only in the way that date, time, and numeric formats are specified:

1. The first OUTFIL statement produces a report that has the date, time, and numeric formats commonly used in the United States.
2. The second OUTFIL statement produces a report that has the date, time, and numeric formats commonly used in France.
3. The third OUTFIL statement produces a report that has the date, time, and numeric formats commonly used in Denmark.

Of course, any one of the three reports can be produced by itself using a single OUTFIL statement instead of three OUTFIL statements. (This may be necessary if you are sorting character data according to a specified locale for that country.)

The FNAMES parameter specifies the ddname (USA, FRANCE, DENMARK) associated with the fixed-length data set for that report.

The HEADER2 parameter specifies the page header to appear at the top of each page for that report, which will consist of:

- A line of text identifying the report. Note that all English text in the report can be replaced by text in the language of that country.
- A blank line (2/).
- A line of text showing the date and time. Note that variations of the DATE, DATE=(abcd), TIME, and TIME=(abc) operands are used to specify the date and time in the format commonly used in that country.
- Two blank lines (3/).
- Two lines of text showing headings for the columns of data. Note that the appropriate currency symbol can be included in the text.

The OUTREC parameter specifies the three columns of data to appear for each input record as follows:

- A 6-byte edited numeric value produced by transforming the ZD value in bytes 15 through 20 according to the pattern specified by M11. M11 is a pattern for showing integers with leading zeros.
- A 9-byte (LENGTH=9) edited numeric value produced by transforming the ZD value in bytes 3 through 6 according to the pattern for integer values with thousands separators commonly used in that country. M12 uses a comma for the thousands separator. M16 uses a blank for the thousands separator. M13 uses a period for the thousands separator.
- A 12-byte (LENGTH=12) edited numeric value produced by transforming the ZD value in bytes 38 through 45 according to the pattern for decimal values with thousands separators and decimal separators commonly used in that country. M18 uses a comma for the thousands separator and a period

for the decimal separator. M22 uses a blank for the thousands separator and a comma for the decimal separator. M19 uses a period for the thousands separator and a comma for the decimal separator.

Table 50 on page 264 shows the twenty-seven pre-defined edit masks (M0-M26) from which you can choose.

132:X is used at the end of the OUTREC parameter to ensure that the data records are longer than the report records. This will result in an LRECL of 132 for the fixed-length OUTFIL data sets (1 byte for the ANSI control character and 131 bytes for the data).

The three reports might look as follows:

```
   Parts Completion Report for USA

   Printed on 03/25/05 at 01:56:20 pm


   Part           Completed         Value ($)
   ------         ---------      ------------
   000310               562          8,317.53
   001184             1,234         23,456.78
   029633                35            642.10
   192199             3,150        121,934.65
   821356               233          2,212.34
```

```
   Parts Completion Report for France

   Printed on 25/03/2005 at 13:56:20


   Part           Completed         Value (F)
   ------         ---------      ------------
   000310               562          8 317,53
   001184             1 234         23 456,78
   029633                35            642,10
   192199             3 150        121 934,65
   821356               233          2 212,34
```

```
   Parts Completion Report for Denmark

   Printed on 25-03-05 at 13.56.20


   Part           Completed        Value (kr)
   ------         ---------      ------------
   000310               562          8.317,53
   001184             1.234         23.456,78
   029633                35            642,10
   192199             3.150        121.934,65
   821356               233          2.212,34
```

Example 4

```
SORT FIELDS=(3,10,A,16,13,A),FORMAT=CH
OUTFIL FNAMES=WEST,
  INCLUDE=(42,6,CH,EQ,C'West'),
  HEADER1=(5/,18:'    Western Region',3/,
              18:'Profit and Loss Report',3/,
              18:'     for  ',&DATE,3/,
              18:'      Page',&PAGE),
  OUTREC=(6:16,13,24:31,10,ZD,M5,LENGTH=20,75:X),
  SECTIONS=(3,10,SKIP=P,
    HEADER3=(2:'Division:  ',3,10,5X,'Page:',&PAGE,2/,
             6:'Branch Office',24:'       Profit/(Loss)',/,
             6:'-------------',24:'--------------------'),
   TRAILER3=(6:'=============',24:'====================',/,
             6:'Total',24:TOTAL=(31,10,ZD,M5,LENGTH=20),/,
             6:'Lowest',24:MIN=(31,10,ZD,M5,LENGTH=20),/,
             6:'Highest',24:MAX=(31,10,ZD,M5,LENGTH=20),/,
             6:'Average',24:AVG=(31,10,ZD,M5,LENGTH=20),/,
             3/,2:'Average for all Branch Offices so far:',
                X,SUBAVG=(31,10,ZD,M5))),
    TRAILER1=(8:'Page ',&PAGE,5X,'Date:  ',&DATE,5/,
```

```
               8:'Total Number of Branch Offices Reporting:  ',
                 COUNT,2/,
               8:'Summary of Profit/(Loss) for all',
                 ' Western Division Branch Offices',2/,
               12:'Total:',
                  22:TOTAL=(31,10,ZD,M5,LENGTH=20),/,
               12:'Lowest:',
                  22:MIN=(31,10,ZD,M5,LENGTH=20),/,
               12:'Highest:',
                  22:MAX=(31,10,ZD,M5,LENGTH=20),/,
               12:'Average:',
                  22:AVG=(31,10,ZD,M5,LENGTH=20))
```

This example illustrates how a report can be produced with a header and trailer page and sections of columns of data, from a sorted subset of fixed-length input records.

The FNAMES parameter specifies the ddname (WEST) associated with the fixed-length data set for the report.

The INCLUDE parameter specifies the records to be selected for the report.

The HEADER1 parameter specifies the report header to appear as the first page of the report, which will consist of five blank lines (5/) followed by four lines of text, each separated by 2 blank lines (3/). The last two lines of text will show the date (&DATE) and page number (&PAGE), respectively.

The OUTREC parameter specifies the two columns of data to appear for each selected input record as follows:

- The character string from bytes 16 through 28 of the input record.
- A 20-byte (LENGTH=20) edited numeric value produced by transforming the ZD value in bytes 31 through 40 according to the pattern specified by M5.

The SECTIONS parameter specifies the section break field (3,10), page ejects between sections (SKIP=P), the header (HEADER3) to appear before each section and the trailer (TRAILER3) to appear after each section. The section header will consist of a line of text showing the page number, a blank line (2/) and two lines of text showing the headings for the columns of data. The section trailer will consist of a line of text separating the data from the trailer, lines of text showing the total (TOTAL), minimum (MIN), maximum (MAX) and average (AVG) for the data in the section as edited numeric values, two blank lines, and a line of text showing the running average (SUBAVG) for all of the data records in the report up to this point.

The TRAILER1 parameter specifies the report trailer to appear as the last page of the report, which will consist of a line of text showing the page and date, four blank lines (5/), a text line showing the count of data records in the report, a blank line, a line of text, a blank line, and lines of text showing the total, minimum maximum and average for all of the data in the report as edited numeric values.

75:X is used at the end of the OUTREC parameter to ensure that the data records are longer than the report records. This will result in an LRECL of 76 for the fixed-length OUTFIL data set (1 byte for the ANSI control character and 75 bytes for the data).

The report might look as follows:

```
                    Western Region

                 Profit and Loss Report

                      for  08/20/05

                       Page     1
```

```
Division:  Chips          Page:     2

      Branch Office            Profit/(Loss)
      -------------     --------------------
      Gilroy                      554,843.42
      Los Angeles                 (22,340.14)
      Morgan Hill                 987,322.32
      Oakland                     234,124.32
      San Francisco               (32,434.31)
      San Jose                  1,232,133.35
      San Martin                  889,022.03
      =============     ====================
      Total                     3,842,670.99
      Lowest                      (32,434.31)
      Highest                   1,232,133.35
      Average                     548,952.99


  Average for all Branch Offices so far:      548,952.99
```

```
Division:  Ice Cream      Page:     3

      Branch Office            Profit/(Loss)
      -------------     --------------------
      Marin                       542,341.23
      Napa                        857,342.83
      San Francisco               922,312.45
      San Jose                       (234.55)
      San Martin                1,003,467.30
      =============     ====================
      Total                     3,325,229.26
      Lowest                         (234.55)
      Highest                   1,003,467.30
      Average                     665,045.85


  Average for all Branch Offices so far:      597,325.02
```

```
Division:  Pretzels       Page:     4

      Branch Office            Profit/(Loss)
      -------------     --------------------
      Marin                     5,343,323.44
      Morgan Hill                 843,843.40
      Napa                      5,312,348.56
      San Francisco             5,412,300.05
      San Jose                  1,234,885.34
      San Martin                   (2,343.82)
      =============     ====================
      Total                    18,144,356.97
      Lowest                       (2,343.82)
      Highest                   5,412,300.05
      Average                   3,024,059.49


  Average for all Branch Offices so far:    1,406,236.51
```

```
         Page      5     Date:  08/20/05



        Total Number of Branch Offices Reporting:        18

        Summary of Profit/(Loss) for all Western Division Branch Offices

            Total:          25,312,257.22
            Lowest:            (32,434.31)
            Highest:         5,412,300.05
            Average:         1,406,236.51
```

Example 5

```
  SORT FIELDS=(6,5,CH,A)
  OUTFIL FNAMES=STATUS,
    HEADER2=(1:C'PAGE ',&PAGE,C' OF STATUS REPORT FOR ',&DATE,2/,
             6:C'ITEM ',16:C'STATUS      ',31:C'PARTS',/,
             6:C'-----',16:C'------------',31:C'-----'),
     OUTREC=(1,4,
             10:6,5,
             20:14,1,CHANGE=(12,
                             C'1',C'SHIP',
                             C'2',C'HOLD',
                             C'3',C'TRANSFER'),
                     NOMATCH=(C'*CHECK CODE*'),
             37:39,1,BI,M10,
             120:X)
```

This example illustrates how a report can be produced with a page header and columns of data from sorted variable-length input records, using a lookup table.

The FNAMES parameter specifies the ddname (STATUS) associated with the variable-length data set for the report.

The HEADER2 parameter specifies the page header to appear at the top of each page, which will consist of a line of text showing the page number (&PAGE) and date (&DATE), a blank line (2/), and two lines of text showing headings for the columns of data.

The OUTREC parameter specifies the RDW and three columns of data to appear for each input record as follows (remember that byte 5 is the first byte of data for variable-length records):

- The character string from bytes 6 through 10 of the input record
- A character string produced by finding a match for byte 14 of the input record in the table defined by CHANGE (lookup and change). NOMATCH indicates the character string to be used if byte 14 does not match any of the entries in the CHANGE table.
- An edited numeric value produced by transforming the BI value in byte 39 according to the pattern specified by M10.

With variable-length input records, you must account for the RDW when specifying the c: values for OUTREC, but not for headers or trailers. The 1: used for the first line of HEADER2 causes it to start in the first data byte (by contrast, 5: must be used to specify the first OUTREC data byte for variable-length records). Also, because 6: is used for the ITEM heading, 10: must be used for the ITEM data to get the heading and data to line up in columns.

120:X is used at the end of the OUTREC parameter to ensure that the data records are longer than the report records. This will result in a maximum LRECL of 121 for the variable-length OUTFIL data set (1 byte for the ANSI control character and a maximum of 120 bytes for the data).

The first page of the printed report might start as follows:

```
PAGE      1 OF STATUS REPORT FOR 07/18/05

     ITEM      STATUS         PARTS
     -----     ------------   -----
     00082     HOLD              36
     00123     SHIP             106
     00300     *CHECK CODE*      95
     10321     TRANSFER          18
     12140     SHIP             120
```

Example 6

```
  OPTION COPY
  OUTFIL FNAMES=(A1,A2,A3),SPLIT
  OUTFIL FNAMES=(B1,B2,B3),SPLITBY=25
  OUTFIL FNAMES=(C1,C2,C3),SPLIT1R=25
```

This example illustrates different ways to split 77 input records among three OUTFIL data sets.

The first OUTFIL statement uses SPLIT to rotate the records among the three OUTFIL data sets one record at a time. A1 will have records 1, 4, ...,76. A2 will have records 2, 5, ...,77. A3 will have records 3, 6, ...,75. A1 will have 26 records. A2 will have 26 records. A3 will have 25 records. A1, A2 and A3 will each have non-contiguous records.

The second OUTFIL statement uses SPLITBY=25 to rotate the records among the three OUTFIL data sets 25 records at a time. B1 will have records 1-25 and 76-77. B2 will have records 26-50. B3 will have records 51-75. B1 will have 27 records. B2 will have 25 records. B3 will have 25 records. B1 will have non-contiguous records. B2 and B3 will have contiguous records.

The third OUTFIL statement uses SPLIT1R=25 to rotate the records once among the three OUTFIL data sets 25 records at a time. C1 will have records 1-25. C2 will have records 26-50. C3 will have records 51-77. C1 will have 25 records. C2 will have 25 records. C3 will have 27 records. C1, C2 and C3 will have contiguous records.

Example 7

```
OPTION COPY
OUTFIL FNAMES=RANGE1,ENDREC=1000000
OUTFIL FNAMES=RANGE2,STARTREC=1000001,ENDREC=2000000
OUTFIL FNAMES=RANGE3,STARTREC=2000001,ENDREC=3000000
OUTFIL FNAMES=RANGE4,STARTREC=3000001,ENDREC=4000000
OUTFIL FNAMES=(RANGE5,EXTRA),STARTREC=4000001
```

This example illustrates how specific ranges of output records can be written to different output data sets. A typical application might be database partitioning.

The first 1 million records will be written to the data set associated with RANGE1, the second million to RANGE2, the third million to RANGE3, and the fourth million to RANGE4. The remaining records will be written to both the data set associated with RANGE5 and the data set associated with EXTRA (SAVE or STARTREC=4000001 will accomplish the same purpose in this case).

Note that the INCLUDE, OMIT, and SAVE parameters of OUTFIL can also be used to select records to be written to different output data sets, based on criteria you specify.

Example 8

```
OPTION COPY,Y2PAST
OUTFIL FNAMES=Y4,
       OUTREC=(1,19,
               21,2,PD0,M11,C'/',  transform mm
               22,2,PD0,M11,C'/',  transform dd
               20,2,Y2P,           transform yy to yyyy
               24,57)
```

This example illustrates how to transform an existing data set with a packed decimal date field of the form P'yymmdd' (X'0yymmddC') in bytes 20-23 into a new data set with a character date field of the form C'mm/dd/yyyy' in bytes 20-29. yy represents the two-digit year, yyyy represents the four-digit year, mm represents the month, dd represents the day, and C represents a positive sign.

The input data set has an LRECL of 80 and the Y4 data set will have an LRECL of 86.

The Y2PAST=26 option sets the century window to be used to transform two-digit years into four-digit years. If the current year is 2006, the century window will be 1980 to 2079. Using this century window, the input and output fields might be as follows:

```
Input Field (HEX)              Output Field (CH)
   20                             20
    |                              |
    0020505F                       05/05/2002
    0950823C                       08/23/1995
    0980316C                       03/16/1998
    0000316F                       03/16/2000
```

Example 9

```
OPTION COPY,Y2PAST=1996
OUTFIL FNAMES=SPCL,
       OUTREC=(1,14, copy positions 1-14
          15,6,Y2T,  transform yy to yyyy - allow blanks
          21,20)     copy positions 21 - 40
```

This example illustrates how to transform an existing data set with a character date field of the form C'yymmdd' and blank special indicators in bytes 15-20, into a new date set with a character date field of the form C'yyyymmdd' and blank special indicators in bytes 15-22.

The input data set has an LRECL of 40 and the SPCL data set will have an LRECL of 42.

The Y2PAST=1996 option sets the century window to 1996-2095. The century window will be used to transform the two-digit years into four-digit years, but will not be used for the special blank indicators.

The input records might be as follows:

```
MORGAN HILL                   CA
SAN JOSE             960512   CA
BOCA RATON           000628   FL
DENVER               951115   CO
```

The output records would be as follows:

```
MORGAN HILL                     CA
SAN JOSE             19960512   CA
BOCA RATON           20000628   FL
DENVER               20951115   CO
```

Example 10

```
OPTION COPY
OUTFIL FNAMES=ALL,OUTREC=(C'US ',1,10,C' is in ',11,15,/,
                          C'WW ',1,10,C' is in ',26,20,2/)
OUTFIL FNAMES=(US,WW),SPLIT,
                      OUTREC=(1,10,C' is in ',11,15,/,
                              1,10,C' is in ',26,20)
```

This example illustrates how multiple OUTFIL output and blank records can be produced from each OUTFIL input record. The input data set has an LRECL of 50 and contains the following three records:

```
Finance   San Francisco  Buenos Aires
Research  New York       Amsterdam
Marketing Los Angeles    Mexico City
```

The first OUTFIL statement creates the data set associated with ddname ALL. This data set will have an LRECL of 40 (the length of the longest output record; the one that includes the 26,20 input field). Each input record will result in two data records followed by two blank records as follows:

`ALL data set`

```
US Finance    is in San Francisco
WW Finance    is in Buenos Aires


US Research   is in New York
WW Research   is in Amsterdam


US Marketing  is in Los Angeles
WW Marketing  is in Mexico City
```

The second OUTFIL statement creates the two data sets associated with ddnames US and WW. These data sets will have an LRECL of 37 (the length of the longest output record; the one that includes the 26,20 input field). Each input record will result in two data records. SPLIT will cause the first data record

to be written to the US data set and the second data record to be written to the WW data set. Thus, each input record will create one record in each OUTFIL data set as follows:

US data set

```
Finance    is in San Francisco
Research   is in New York
Marketing  is in Los Angeles
```

WW data set

```
Finance    is in Buenos Aires
Research   is in Amsterdam
Marketing  is in Mexico City
```

Example 11

```
  SORT FIELDS=(6,3,CH,D)
  OUTFIL FNAMES=SET60,BUILD=(1,60),VLFILL=C' '
  OUTFIL FNAMES=VARFIX,VTOF,BUILD=(5,20,5X,28,20),VLFILL=C'*'
```

This example illustrates how variable-length records that are too short to contain all OUTFIL BUILD fields can be processed successfully.

The input data set has RECFM=VB and LRECL=80. The records in this data set have lengths that vary from 15 bytes to 75 bytes.

The first OUTFIL statement creates the data set associated with ddname SET60. This data set will have RECFM=VB and LRECL=60. Every record in this data set will have a length of 60. The 1,60 field truncates records longer than 60 bytes to 60 bytes. Because VLFILL=C' ' is specified, the 1,60 field pads records shorter than 60 bytes to 60 bytes using a blank (C' ') as the fill byte.

Note: Without VLFILL=byte, this OUTFIL statement would terminate with an ICE218A message because some of the input records are too short to contain the BUILD field.

The second OUTFIL statement creates the data set associated with ddname VARFIX. This data set will have RECFM=FB and LRECL=45. VTOF changes the variable-length input records to fixed-length output records according to the fields specified by BUILD. VLFILL=C'*' allows short input records to be processed. Each missing byte in an OUTFIL BUILD field is replaced with an asterisk (C'*') fill byte.

Note:

1. CONVERT can be used instead of VTOF.
2. VLFILL=C'*' overrides the default of VLFILL=X'40' for VTOF or CONVERT.

Example 12

```
OPTION COPY
  OUTFIL OUTREC=(SEQNUM,4,BI,Z,8,5,ZD,TO=PD,Z,
         31,2,PD,TO=FI,LENGTH=2),Z,
         16,3,ZD,ADD,+1,TO=FI,LENGTH=2,Z,
         (16,3,ZD,MAX,31,2,PD),MUL,+2,TO=ZD,LENGTH=4)
```

This example illustrates how a sequence number can be generated, how values in one numeric format can be converted to another numeric format, and how arithmetic expressions involving fields and decimal constants can be used.

The input data set has an LRECL of 50 and the SORTOUT data set will have an LRECL of 19.

The OUTFIL statement creates output records with the following fields:

- A binary sequence number in bytes 1-4 that starts at 1 and increments by 1.
- X'00' in byte 5.
- A PD field in bytes 6-8 containing the converted ZD field from input bytes 8-12
- X'00' in position 9.

- An FI field in bytes 10-11 containing the converted PD field from input bytes 31-32.
- X'00' in position 12.
- An FI field in bytes 13-14 containing the converted result of the ZD field from input bytes 16-18 incremented by 1.
- X'00' in position 15.
- A ZD field in bytes 16-19 containing the converted result of the maximum of the ZD field from input bytes 16-18 and the PD field from input bytes 31-32, multiplied by 2.

Example 13

```
SORT FIELDS=COPY
OUTFIL FNAMES=VAROUT1,FTOV
OUTFIL FNAMES=VAROUT2,FTOV,
  OUTREC=(20,8,35,10)
OUTFIL FNAMES=VAROUT3,FTOV,VLTRIM=X'40'
```

This example illustrates several ways to convert a fixed-length record data set to a variable-length record data set using the FTOV parameter of OUTFIL.

The input data set has an RECFM=FB and LRECL=60.

- The first OUTFIL statement converts the fixed-length input data set to a variable-length OUTFIL data set associated with ddname VAROUT1. VAROUT1 will have RECFM=VB and LRECL=64. All of its records will be 64 bytes long (4-byte RDW plus 60-byte input record).
- The second OUTFIL statement converts the fixed-length input data set to a variable-length OUTFIL data set associated with ddname VAROUT2. OUTREC is used to select two input fields for the output records, bytes 20-27 and bytes 35-44. VAROUT2 will have RECFM=VB and LRECL=22. All of its records will be 22 bytes long (4-byte RDW plus 8-byte input field plus 10-byte input field).
- The third OUTFIL statement converts the fixed-length input data set to a variable-length OUTFIL data set associated with ddname VAROUT3. VAROUT3 will have RECFM=VB and LRECL=64. VLTRIM=X'40' is used to remove the trailing blanks from the variable-length output records. The records can vary from 5 bytes long to 64 bytes long depending on the number of trailing blanks in each record.

Example 14

```
OPTION COPY
OUTFIL FNAMES=OUT1,OUTREC=(DATE1(/),X,TIME1(:),X,1,80)
OUTFIL FNAMES=OUT2,OUTREC=(DATE2P,TIME3P,1,80)
OUTFIL FNAMES=OUT3,OUTREC=(DATE3(.),X,TIME2,X,1,80)
```

This example illustrates several different ways to insert timestamps into your records.

The input data set has RECFM=FB and LRECL=80.

The first OUTFIL statement creates the data set associated with ddname OUT1. This data set will have LRECL=100. Each output record will have a timestamp consisting of the date and time of the run in the form C'yyyy/mm/dd hh:mm:ss ' (20 bytes), followed by the original input record (80 bytes).

The second OUTFIL statement creates the data set associated with ddname OUT2. This data set will have LRECL=86. Each output record will have a timestamp consisting of the date of the run in the form P'yyyymm' (4 bytes) and the time of the run in the form P'hh' (2 bytes), followed by the original input record (80 bytes).

The third OUTFIL statement creates the data set associated with ddname OUT3. This data set will have LRECL=94. Each output record will have a timestamp consisting of the date and time of the run in the form C'yyyy.ddd hhmm ' (14 bytes), followed by the original input record (80 bytes).

Example 15

```
OPTION COPY
OUTREC FIELDS=(1,4,11,4,DT1,7,4,TM1,60:X)
```

```
OUTFIL NODETAIL,
 TRAILER1=(//,
  3:'Earliest SMF timestamp is  ',
     MIN=(5,14,ZD,EDIT=('TTTT/TT/TT TT:TT:TT')),/,
  3:'Latest SMF timestamp is    '
     MAX=(5,14,ZD,EDIT=('TTTT/TT/TT TT:TT:TT')))
```

This example illustrates how the earliest and latest timestamps from a set of SMF records can be displayed.

The OUTREC statement uses the DT1 format to convert the SMF date in input bytes 11-15 to a Z'yyyymmdd' value in bytes 5-12, and uses the TM1 format to convert the SMF time in input bytes 7-10 to a Z'hhmmss' value in bytes 13-18.

The OUTFIL statement uses the Z'yyyymmddhhmmss' value created by OUTREC in bytes 5-18 to determine the minimum (earliest) and maximum (latest) timestamp, and displays those timestamps in a trailer record in the form C'yyyy/mm/dd hh:mm:ss'.

The report might look as follows:

```
 Earliest SMF timestamp is  2001/01/09 10:27:04
 Latest SMF timestamp is    2001/04/24 06:13:22
```

Example 16

```
SORT FIELDS=(1,20,BI,A)
OUTFIL FNAMES=FUPPER,OUTREC=(1,80,TRAN=LTOU)
OUTFIL FNAMES=FHEX,OUTREC=(1,80,HEX)
```

This example illustrates two types of conversion for fixed length records: lowercase to uppercase conversion and hex conversion.

The input data set has RECFM = FB and LRECL = 80.

The first OUTFIL statement creates the data set associated with ddname FUPPER. This data set will have RECFM = FB and LRECL = 80. All of the lowercase EBCDIC characters (a-z) from byte 1 to byte 80 will be converted to uppercase EBCDIC characters (A-Z). Other characters will remain unchanged. For example, the characters 'san jose, ca 95193' will be converted to 'SAN JOSE, CA 95193'.

The second OUTFIL statement creates the data set associated with ddname FHEX. This data set will have RECFM = FB and LRECL = 160 (2 * 80 data bytes). Each byte from 1 to 80 will be converted to the two bytes representing its hex value. For example, the three characters 'A12' will be converted to the six characters 'C1F1F2'.

Example 17

```
OPTION COPY
OUTFIL FNAMES=VUPPER,OUTREC=(1,4,5,TRAN=UTOL)
OUTFIL FNAMES=VHEX,OUTREC=(1,4,5,HEX)
```

This example illustrates two types of conversion for variable-length records: uppercase to lowercase conversion and hex conversion.

The input data set has RECFM = VB and LRECL = 5000.

The first OUTFIL statement creates the data set associated with ddname VUPPER. This data set will have RECFM = VB and LRECL = 5000. All of the uppercase EBCDIC characters (A-Z) from bytes 5 (after the RDW) to the end of each record will be converted to lowercase EBCDIC characters (a-z). Other characters will remain unchanged. For example, the characters 'SAN JOSE, CA 95193' will be converted to 'san jose, ca 95193'.

The second OUTFIL statement creates the data set associated with ddname VHEX. This data set will have RECFM = VB and LRECL = 9996 (4 for RDW plus 2 * 4996 data bytes). Each byte from 5 (after the RDW) to the end of each record will be converted to the two bytes representing its hex value. For example, the three characters 'A12' will be converted to the six characters 'C1F1F2'.

Example 18

```
SORT FIELDS=(22,8,CH,A)
OUTFIL FNAMES=SAMP1,REMOVECC,HEADER1=('Sample 1',2/),
  STARTREC=120,SAMPLE=(20,4),
  OUTREC=(1,8,9,5,PD,ZD,C'***',14,50)
OUTFIL FNAMES=SAMP2,REMOVECC,HEADER1=('Sample 2',2/),
  SAMPLE=1000,ENDREC=5001
```

This example illustrates how to take different "samples" of sorted output records. Sorted records 120, 121, 122, 123, 140, 141, 142, 143, and so on will be reformatted as indicated by the OUTREC parameter and written to the output data set associated with SAMP1. The heading 'Sample 1' will appear before the sample output records.

Sorted records 1, 1001, 2001, 3001, 4001 and 5001 will be written to the output data set associated with SAMP2. The heading 'Sample 2' will appear before the sample output records.

Example 19

```
OPTION COPY
OUTFIL FNAMES=R500,REPEAT=500
OUTFIL FNAMES=R100,OUTREC=(SEQNUM,4,ZD,80X'FF'),REPEAT=100
```

This example illustrates how one record can be used to generate many records.

The first OUTFIL statement writes each output record 500 times to the data set associated with R500. Each set of 500 records will be identical.

The second OUTFIL statement writes each reformatted output record 100 times to the data set associated with R100. Each set of 100 records will be identical except for the sequence number. The 100 records written from the first output record will have sequence numbers 1-100, the 100 records written from the second output record will have sequence numbers 101-200, and so on.

Example 20

```
OPTION COPY
OUTFIL OMIT=(56,6,CH,EQ,C'******'),
  OVERLAY=(121:SEQNUM,8,ZD,56:56,6,TRAN=UTOL)
```

This example illustrates how you can use the OVERLAY parameter with OUTFIL to add sequence numbers at the end of your records, and to convert uppercase EBCDIC characters to lowercase EBCDIC characters in certain columns, without affecting the rest of the record.

The input data set has RECFM=FB and LRECL=120. The output data set will have RECFM=FB and LRECL=128.

The OMIT parameter removes records which have asterisks in positions 56-61.

121:SEQNUM,8,ZD in the OVERLAY parameter adds an 8-byte sequence number in positions 121-128 of every record. The LRECL is increased from 120 to 128 to hold the sequence number.

56:56,6,TRAN=UTOL in the OVERLAY parameter converts uppercase EBCDIC characters in positions 56-61 to lowercase EBCDIC characters in positions 56-61.

Only the two overlaid fields are changed; all of the other data in the records is unaffected.

Example 21

```
OPTION COPY
OUTFIL FNAMES=RPT1,
  HEADER2=(1:'Type',9:'Date',21:'Time',/,
           1:'----',9:'--------',21:'------'),
  IFTHEN=(WHEN=(5,2,BI,EQ,X'0001'),
    BUILD=(1,4,5:5,2,BI,M11,LENGTH=4,
           13:8,4,DT1,25:15,4,TM1)),
  IFTHEN=(WHEN=(5,2,BI,EQ,X'0002'),
    BUILD=(1,4,5:5,2,BI,M11,LENGTH=4,
```

```
             13:12,8,DC1,25:12,8,TC1)),
    IFTHEN=(WHEN=(5,2,BI,EQ,X'0003'),
      BUILD=(1,4,5:5,2,BI,M11,LENGTH=4,
             13:17,8,DE1,25:17,8,TE1)),
    IFTHEN=(WHEN=NONE,
      BUILD=(1,4,5:5,2,BI,M11,LENGTH=4,
             13:C'n/a',25:C'n/a'))
  OUTFIL FNAMES=RPT2,
    HEADER2=(1:'Type',9:'Date',21:'Time',/,
             1:'----',9:'-------',21:'--------'),
    IFTHEN=(WHEN=(5,2,BI,EQ,X'0001'),
      BUILD=(1,4,5:5,2,BI,M11,LENGTH=4,
            13:8,4,DT3,25:15,4,TM4)),
    IFTHEN=(WHEN=(5,2,BI,EQ,X'0002'),
      BUILD=(1,4,5:5,2,BI,M11,LENGTH=4,
            13:12,8,DC3,25:12,8,TC4)),
    IFTHEN=(WHEN=(5,2,BI,EQ,X'0003'),
      BUILD=(1,4,5:5,2,BI,M11,LENGTH=4,
            13:17,8,DE3,25:17,8,TE4)),
    IFTHEN=(WHEN=NONE,
      BUILD=(1,4,5:5,2,BI,M11,LENGTH=4,
            13:C'n/a',25:C'n/a'))
```

This example illustrates how you can use IFTHEN clauses to reformat different records in different ways.

The input data set has RECFM=VB and consists of several different types of input records as follows:

- Type1: Has X'0001' in positions 5-6, a 4-byte SMF date in positions 8-11 and a 4-byte SMF time in positions 15-18.
- Type2: Has X'0002' in positions 5-6 and an 8–byte TOD date and time in positions 12–19.
- Type3: Has X'0003' in positions 5-6 and an 8–byte ETOD date and time in positions 17–24.
- Other types: Do not have X'0001', X'0002' or X'0003' in positions 5-6, and do not have a date or time value.

The first OUTFIL statement produces a report with RECFM=VBA and LRECL=31 that might look like this:

```
Type    Date        Time
----    --------    ------
0001    20040827    124531
0003    20040907    230603
0008    n/a         n/a
0002    20040901    061559
0001    20040731    152201
0004    n/a         n/a
```

The first IFTHEN clause operates only against Type1 records; it converts the SMF date using DT1 and the SMF time using TM1. The second IFTHEN clause only operates against Type2 records; it converts the TOD date using DC1 and the TOD time using TC1. The third IFTHEN clause only operates against Type3 records; it converts the ETOD date using DE1 and the ETOD time using TE1. The fourth IFTHEN clause operates against all other types of records; it uses 'n/a' for the date and time.

The second OUTFIL statement produces a report with RECFM=VBA and LRECL=33 that looks like this:

```
Type    Date        Time
----    -------     --------
0001    2004240     12453184
0003    2004251     23060373
0008    n/a         n/a
0002    2004245     06155903
0001    2004213     15220150
0004    n/a         n/a
```

The first IFTHEN clause operates only against Type1 records; it converts the SMF date using DT3 and the SMF time using TM4. The second IFTHEN clause operates only against Type2 records; it converts the TOD date using DC3 and the TOD time using TC4. The third IFTHEN clause operates only against Type3 records; it converts the ETOD date using DE3 and the ETOD time using TE4. The fourth IFTHEN clause operates against all other types of records; it uses 'n/a' for the date and time.

Example 22

```
  OPTION COPY
  OUTFIL BUILD=(1,80,SQZ=(SHIFT=LEFT,PAIR=APOST,MID=C','))
```

This example illustrates how you can create FB output records with comma separated values from FB input records containing fields in fixed positions.

The 80-byte FB input records might look like this:

```
  'John Lewis'         -12.83  'Research'
  'Ted Blank'         +128.37  'Manufacturing'
  'Marilyn Carlson'   -282.83  'Technical Support'
  'Rex Otis'            +2.83  'Marketing'
```

Note that the data has three fields in fixed positions. The first field is a character string surrounded by apostrophes. The second field is a numeric value. The third field is another character string surrounded by apostrophes.

The 80-byte FB output records are in the form of comma separated values as follows:

```
'John Lewis',-12.83,'Research'
'Ted Blank',+128.37,'Manufacturing'
'Marilyn Carlson',-282.83,'Technical Support'
'Rex Otis',+2.83,'Marketing'
```

We use OUTFIL BUILD to build the output records with comma separated values. We use SQZ to squeeze out the blanks between the fields, shift the remaining characters to the left and insert commas between the fields. SHIFT=LEFT shifts the characters to the left. PAIR=APOST ensures that blanks within paired apostrophes are not squeezed out (for example, we want to keep the blank in the 'John Lewis' field.) MID=C',' inserts a comma for each group of blanks removed between the fields (for example, between 'John Lewis' and -12.83).

Example 23

```
   OPTION COPY
   OUTFIL IFTHEN=(WHEN=INIT,
     OVERLAY=(5:5,18,JFY=(SHIFT=LEFT,LEAD=C'''',TRAIL=C''''),
              34:34,19,JFY=(SHIFT=LEFT,LEAD=C'''',TRAIL=C''''))),
    IFTHEN=(WHEN=INIT,
     BUILD=(1,4,5,60,SQZ=(SHIFT=LEFT,PAIR=APOST,MID=C','))),
    VLTRIM=C' '
```

This example illustrates how you can create VB output records with comma separated values from VB input records containing fields in fixed positions.

The 84-byte VB input records might look like this (4-byte RDW followed by data):

```
Length   |  Data
    45       John Lewis          -12.83  Research
    50       Ted Blank          +128.37  Manufacturing
    54       Marilyn Carlson    -282.83  Technical Support
    51       Rex Otis             +2.83  Marketing
```

Note that the data has three fields in fixed positions. The first field is a character string. The second field is a numeric value. The third field is another character string. (In the previous example, the character strings were surrounded by apostrophes; in this example the apostrophes must be added around the character strings.)

The 84-byte VB output records are in the form of comma separated values as follows:

```
Length   |  Data
    34      'John Lewis',-12.83,'Research'
    39      'Ted Blank',+128.37,'Manufacturing'
    49      'Marilyn Carlson',-282.83,'Technical Support'
    32      'Rex Otis',+2.83,'Marketing'
```

We use OUTFIL IFTHEN to ensure that short records are padded on the right with blanks. (OUTFIL BUILD would terminate due to the short records.) WHEN=INIT reformats every record. OVERLAY surrounds the character strings with apostrophes. BUILD builds the output records with comma separated values.

We use JFY to surround the character strings with apostrophes without removing embedded blanks (for example, John Lewis is changed to 'John Lewis'). After the first IFTHEN, the records look like this:

```
Length   |  Data
    52      'John Lewis'          -12.83 'Research'
    52      'Ted Blank'          +128.37 'Manufacturing'
    54      'Marilyn Carlson'    -282.83 'Technical Support'
    52      'Rex Otis'             +2.83 'Marketing'
```

We then use SQZ to squeeze out the blanks between the fields, shift the remaining characters to the left and insert commas between the fields. SHIFT=LEFT shifts the characters to the left. PAIR=APOST ensures that blanks within paired apostrophes are not squeezed out (for example, we want to keep the blank in the 'John Lewis' field.) MID=C',' inserts a comma for each group of blanks removed between the fields (for example, between 'John Lewis' and -12.83).

Finally, we use VLRTIM=C' ' to remove trailing blanks at the end of each VB record by adjusting its length appropriately.

Example 24

```
  OPTION COPY
  OUTFIL REMOVECC,
   HEADER1=(C'<?xml version="1.0"?>',/,
     3:C'<booklist>'),
   BUILD=(5:C'<book>',/,
     7:1,20,JFY=(SHIFT=LEFT,LEAD=C'<title>',TRAIL=C'</title>',
       LENGTH=36),/,
     7:24,15,SQZ=(SHIFT=LEFT,LEAD=C'<author>',MID=C', ',
       TRAIL=C'</author>',LENGTH=33),/,
     5:C'</book>'),
   TRAILER1=(3:C'</booklist>')
```

This example illustrates how you can generate XML statements from FB input records containing fields in fixed positions.

The 40-byte FB input records might look like this:

```
Modern Poetry          Friedman    KR
Intro to Computers     Chatterjee  CL
Marketing              Maxwell     G
```

Note that the data has three character fields in fixed positions.

The 42-byte FB output records look like this:

```
<?xml version="1.0"?>
  <booklist>
    <book>
      <title>Modern Poetry</title>
      <author>Friedman, KR</author>
    </book>
    <book>
      <title>Intro to Computers</title>
      <author>Chatterjee, CL</author>
    </book>
    <book>
      <title>Marketing</title>
      <author>Maxwell, G</author>
    </book>
  </booklist>
```

We use OUTFIL HEADER1 to generate the xml and booklist starting tags that precede the set of tags for each record. We use OUTFIL BUILD to generate the set of tags for each record as follows:

- A constant is used to generate the book starting tag.

- JFY is used to generate the title tags and data from the first input field. LEAD generates the title starting tag before the input field from the record. TRAIL generates the title ending tag after the last nonblank character from the input field. JFY keeps embedded blanks between the first nonblank character and the last nonblank character of the input field. LENGTH ensures that the addition of the LEAD and TRAIL strings does not cause truncation by increasing the output length to 36 bytes (overriding the default of 20 bytes from the input field).
- SQZ is used to generate the author tags and data from the second and third input fields. LEAD generates the author starting tag before the input field from the record. MID replaces the blanks between the second and third input fields with a comma and one blank. TRAIL generates the author ending tag after the last nonblank character from the input fields. LENGTH ensures that the addition of the LEAD, MID and TRAIL strings does not cause truncation by increasing the output length to 33 bytes (overriding the default of 15 bytes from the input field).

We use OUTFIL TRAILER1 to generate the booklist ending tag that follows the set of tags for each record.

Example 25

```
  OPTION COPY
  OUTFIL HEADER2=(1:'First',13:'Initial',22:'Last',37:'Score',/,
                  1:10'-',13:7'-',22:11'-',35:7'-'),
     PARSE=(%00=(STARTAFT=C'LAST-> ',ENDBEFR=C' ',FIXLEN=11),
            %01=(STARTAFT=C'FIRST-> ',ENDBEFR=C' ',FIXLEN=11),
            %02=(STARTAFT=C'INITIAL-> ',ENDBEFR=C' ',FIXLEN=7),
            %03=(STARTAFT=C'SCORE-> ',FIXLEN=7)),
     BUILD=(1:%01,13:%02,22:%00,
            35:%03,SFF,EDIT=(SIIIT.T),SIGNS=(,-))
```

This example illustrates how you can create a report from FB input records with variable position/length fields, such as keyword delimited values.

The 70-byte input records might look like this:

```
     LAST-> Clark   FIRST-> Oscar     INITIAL-> D   SCORE-> +98.2
  LAST-> Roberts  FIRST-> Harriet   INITIAL->    SCORE-> -152.6
   LAST-> Stein FIRST-> Gertrude INITIAL-> V SCORE-> +5.1
```

Note that each record has four variable fields each identified by a specific keyword. The fields do not start and end in the same position in every record and they have different lengths in different records.

The output report has RECFM=FBA and LRECL=42 and looks like this:

```
First       Initial  Last           Score
----------  -------  -----------  -------
Oscar       D        Clark          98.2
Harriet              Roberts      -152.6
Gertrude    V        Stein            5.1
```

In order to create a report like this from variable position/length fields, we use HEADER2 to create the page header, and PARSE and BUILD to create a fixed parsed field from each variable field. We use %00 to create an 11-byte fixed parsed field with the extracted 'LAST-> ' value. We use %01 to create an 11-byte fixed parsed field with the extracted 'FIRST-> ' value. We use %02 to create a 7-byte fixed parsed field with the extracted 'INITIAL-> ' value. We use %03 to create a 7-byte fixed parsed field with the extracted 'SCORE-> ' value. Since the fourth field (%03) is extracted as signed and padded on the right with blanks, we treat it as SFF format in order to edit it.

Example 26

```
  INREC IFTHEN=(WHEN=INIT,
     PARSE=(%1=(FIXLEN=10,STARTAFT=C'First="',ENDBEFR=C'"')),
    IFTHEN=(WHEN=INIT,
     PARSE=(%2=(FIXLEN=10,STARTAFT=C'Middle="',ENDBEFR=C'"')),
    IFTHEN=(WHEN=INIT,
     PARSE=(%3=(FIXLEN=10,STARTAFT=C'Last="',ENDBEFR=C'"')),
    IFTHEN=(WHEN=INIT,
     PARSE=(%4=(FIXLEN=10,STARTAFT=C'Wife="',ENDBEFR=C'"')),
     BUILD=(%1,13:%2,25:%3,37:%4))
```

```
  SORT FIELDS=(25,10,CH,A,1,10,CH,A,13,10,CH,A)
  OUTFIL HEADER2=('First',13:'Middle',
      25:C'Last',37:'Wife',/,10C'-',13:10C'-',
      25:10C'-',37:10C'-')
```

This example illustrates how you can create a sorted report from FB input records with keyword values that can occur in any order or not occur at all.

The 80-byte input records might look like this:

```
   Last="Buchanan" First="James"
  Wife="Louisa" First="John" Middle="Quincy" Last="Adams"
   First="George" Last="Washington" Wife="Martha"
Last="Clinton" Wife="Hillary" Middle="Jefferson" First="William"
  First="John" Wife="Abigail" Last="Adams"
```

Note that each record has up to four variable fields each identified by a specific keyword (First, Middle, Last, Wife). In each record, the fields can be in any order, do not start and end in the same position and can have different lengths.

We want to sort the records by last name, first name and middle name, and create a report with fixed headings and values for the first name, middle name, last name and wife's name that looks like this:

```
First       Middle      Last        Wife
----------  ----------  ----------  ----------
John                    Adams       Abigail
John        Quincy      Adams       Louisa
James                   Buchanan
William     Jefferson   Clinton     Hillary
George                  Washington  Martha
```

In order to extract and sort variable fields like these, we use a separate IFTHEN PARSE clause for each keyword. This allows us to find a particular keyword anywhere in the record or ignore it if it is not in the record. If we used PARSE instead of IFTHEN PARSE, a missing keyword would cause any keywords that followed to be ignored. But because each IFTHEN PARSE starts scanning at position 1 by default, we can look for each keyword independently of the others. If a keyword is missing, the parsed field is set to all blanks (for example, %2 and %4 are set to blanks for the James Buchanan record).

We use %1 to create a 10-byte fixed parsed field into which we extract the first name. We use %2 to create a 10-byte fixed parsed field into which we extract the middle name. We use %3 to create a 10-byte fixed parsed field into which we extract the last name. We use %4 to create a 10-byte fixed parsed field into which we extract the wife's name. We use BUILD to reformat the records to contain the fixed parsed fields. Then we SORT on the fixed parsed fields. Finally, we use OUTFIL to create the headings.

Example 27

```
  OPTION COPY
  OUTFIL FINDREP=(INOUT=(C'AM',C'IN THE MORNING',
    C'PM',C'IN THE EVENING'),MAXLEN=70)
```

This example illustrates how you can replace values in FB or VB records with larger values and shift the rest of the bytes to the right.

The 40 byte FB input records might look like this:

```
COFFEE AT 12:28 AM, TOAST AT 06:15 PM
MILK AT 03:17 PM, BAGELS AT 05:03 PM
PUDDING AT 09:32 AM
```

We want to replace each instance of 'AM' with 'IN THE MORNING' and each instance of 'PM' with 'IN THE EVENING' and shift the bytes after 'AM' or 'PM' to the right. We use INOUT to indicate find and replace pairs. Since replacing the smaller values with the larger values can cause the remaining bytes to be shifted beyond the end of the 40 byte record, we use MAXLEN to set the output record to 70 bytes to allow for the expansion, overriding the default of using the input length for the output record.

The 70 byte FB output records look like this:

```
COFFEE AT 12:28 IN THE MORNING, TOAST AT 06:15 IN THE EVENING
MILK AT 03:17 IN THE EVENING, BAGELS AT 05:03 IN THE EVENING
PUDDING AT 09:32 IN THE MORNING
```

Example 28

```
  OPTION COPY
  OUTREC IFTHEN=(WHEN=INIT,BUILD=(1,4,8:5)),
    IFTHEN=(WHEN=GROUP,BEGIN=(8,5,CH,EQ,C'PAGE:'),
    PUSH=(5:SEQ=3))
  OUTFIL INCLUDE=(5,3,ZD,EQ,2,OR,5,3,ZD,EQ,3),
    BUILD=(1,4,5:8)
```

This example illustrates how you can INCLUDE specific relative records from groups of VB records. We insert a sequence number between the RDW and the first data byte of each record. The sequence number restarts at 1 for the first record in each group. We INCLUDE on the sequence number and then remove it.

The VB input records might look like this:

```
Len|Data
 12|PAGE:  1
 22|LINE 1 OF REPORT A
 22|LINE 2 OF REPORT A
 22|LINE 3 OF REPORT A
 22|LINE 4 OF REPORT A
   |...
 12|PAGE:  2
 23|LINE 66 OF REPORT A
 23|LINE 67 OF REPORT A
 23|LINE 68 OF REPORT A
 23|LINE 69 OF REPORT A
   |...
 12|PAGE:  3
 24|LINE 131 OF REPORT A
 24|LINE 132 OF REPORT A
 24|LINE 133 OF REPORT A
 24|LINE 134 OF REPORT A
   |...
```

We use an IFTHEN WHEN=INIT clause to reformat each record so it has room for the 3-byte sequence number between the RDW and the first data byte. After the WHEN=INIT clause is executed, the intermediate records look like this:

```
Len|Data
 15|   PAGE:  1
 25|   LINE 1 OF REPORT A
 25|   LINE 2 OF REPORT A
 25|   LINE 3 OF REPORT A
 25|   LINE 4 OF REPORT A
   |   ...
 15|   PAGE:  2
 26|   LINE 66 OF REPORT A
 26|   LINE 67 OF REPORT A
 26|   LINE 68 OF REPORT A
 26|   LINE 69 OF REPORT A
   |   ...
 15|   PAGE:  3
 27|   LINE 131 OF REPORT A
 27|   LINE 132 OF REPORT A
 27|   LINE 133 OF REPORT A
 27|   LINE 134 OF REPORT A
```

Note that positions 5-7 are blank and the 'PAGE:' characters have been shifted over to positions 8-12.

We use an IFTHEN WHEN=GROUP clause to put a 3-byte sequence number in each record. BEGIN indicates a group starts with a record that has 'PAGE:' in positions 8-12. PUSH overlays a 3-byte sequence number at positions 5-7 of each record. The sequence number starts at 1 for the first record of a group and is incremented by 1 for each subsequent record of the group. After the IFTHEN GROUP clause is executed, the intermediate records look like this:

```
Len|Data
 15|001PAGE:  1
```

```
 25|002LINE 1 OF REPORT A
 25|003LINE 2 OF REPORT A
 25|004LINE 3 OF REPORT A
 25|005LINE 4 OF REPORT A
   |006...
 15|001PAGE:  2
 26|002LINE 66 OF REPORT A
 26|003LINE 67 OF REPORT A
 26|004LINE 68 OF REPORT A
 26|005LINE 69 OF REPORT A
   |006...
 15|001PAGE:  3
 27|002LINE 131 OF REPORT A
 27|003LINE 132 OF REPORT A
 27|004LINE 133 OF REPORT A
 27|005LINE 134 OF REPORT A
   |006...
```

Note that the records in each group are numbered starting with 001 for the first record of each group. The records we want have sequence numbers 002 and 003.

We use an OUTFIL statement to only INCLUDE the records with sequence number 002 or 003, and to remove the sequence numbers so the included output records will be identical to the input records. After the OUTFIL statement is executed, the final output records look like this:

```
Len|Data
 22|LINE 1 OF REPORT A
 22|LINE 2 OF REPORT A
 23|LINE 66 OF REPORT A
 23|LINE 67 OF REPORT A
 24|LINE 131 OF REPORT A
 24|LINE 132 OF REPORT A
```

Example 29

```
  OPTION COPY
  OUTFIL FNAMES=OUT1,INCLUDE=(11,3,CH,EQ,C'D51'),ACCEPT=3
  OUTFIL FNAMES=OUT2,STARTREC=2,ACCEPT=5
```

This example illustrates how you can stop processing OUTFIL input records for various groups in different ways.

The input data set has these records:

```
HEADER  2010/06/30
FRANK     D51
ED        D52
VICKY     D51
MARTIN    D52
LILY      D50
MARC      D51
JUNE      D51
LUCY      D51
TRAILER     8
```

The first OUTFIL statement uses INCLUDE to extract the D51 records and ACCEPT=3 to only write the first three D51 records to OUT1. Thus, OUT1 will have these records:

```
FRANK     D51
VICKY     D51
MARC      D51
```

The second OUTFIL statement uses STARTREC=2 to skip the HEADER record and ACCEPT=5 to only write the next five records to OUT2. Thus, OUT2 will have these records:

```
FRANK     D51
ED        D52
VICKY     D51
MARTIN    D52
LILY      D50
```

Example 30

```
  OPTION COPY
  OUTFIL FNAMES=OUT1,
   INCLUDE=(3,4,CH,EQ,C'key1'),
   IFTRAIL=(HD=YES,TRLID=(1,1,CH,EQ,C'T'),
    TRLUPD=(6:COUNT=(M11,LENGTH=4),
           14:TOT=(10,4,ZD,TO=ZD,LENGTH=6)))
  OUTFIL FNAMES=OUT2,
   INCLUDE=(3,4,CH,EQ,C'key2'),
   IFTRAIL=(HD=YES,TRLID=(1,1,CH,EQ,C'T'),
    TRLUPD=(6:COUNT=(M11,LENGTH=4),
           14:TOT=(10,4,ZD,TO=ZD,LENGTH=6)))
```

This example illustrates how you can split an input file with a header, and a trailer containing a count and total for the input data records, into two output files each of which has the header and a trailer with accurate count and total for the included data records.

The input data set has RECFM=FB and LRECL=80 with these input records:

```
H 2010/07/06
D key1   0100
D key2   0200
D key2   0118
D key1   0150
D key1   0025
D key2   1000
D key2   0310
T X  0007 QR 001903  D52-007-321-7526
```

The first record is a header record with a date. The records with D in position 1 are data records; they have key1 or key2 in positions 3-6 and an amount field in positions 10-13.

The trailer record is identified by a 'T' in position 1. Positions 6-9 contain a count (0007) of the data records, and positions 14-19 contain a total (001903) for the amount fields in the data records. In addition, there are other static fields in the trailer record that must not be changed. Note that IFTRAIL with HD=YES does not process the header or trailer records as data records, so they are not included in the count or total.

The first OUTFIL statement writes the header record, data records with key1, and trailer record, in the OUT1 data set. The IFTRAIL operand is used to identify the trailer (TRLID), indicate the first record is a header (HD=YES), and update the count and total in the trailer (TRLUPD) to reflect the input data records included in this output data set.

OUT1 will have these records:

```
H 2010/07/06
D key1   0100
D key1   0150
D key1   0025
T X  0003 QR 000275  D52-007-321-7526
```

The second OUTFIL statement writes the header record, data records with key2, and trailer record, in the OUT2 data set. The IFTRAIL operand is used to identify the trailer (TRLID), indicate the first record is a header (HD=YES), and update the count and total in the trailer (TRLUPD) to reflect the input data records included in this output data set. Note that IFTRAIL with HD=YES does not process the header or trailer records as data records, so they are not included in the count or total.

OUT2 will have these records:

```
H 2010/07/06
D key2   0200
D key2   0118
D key2   1000
D key2   0310
T X  0004 QR 001628  D52-007-321-7526
```

Example 31

```
  OPTION Y2PAST=1990
  SORT FIELDS=(1,6,Y2W,A)
  OUTFIL REMOVECC,
     HEADER1=(1:'Input',15:'NEXTDFRI',25:'PREVDSUN',35:'LASTDAYQ'),
     BUILD=(1:1,6,Y2W,TOJUL=Y4W(-),
           15:1,6,Y2W,NEXTDFRI,TOJUL=Y4W(-),
           25:1,6,Y2W,PREVDSUN,TOJUL=Y4W(-),
           35:1,6,Y2W,LASTDAYQ,TOJUL=Y4W(-))
```

This example illustrates how you can sort by a date, and calculate a specific day after and before a date, and the last day of the quarter for a date. The input date is in the form C'mmddyy' and the output dates will be in the form 'ddd-yyyy'.

The SORTIN data set has these input records with a C'mmddyy' date field in positions 1-6:

```
010105
120699
021610
999999
092810
031500
000000
032505
110210
```

The SORT statement sorts by the C'mmddyy' date. We use 1,6,Y2W for the sort field to match the C'mmddyy' date.

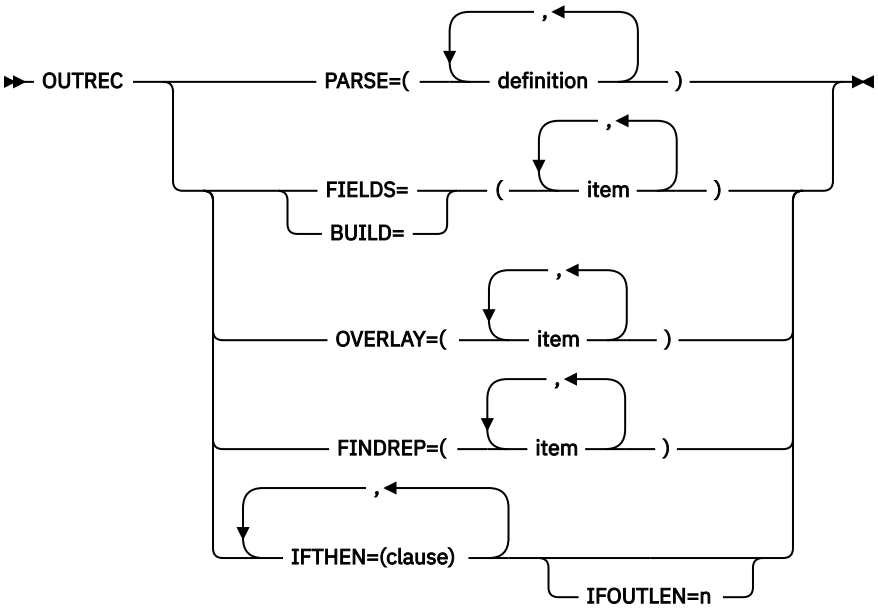
The OUTFIL statement writes a heading and performs four operations on each date. It converts the input date to C'ddd-yyyy' form, gets the next Friday date in C'ddd-yyyy' form, gets the previous Sunday date in C'ddd-yyyy' form, and gets the end of the quarter date in C'ddd-yyyy' form. We use 1,6,Y2W for the input field to match the C'mmddyy' date, and we use TOJUL=Y4W(-) to give us a C'ddd-yyyy' output date.

SORTOUT will have these records:

```
Input         NEXTDFRI  PREVDSUN  LASTDAYQ
000-0000      000-0000  000-0000  000-0000
340-1999      344-1999  339-1999  365-1999
075-2000      077-2000  072-2000  091-2000
001-2005      007-2005  361-2004  090-2005
084-2005      091-2005  079-2005  090-2005
047-2010      050-2010  045-2010  090-2010
271-2010      274-2010  269-2010  273-2010
306-2010      309-2010  304-2010  365-2010
999-9999      999-9999  999-9999  999-9999
```

Note that the '000000' and '999999' input values are treated as special indicators for output.

OUTREC control statement



The OUTREC control statement allows you to reformat the input records **after** they are sorted, merged or copied.

The OUTREC control statement supports a wide variety of parsing, editing, and reformatting tasks, including:

- The use of fixed position/length fields or variable position/length fields. For fixed fields, you specify the starting position and length of the field directly. For variable fields, such as delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings (and many other types), you define rules that allow DFSORT to extract the relevant data into fixed parsed fields, and then use the parsed fields as you would use fixed fields.
- Insertion of blanks, zeros, strings, current date, future date, past date, current time, sequence numbers, decimal constants, and the results of arithmetic expressions before, between, and after the input fields in the reformatted records.
- Sophisticated conversion capabilities, such as find and replace, hexadecimal display, bit display, translation of EBCDIC letters from lowercase to uppercase or uppercase to lowercase, translation of characters from EBCDIC to ASCII and from ASCII to EBCDIC, translation of characters using the ALTSEQ translation table, conversion of numeric values from one format to another, left-justify or left-squeeze (remove leading blanks or all blanks and shift left), and right-justify or right-squeeze (remove trailing blanks or all blanks and shift right).
- Sophisticated editing capabilities, such as control of the way numeric fields are presented with respect to length, leading or suppressed zeros, thousands separators, decimal points, leading and trailing positive and negative signs, and so on.

  Twenty-seven pre-defined editing masks are available for commonly used numeric editing patterns, encompassing many of the numeric notations used throughout the world. In addition, a virtually unlimited number of numeric editing patterns are available via the user-defined editing masks.
- Transformation of SMF, TOD, and ETOD date and time values to more usable forms.
- Various types of arithmetic operations for input date fields.
- Conversion of input date fields of one type (CH, ZD, PD, 2-digit year, 4-digit year, Julian, Gregorian) to corresponding output date fields of another type or to a corresponding day of the week.
- Selection of a character constant, hexadecimal constant, or input field from a lookup table, based on a character, hexadecimal, or bit string as input (that is, lookup and change).

You can create the reformatted OUTREC records in one of the following three ways using unedited, edited, or converted input fields (p,m for fixed fields, or %nn for parsed fields - see PARSE), and a variety of constants:

- **BUILD or FIELDS:** Reformat each record by specifying all of its items one by one. Build gives you complete control over the items you want in your reformatted OUTREC records and the order in which they appear. You can delete, rearrange and insert fields and constants. Example:

```
OUTREC BUILD=(1,20,C'ABC',26:5C'*',
     15,3,PD,EDIT=(TTT.TT),21,30,80:X)
```

- **OVERLAY:** Reformat each record by specifying just the items that overlay specific columns. Overlay lets you change specific existing columns without affecting the entire record. Example:

```
OUTREC OVERLAY=(45:45,8,TRAN=LTOU)
```

- **FINDREP:** Reformat each record by doing various types of find and replace operations. Example:

```
OUTREC FINDREP=(IN=C'Mr.',OUT=C'Mister')
```

- **IFTHEN clauses:** Reformat different records in different ways by specifying how build, overlay, find/replace, or group operation items are applied to records that meet given criteria. IFTHEN clauses let you use sophisticated conditional logic to choose how different record types are reformatted. Example:

```
OUTREC IFTHEN=(WHEN=(1,5,CH,EQ,C'TYPE1'),
         BUILD=(1,40,C'**',+1,TO=PD)),
       IFTHEN=(WHEN=(1,5,CH,EQ,C'TYPE2'),
         BUILD=(1,40,+2,TO=PD,X'FFFF')),
       IFTHEN=(WHEN=NONE,OVERLAY=(45:C'NONE'))
```

You can choose to include any or all of the following items in your reformatted OUTREC records:

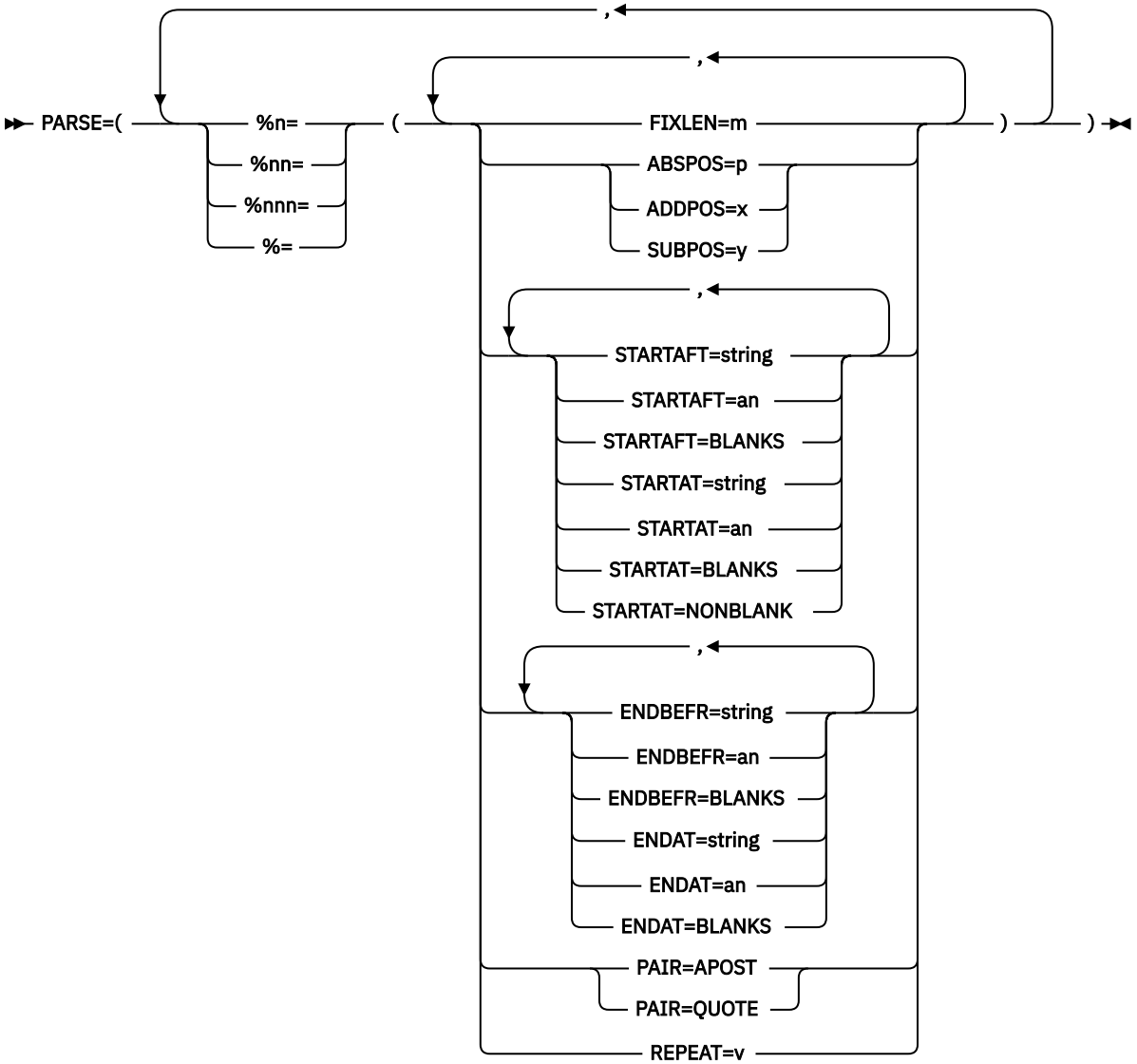
- Fixed position/length fields or variable position/length fields. For fixed fields, you specify the starting position and length of the field directly. For variable fields, such as delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings (and many other types), you define rules that allow DFSORT to extract the relevant data into fixed parsed fields, and then use the parsed fields as you would use fixed fields.
- Blanks, binary zeros, character strings, and hexadecimal strings
- Current date, future date, past date and current time in various forms
- Unedited input fields aligned on byte, halfword, fullword, and doubleword boundaries
- Replaced or removed strings.
- Hexadecimal or bit representations of binary input fields
- Characters translated from uppercase to lowercase, lowercase to uppercase, ASCII to EBCDIC or EBCDIC to ASCII.
- Left-justified, right-justified, left-squeezed, or right-squeezed input fields.
- Numeric input fields of various formats converted to different numeric formats, or to character format edited to contain signs, thousands separators, decimal points, leading zeros or no leading zeros, and so on.
- Decimal constants converted to different numeric formats, or to character format edited to contain signs, thousands separators, decimal points, leading zeros or no leading zeros, and so on.
- The results of arithmetic expressions combining fields, decimal constants, operators (MIN, MAX, MUL, DIV, MOD, ADD and SUB) and parentheses converted to different numeric formats, or to character format edited to contain signs, thousands separators, decimal points, leading zeros or no leading zeros, and so on.
- SMF, TOD and ETOD date and time fields converted to different numeric formats, or to character format edited to contain separators, leading zeros or no leading zeros, and so on.
- Input date fields of one type (CH, ZD, PD, 2-digit year, 4-digit year, Julian, Gregorian) converted to corresponding output date fields of another type or to a corresponding day of the week.

- The results of various types of arithmetic operations for input date fields.
- Sequence numbers in various formats.
- A character constant, hexadecimal constant or input field selected from a lookup table, based on a character, hexadecimal or bit constant as input.
- A zoned decimal group identifier, a zoned decimal group sequence number, or a field propagated from the first record of a group to all of the records of a group.

For information concerning the interaction of INREC and OUTREC, see "INREC statement notes" on page 150 and "OUTREC statement notes" on page 406.

See "OUTFIL control statements" on page 221 for complete details on the OUTFIL OUTREC parameter.

PARSE



This operand allows you to extract variable position/length fields into fixed parsed fields. Parsed fields (%n, %nn or %nnn) can be used where fixed position/length fields (p,m) can be used in the BUILD (or FIELDS) or OVERLAY operands as described later in this section.

Note: Although you can use %n (%0-%9), %nn (%00-%99) or %nnn (%000-%999) for a parsed field, for convenience in this book %nn will be used in general when referring to a parsed field. %n, %0n or %00n can be used interchangeably for parsed field n (for example, %1, %01 or %001 for parsed

field 1). %nn or %0nn can be used interchangeably for parsed field nn (for example, %12 or %012 for parsed field 12).

PARSE can be used for many different types of variable fields including delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, null-terminated strings, and so on. You can assign up to 1000 parsed fields (%0-%999) to the variable fields you want to extract.

Note that if all of the fields in your records have fixed positions and lengths, you don't need to use PARSE. But if any of the fields in your records have variable positions or lengths, you can use PARSE to treat them as fixed parsed fields in BUILD or OVERLAY. You can mix p,m fields (fixed fields) and %nn fields (parsed fields) in BUILD and OVERLAY.

See PARSE under "OUTFIL Control Statements" for complete details.

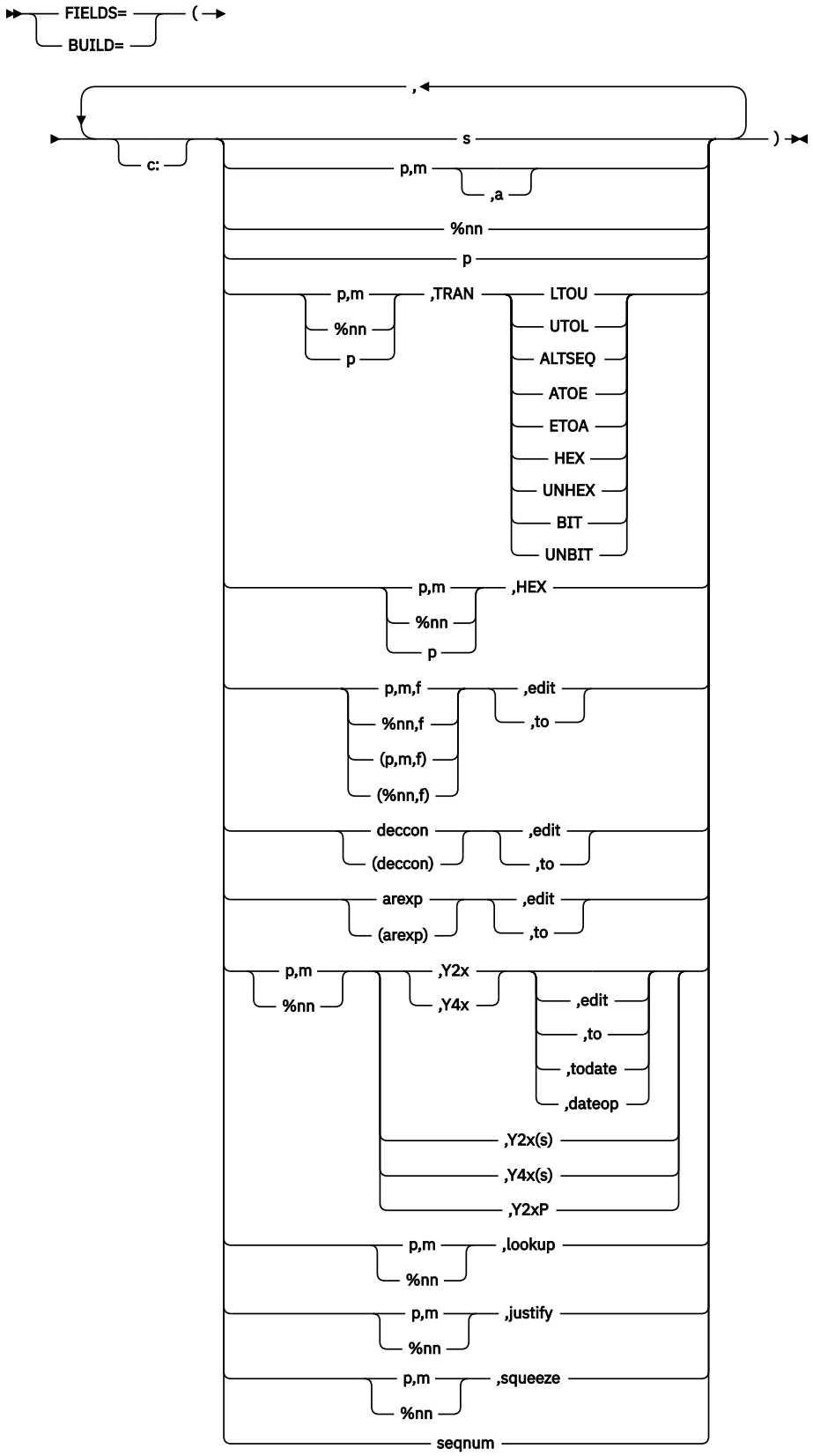
Sample Syntax:

```
OUTREC PARSE=(%200=(ENDBEFR=C'*',FIXLEN=3),
       %201=(ENDBEFR=BLANKS,FIXLEN=6),
       %202=(STARTAT=C'MAX',FIXLEN=8),
       %203=(STARTAFT=C'(',ENDBEFR=C')',FIXLEN=6),
       %204=(STARTAFT=BLANKS,FIXLEN=5)),
     BUILD=(%203,X,%203,HEX,21:%202,31:%201,SFF,M26,LENGTH=7,
            18,6,%200,UFF,M11,LENGTH=3,%204,JFY=(SHIFT=RIGHT))
```

Default for PARSE: None; must be specified. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

FIELDS or BUILD



Specifies all of the items in the reformatted OUTREC record in the order in which they are to be included. The reformatted OUTREC record consists of the separation fields, edited and unedited input fields (p,m for fixed fields, or %nn for parsed fields - see PARSE), edited decimal constants, edited

results of arithmetic expressions, and sequence numbers you select, in the order in which you select them, aligned on the boundaries or in the columns you indicate.

For variable-length records, the first item in the BUILD or FIELDS parameter must specify or include the unedited 4-byte record descriptor word (RDW), that is, you must start with 1,m with m equal to or greater than 4. If you want to include the bytes from a specific position to the end of each input record at the end of each reformatted output record, you can specify that starting position (p) as the last item in the BUILD or FIELDS parameter. For example:

```
OUTREC FIELDS=(1,4,          unedited RDW
   1,2,BI,TO=ZD,LENGTH=5,    display RDW length in decimal
   C'|',                     | separator
   5)                        display input positions 5 to end
```

For fixed-length records, the first input and output data byte starts at position 1. For variable-length records, the first input and output data byte starts at position 5, after the RDW in positions 1-4.

The BUILD or FIELDS parameter of the OUTREC statement differs from the BUILD or OUTREC parameter of the OUTFIL statement in the following ways:

- The BUILD or FIELDS parameter of the OUTREC statement applies to all input records; the BUILD or OUTREC parameter of the OUTFIL statement only applies to the OUTFIL input records for its OUTFIL group.
- The BUILD or OUTREC parameter of the OUTFIL statement supports the slash (/) separator for creating blank records and new records; the BUILD or FIELDS parameter of the OUTREC statement does not.

c:

specifies the position (column) for a separation field, input field decimal constant, arithmetic expression, or sequence number, relative to the start of the reformatted output record. Unused space preceding the specified column is padded with EBCDIC blanks. The following rules apply:

- c must be a number between 1 and 32752.
- c: must be followed by a separation field, input field, decimal constant, or arithmetic expression.
- c must not overlap the previous input field or separation field in the reformatted output record.
- For variable-length records, c: must not be specified before the first input field (the record descriptor word) nor after the variable part of the input record.
- The colon (:) is treated like the comma (,) or semicolon (;) for continuation to another line.

See Table 33 on page 132 for examples of valid and invalid column alignment.

s

specifies that a separation field (blanks, zeros, character string, hexadecimal string, current date, future date, past date, or current time) is to appear in the reformatted output record. It can be specified before or after any input field. Consecutive separation fields may be specified. For variable-length records, separation fields must not be specified before the first input field (the record descriptor word) or after the variable part of the input record. Permissible values are nX, nZ, nC'xx...x', nX'yy...yy', and various date and time constants.

nX

Blank separation. n bytes of EBCDIC blanks (X'40') are to appear in the reformatted output records. n can range from 1 to 4095. If n is omitted, 1 is used.

See Table 34 on page 133 for examples of valid and invalid blank separation.

nZ

Binary zero separation. n bytes of binary zeros (X'00') are to appear in the reformatted output records. n can range from 1 to 4095. If n is omitted, 1 is used.

See Table 35 on page 133 for examples of valid and invalid binary zero separation.

nC'xx...x'

Character string separation. n repetitions of the character string constant (C'xx...x') are to appear in the reformatted output records. n can range from 1 to 4095. If n is omitted, 1 is used. x can be any EBCDIC character. You can specify from 1 to 256 characters.

If you want to include a single apostrophe in the character string, you must specify it as two single apostrophes:

```
Required:  O'NEILL     Specify:  C'O''NEILL'
```

See Table 36 on page 134 for examples of valid and invalid character string separation.

nX'yy...yy'

Hexadecimal string separation. n repetitions of the hexadecimal string constant (X'yy...yy') are to appear in the reformatted output records. n can range from 1 to 4095. If n is omitted, 1 is used.

The value yy represents any pair of hexadecimal digits. You can specify from 1 to 256 pairs of hexadecimal digits.

See Table 37 on page 135 for examples of valid and invalid hexadecimal string separation.

DATEn, DATEn(c), DATEnP

Constant for current date. The date of the run is to appear in the reformatted output records. See DATEn, DATEn(c), DATEnP under OUTFIL OUTREC for details.

&DATEn, &DATEn(c), &DATEnP

can be used instead of DATEn, DATEn(c), and DATEnP, respectively.

DATEn+r, DATEn(c)+r, DATEnP+r

Constant for future date. A future date relative to the current date of the run is to appear in the reformatted output records. See DATEn+r, DATEn(c)+r, DATEnP+r under OUTFIL OUTREC for details.

&DATEn+r, &DATEn(c)+r, &DATEnP+r

can be used instead of DATEn+r, DATEn(c)+r, and DATEnP+r, respectively.

DATEn-r, DATEn(c)-r, DATEnP-r

Constant for past date. A past date relative to the current date of the run is to appear in the reformatted output records. See DATEn-r, DATEn(c)-r, DATEnP-r under OUTFIL OUTREC for details.

&DATEn-r, &DATEn(c)-r, &DATEnP-r

Can be used instead of DATEn-r, DATEn(c)-r, and DATEnP-r, respectively

TIMEn, TIMEn(c), TIMEnP

Constant for current time. The time of the run is to appear in the reformatted output records. See TIMEn, TIMEn(c), TIMEnP under OUTFIL OUTREC for details.

&TIMEn, &TIMEn(c), &TIMEnP

Can be used instead of TIMEn, TIMEn(c), and TIMEnP, respectively.

DATE
specifies that the current date is to appear in the reformatted output records in the form 'mm/dd/yy'. See DATE under OUTFIL OUTREC for details.

&DATE
can be used instead of DATE.

DATE=(abcd)
specifies that the current date is to appear in the reformatted output records in the form 'adbdc'. See DATE=(abcd) under OUTFIL OUTREC for details.

&DATE=(abcd)
can be used instead of DATE=(abcd).

DATENS=(abc)
specifies that the current date is to appear in the reformatted output records in the form 'abc'. See DATENS=(ab) under OUTFIL OUTREC for details.

&DATENS=(abc)
can be used instead of DATENS=(abc).

YDDD=(abc)
specifies that the current date is to appear in the reformatted output records in the form 'acb'. See YDDD=(abc) under OUTFIL OUTREC for details.

&YDDD=(abc)
can be used instead of YDDD=(abc).

YDDDNS=(ab)
specifies that the current date is to appear in the reformatted output records in the form 'ab'. See YDDDNS=(ab) under OUTFIL OUTREC for details.

&YDDDNS=(ab)
can be used instead of YDDDNS=(ab).

TIME
specifies that the current time is to appear in the reformatted output records in the form 'hh:mm:ss'. See TIME under OUTFIL OUTREC for details.

&TIME
can be used instead of TIME.

TIME=(abc)
specifies that the current time is to appear in the reformatted output records in the form 'hhcmmcss' (24-hour time) or 'hhcmmcss xx' (12-hour time). See TIME=(abc) under OUTFIL OUTREC for details.

&TIME=(abc)
can be used instead of TIME=(abc).

TIMENS=(ab)
specifies that the current time is to appear in the reformatted output record in the form 'hhmmss' (24-hour time) or 'hhmmss xx' (12-hour time). See TIMENS=(ab) under OUTFIL OUTREC for details.

&TIMENS=(ab)
can be used instead of TIMENS=(ab).

p,m,a
specifies that an unedited input field is to appear in the reformatted output record.

p
specifies the first byte of the input field relative to the beginning of the input record.[13] The first data byte of a fixed-length record has relative position 1. The first data byte of a variable-length

record has relative position 5, because the first four bytes are occupied by the RDW. All fields must start on a byte boundary, and no field may extend beyond byte 32752. See "OUTREC statement notes" on page 406 for special rules concerning variable-length records.

m
specifies the length of the input field. It must include the sign if the data is signed and must be a whole number of bytes. See "OUTREC statement notes" on page 406 for more information.

a
specifies the alignment (displacement) of the input field in the reformatted output record relative to the start of the reformatted output record.

The permissible values of **a** are:

H
Halfword aligned. The displacement (p-1) of the field from the beginning of the reformatted input record, in bytes, is a multiple of 2 (that is, position 1, 3, 5, and so forth).

F
Fullword aligned. The displacement is a multiple of 4 (that is, position 1, 5, 9, and so forth).

D
Doubleword aligned. The displacement is a multiple of 8 (that is, position 1, 9, 17, and so forth).

Alignment can be necessary if, for example, the data is to be used in a COBOL application program where items are aligned through the SYNCHRONIZED clause. Unused space preceding aligned fields are always padded with binary zeros.

%nn
specifies that an unedited parsed input field is to appear in the reformatted output record. See PARSE for details of parsed fields. See p,m,a for further details. Note that alignment (H, F, D) is not permitted for %nn fields (for example, %nn,F results in an error message and ermination).

p
specifies that the unedited part of the variable input record (that part beyond the minimum record length), is to appear in the reformatted output record, as the last field. p without m can only be used for variable-length records; not for fixed-length records.

Attention: If 1,4,p is specified (only RDW and variable part of record), "null" records containing only an RDW will result if the record length is less than p.

A value must be specified for p that is less than or equal to the minimum record length (RECORD statement L4 value) plus 1 byte.

p,m,TRAN=keyword
Specifies that an input field is to be translated as indicated by the keyword. See p,m,TRAN=keyword under OUTFIL OUTREC for details.

%nn,TRAN=keyword
Specifies that a parsed input field is to be translated as indicated by the keyword. See %nn,TRAN=keyword under OUTFIL OUTREC for details.

p,TRAN=keyword
Specifies that the variable part of the input record is to be translated as indicated by the keyword. See p,TRAN=keyword under OUTFIL OUTREC for details.

p,m,HEX
Can be used instead of p,m,TRAN=HEX. See p,m,HEX under OUTFIL OUTREC for details.

%nn,HEX
Can be used instead of %nn,TRAN=HEX. See %nn,HEX under OUTFIL OUTREC for details.

p,HEX
Can be used instead of p,TRAN=HEX. See p,HEX under OUTFIL OUTREC for details.

[13] If INREC is specified, p must refer to the record as reformatted by INREC. If your E15 user exit reformats the record, and INREC is not specified, p must refer to the record as reformatted by your E15 user exit.

p,m,f,edit or (p,m,f),edit
specifies that an edited numeric input field is to appear in the reformatted output record. You can edit BI, FI, PD, PD0, ZD, FL, CSF, FS, UFF, SFF, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3 or TM4 fields using either pre-defined edit masks (M0-M26) or specific edit patterns you define. You can control the way the edited fields look with respect to length, leading or suppressed zeros, thousands separators, decimal points, leading and trailing positive and negative signs, and so on.

See p,m,f,edit under OUTFIL OUTREC for details.

Sample Syntax:

```
OUTREC FIELDS=(5:(21,8,ZD),M19,X,46,5,ZD,M13,
               31:35,4,FL,SIGNS=(,,+,-),LENGTH=10,
               51:8,4,PD,EDIT=(**II,IIT.TTXS),SIGNS=(,,+,-))
```

%nn,f,edit or (%nn,f),edit
specifies that an edited numeric parsed input field is to appear in the reformatted output record. See PARSE for details of parsed fields. See p,m,f,edit or (p,m,f),edit for further details.

p,m,f,to or (p,m,f),to
specifies that a converted numeric input field is to appear in the reformatted output record. You can convert BI, FI, PD, PD0, ZD, FL, CSF, FS, UFF, SFF, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3, or TM4 fields to BI, FI, PD, PDC, PDF, ZD, ZDF, ZDC, CSF, or FS fields.

See p,m,f,to under OUTFIL OUTREC for details.

Sample Syntax:

```
OUTREC FIELDS=(21,4,BI,TO=PDF,X,8,4,ZD,FI,LENGTH=2)
```

%nn,f,to or (%nn,f),to
specifies that a converted numeric parsed input field is to appear in the reformatted output record. See PARSE for details of parsed fields. See p,m,f,to or (p,m,f),to for further details.

deccon,edit or (deccon),edit
specifies that an edited decimal constant is to appear in the reformatted output record. The decimal constant must be in the form +n or -n where n is 1 to 31 decimal digits. The sign (+ or -) must be specified. A decimal constant produces a signed, 31-digit zoned decimal (ZD) result to be edited as specified.

See deccon,edit under OUTFIL OUTREC for details.

Sample Syntax:

```
OUTREC FIELDS=(-5000,EDIT=(-T,TT.T),21:(+0),M11,LENGTH=7)
```

deccon,to or (deccon),to
specifies that a converted decimal constant is to appear in the reformatted output record. The decimal constant must be in the form +n or -n where n is 1 to 31 decimal digits. The sign (+ or -) must be specified. A decimal constant produces a signed, 31-digit zoned decimal (ZD) result to be converted as specified.

See deccon,to under OUTFIL OUTREC for details.

Sample Syntax:

```
OUTREC FIELDS=(+1,PD,LENGTH=6,(-1),PD,LENGTH=6,
    -50,TO=ZD,LENGTH=8)
```

arexp,edit or (arexp),edit
specifies that the edited result of an arithmetic expression is to appear in the reformatted output record. The arithmetic expression can consist of input fields, decimal constants, operators and

parentheses. An arithmetic expression produces a signed, 31-digit zoned decimal (ZD) result to be edited as specified.

See arexp,edit under OUTFIL OUTREC for details.

Sample Syntax:

```
OUTREC FIELDS=(C'**',27,2,FI,SUB,
                    83,4,PD,EDIT=(STTTTTTT),SIGNS=(+,-),
  25:(((15,5,ZD,MUL,+2),ADD,+100),MAX,62,2,PD),M25,LENGTH=12)
```

arexp,to or (arexp),to
specifies that the converted result of an arithmetic expression is to appear in the reformatted output record. The arithmetic expression can consist of input fields, decimal constants, operators and parentheses. An arithmetic expression produces a signed, 31-digit zoned decimal (ZD) result to be converted as specified.

See arexp,to under OUTFIL OUTREC for details.

Sample Syntax:

```
OUTREC FIELDS=((15,6,FS,DIV,+5),ADD,(-1,ADD,36,6,FS),TO=FI,X,
     3,2,FI,MAX,-6,LENGTH=4,TO=FS)
```

p,m,Y2x or p,m,Y4x
Specifies that an input date field is to be edited. Real Y2x dates are edited using the century window established by the Y2PAST option in effect. Y2x and Y4x dates with special indicators are expanded appropriately (for example, p,6,Y2T transforms C'000000' to C'00000000').

See "p,m,Y2x or p,m,Y4x" under OUTFIL OUTREC for details.

Sample Syntax:

```
OUTREC BUILD=(21,3,Y2U,X,3,8,Y4W)
```

%nn,Y2x or %nn,Y4x
Specifies that a parsed input date field is to be edited. See PARSE for details of parsed fields. See "p,m,Y2x or p,m,Y4x" for further details.

p,m,Y2x,edit or p,m,Y4x,edit
Specifies that the output for a p,m,Yxx input date field is to be edited according to the edit parameters you specify. For example, if you specify:

```
OUTREC BUILD=(28,5,Y4V,EDIT=(TTTT-TT-TT)
```

The P'ccyymmdd' (X'0ccyymmddC') date value will be transformed to a C'ccyymmdd' date value and then edited to a C'ccyy-mm-dd' date value.

See "p,m,Y2x or p,m,Y4x" and "p,m,f,edit" for related details.

%nn,Y2x,edit or %nn,Y4x,edit
Specifies that the output for a parsed Yxx input date field is to be edited according to the edit parameters you specify. See PARSE for details of parsed fields. See "p,m,Y2x,edit or p,m,Y4x,edit" for further details.

p,m,Y2x,to or p,m,Y4x,to
Specifies that an input date field is to be converted according to the to parameters you specify. For example, if you specify:

```
OUTREC BUILD=(5,6,Y2W,TO=PD,LENGTH=5)
```

The C'mmddyy' date value will be transformed to a Z'mmddccyy' date value and then converted to a P'mmddccyy' (X'0mmddccyyC') date value.

See "p,m,Y2x or p,m,Y4x" and "p,m,f,to" for related details.

%nn,Y2x,to or %nn,Y4x,to
Specifies that a parsed input date field is to be converted according to the to parameters you specify. See PARSE for details of parsed fields. See "p,m,Y2x,to or p,m,Y4x,to" for further details.

p,m,Y2x,todate or p,m,Y4x,todate
Specifies that an input date field of one type is to be converted to a corresponding output date field of another type. You can convert date fields between combinations of 2-digit and 4-digit year dates, CH/ZD and PD dates, and Julian and Gregorian dates. You can also convert a date field to a corresponding day of the week in several forms.

See "p,m,Y2x,todate or p,m,Y4x,todate" under OUTFIL OUTREC for details.

Sample Syntax:

```
* Convert a P'dddyy' input date to a C'ccyy/mm/dd' output date
  OUTREC BUILD=(21,3,Y2X,TOGREG=Y4T(/),X,
* Convert a C'ccyymmdd' input date to a P'ccyyddd' output date
     42,8,Y4T,TOJUL=Y4U,X,
* Convert a C'mmddyy' input date to a C'yymmdd' output date
     11,6,Y2W,TOGREG=Y2T)
```

%nn,Y2x,todate or %nn,Y4x,todate
Specifies that an input date field of one type is to be converted to a corresponding output date field of another type. See PARSE for details of parsed fields. See "p,m,Y2x,todate or p,m,Y4x,todate" for further details.

p,m,Y2x,dateop or p,m,Y4x,dateop
Specifies an arithmetic operation for an input date field. See "p,m,Y2x,dateop or p,m,Y4x,dateop" under OUTFIL OUTREC for details.

%nn,Y2x,dateop or %nn,Y4x,dateop
Specifies an arithmetic operation for a parsed input date field. See "%nn,Y2x,dateop or %nn,Y4x,dateop" under OUTFIL OUTREC for details.

p,m,Y2x(s) or p,m,Y4x(s)
Specifies that an input date field is to be edited with separators. s can be any character except a blank. Real Y2x dates are edited using the century window established by the Y2PAST option in effect. Y2x and Y4x dates with special indicators are expanded appropriately (for example, p,8,Y4T(s) transforms C'00000000' to C'0000/00/00').

See "p,m,Y2x(s) or p,m,Y4x(s)" under OUTFIL OUTREC for details.

Sample Syntax:

```
* Convert a Z'dddccyy' date to a C'ddd/ccyy' date.
  OUTREC BUILD=(19,7,Y4W(/),X,
* Convert a P'ccyymmdd' date to a C'ccyy-mm-dd' date.
     43,5,Y4V(-))
```

%nn,Y2x(s) or %nn,Y4x(s)
Specifies that a parsed input date field is to be edited with separators. See PARSE for details of parsed fields. See "p,m,Y2x(s) or p,m,Y4x(s)" for further details.

p,m,Y2xP
Specifies that an input date field is to be converted to a packed decimal output date field. Real Y2x dates are edited using the century window established by the Y2PAST option in effect. Y2x and Y4x dates with special indicators are expanded appropriately (for example, p,6,Y2TP transforms C'000000' to P'00000000').

See "p,m,Y2xP" under OUTFIL OUTREC for details.

Sample Syntax:

```
  OUTREC BUILD=(11,3,Y2XP,X,21,4,Y2WP)
```

%nn,Y2xP
Specifies that a parsed input date field is to be converted to a packed decimal output date field. See PARSE for details of parsed fields. See "p,m,Y2xP" for further details.

p,m,lookup or %nn,lookup
specifies that a character constant, hexadecimal constant, input field (p,m) or parsed input field (%nn) from a lookup table is to appear in the reformatted output record. You can use p,m,lookup or %nn,lookup to select a specified character set constant (that is, a character or hexadecimal string) or set field (that is, an input field or parsed input field) based on matching an input value against find constants (that is, character, hexadecimal, or bit constants).

See p,m,lookup or %nn,lookup under OUTFIL OUTREC for details.

Sample Syntax:

```
OUTREC FIELDS=(11,1,
               CHANGE=(6,
                 C'R',C'READ',
                 C'U',C'UPDATE',
                 X'FF',C'EMPTY',
                 C'A',C'ALTER'),
               NOMATCH=(11,6),
               4X,
               21,1,
               CHANGE=(10,
                 B'.1......',C'VSAM',
                 B'.0......',C'NON-VSAM'))
```

p,m,justify
specifies that a left-justified or right-justified input field is to appear in the reformatted output record. For a left-justified field, leading blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For a right-justified field, trailing blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed. Optionally:

- specific leading and trailing characters can be changed to blanks before justification begins
- a leading string can be inserted
- a trailing string can be inserted
- the output length can be changed (it's equal to the input length by default)

See p,m,justify under OUTFIL OUTREC for details.

Sample Syntax:

```
  OUTREC FIELDS=(1,10,
      21,20,JFY=(SHIFT=LEFT),5X,
      52,12,JFY=(SHIFT=RIGHT,PREBLANK=C'+',LEAD=C'$',TRAIL=C'.00'))
```

%nn,justify
specifies that a left-justified or right-justified parsed input field is to appear in the reformatted output record. See PARSE for details of parsed fields. See p,m,justify for further details.

p,m,squeeze
specifies that a left-squeezed or right-squeezed input field is to appear in the reformatted output record. For a left-squeezed field, all blanks are removed and the characters from the first nonblank to the last nonblank are shifted left, with blanks inserted on the right if needed. For a right-justified field, all blanks are removed and the characters from the last nonblank to the first nonblank are shifted right, with blanks inserted on the left if needed. Optionally:

- specific characters can be changed to blanks before squeezing begins
- a leading string can be inserted
- a trailing string can be inserted
- a string (for example, a comma delimiter) can be inserted wherever a group of blanks is removed between the first nonblank and the last nonblank.

- blanks can be kept as is between paired apostrophes ('AB CD EF') or paired quotes ("AB CD EF")
- the output length can be changed (it's equal to the input length by default)

See p,m,squeeze under OUTFIL OUTREC for details.

Sample Syntax:

```
      OUTREC FIELDS=(21,20,SQZ=(SHIFT=RIGHT),5X,
        152,18,SQZ=(SHIFT=LEFT,PREBLANK=X'05',
          LEAD=C'VOL=SER=',MID=C',',PAIR=QUOTE,LENGTH=30))
```

%nn,squeeze
specifies that a left-squeezed or right-squeezed parsed input field is to appear in the reformatted output record. See PARSE for details of parsed fields. See p,m,squeeze for further details.

seqnum
specifies that a sequence number is to appear in the reformatted output record. The sequence numbers are assigned in the order in which the records are received for OUTREC processing. You can create BI, PD, ZD, CSF, or FS sequence numbers and control their lengths, starting values and increment values. You can restart the sequence number at the start value each time a specified input field (p,m) or parsed input field (%nn) changes.

See seqnum under OUTFIL OUTREC for details.

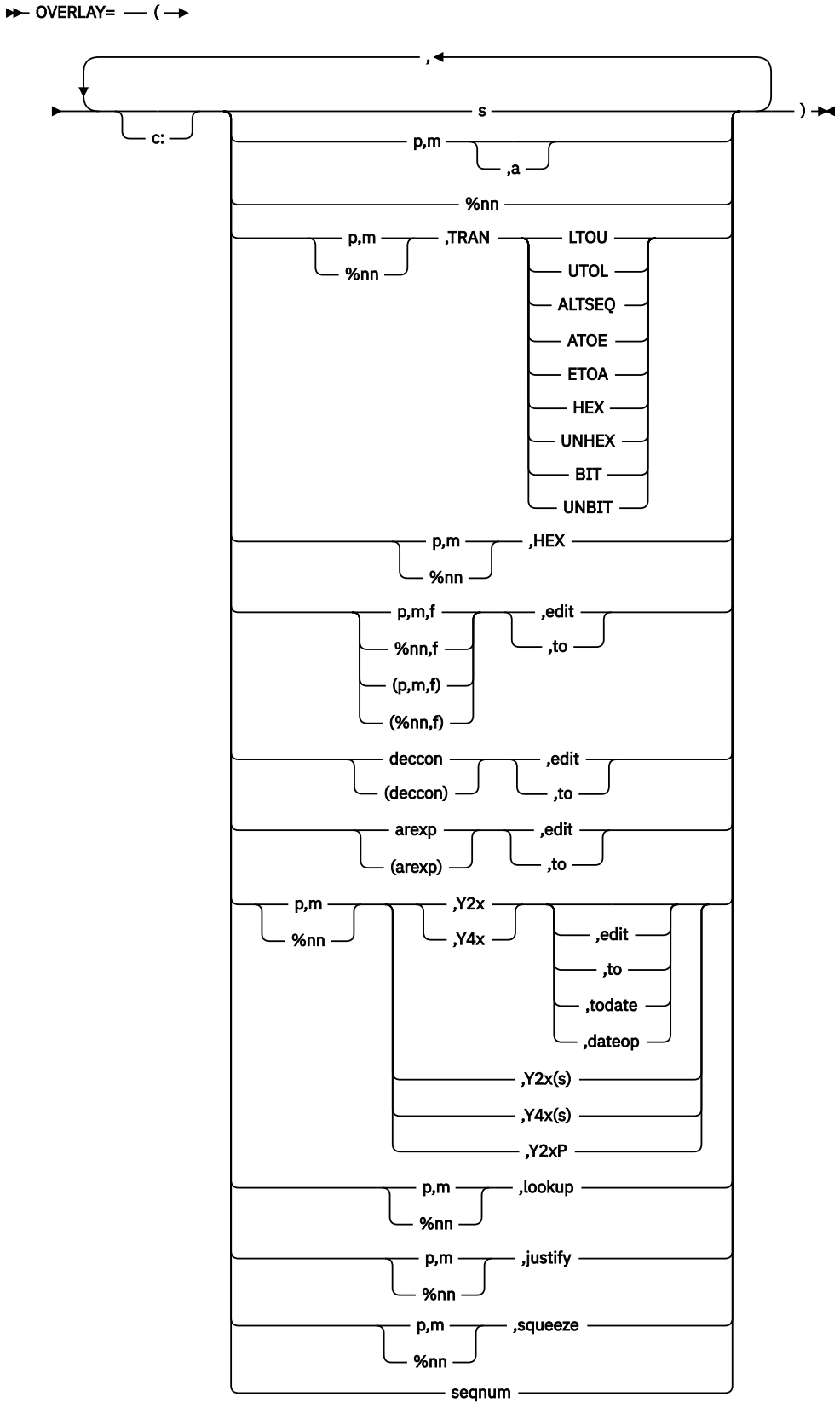
Sample Syntax:

```
  OUTREC FIELDS=(SEQNUM,6,ZD,START=1000,INCR=50,RESTART=(21,5),1,60)
```

Default for BUILD or FIELDS: None; must be specified. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

OVERLAY



Specifies each item that is to overlay specific columns in the reformatted record. Columns that are not overlaid remain unchanged. If you want to insert, rearrange, or delete fields, use BUILD or FIELDS

rather than OVERLAY. Use OVERLAY only to overlay existing columns or to add fields at the end of every record. OVERLAY can be easier to use then BUILD or FIELDS when you just want to change a few fields without rebuilding the entire record.

For fixed-length records, the first input and output data byte starts at position 1. For variable-length records, the first input and output data byte starts at position 5, after the RDW in positions 1-4.

Use c: (column) to specify the output positions to be overlaid. If you do not specify c: for the first item, it defaults to 1:. If you do not specify c: for any other item, it starts after the previous item. For example, if you specify:

```
  OUTREC OVERLAY=(25,2,11:C'A',15,3,C'**')
```

Input positions 25-26 are placed at output positions 1-2; C'A' is placed at output position 11; input positions 15-17 are placed at output positions 12-14; and C'**' is placed at output positions 15-16. The rest of the record remains unchanged.

You can specify items in any order, you can change the same item multiple times and you can overlap output columns. **Changes to earlier items affect changes to later items.** For example, say you specify:

```
OUTREC OVERLAY=(21:8,4,ZD,MUL,+10,TO=ZD,LENGTH=6,
    5:5,1,TRAN=UTOL,
    5:5,1,CHANGE=(1,C'a',C'X',C'b',C'Y'),NOMATCH=(5,1))
```

and input position 5 has 'A'. The second item (UTOL) would change 'A' to 'a' and the third item (CHANGE) would change 'a' again to 'X'.

If you specify an OVERLAY item that extends the overlay record beyond the end of the input record, the reformatted record length is automatically increased to that length, and blanks are filled in on the left as needed. For variable-length records, the RDW length is also increased to correspond to the larger reformatted record length after all of the OVERLAY items are processed. For example, if your input record has a length of 40 and you specify:

```
OUTREC OVERLAY=(16:C'ABC',51:5C'*',35:15,2)
```

the output record is given a length of 55. Blanks are filled in from columns 41-50. For variable-length records, the length in the RDW is changed from 40 to 55 after all of the OVERLAY items are processed.

Missing bytes in specified input fields are replaced with blanks so the padded fields can be processed.

The OVERLAY parameter of the OUTREC statement applies to all input records whereas the OVERLAY parameter of the OUTFIL statement only applies to the OUTFIL input records for its OUTFIL group.

See OUTREC FIELDS for details of the items listed in the OVERLAY syntax diagram shown previously in this section. You can specify all of the items for OVERLAY in the same way that you can specify them for BUILD or FIELDS with the following exceptions:

- You cannot specify p or p,HEX or p,TRAN=keyword for OVERLAY.
- For p,m,H or p,m,F or p,m,D specified for OVERLAY, fields are aligned as necessary without changing the preceding bytes.
- For variable-length records, you must not overlay positions 1-4 (the RDW) for OVERLAY, so be sure to specify the first column (c:) as 5 or greater. If you do not specify the first column, it will default to 1: which is invalid for variable-length records with OVERLAY. Whereas FIELDS=(1,m,...) is required, OVERLAY=(1,m) is not allowed since it would overlay the RDW.

Sample Syntax:

Fixed input records:

```
OUTREC OVERLAY=(21:21,4,ZD,TO=PD,LENGTH=4,
         2:5,8,HEX,45:C'*',32,4,C'*',81:SEQNUM,5,ZD)
```

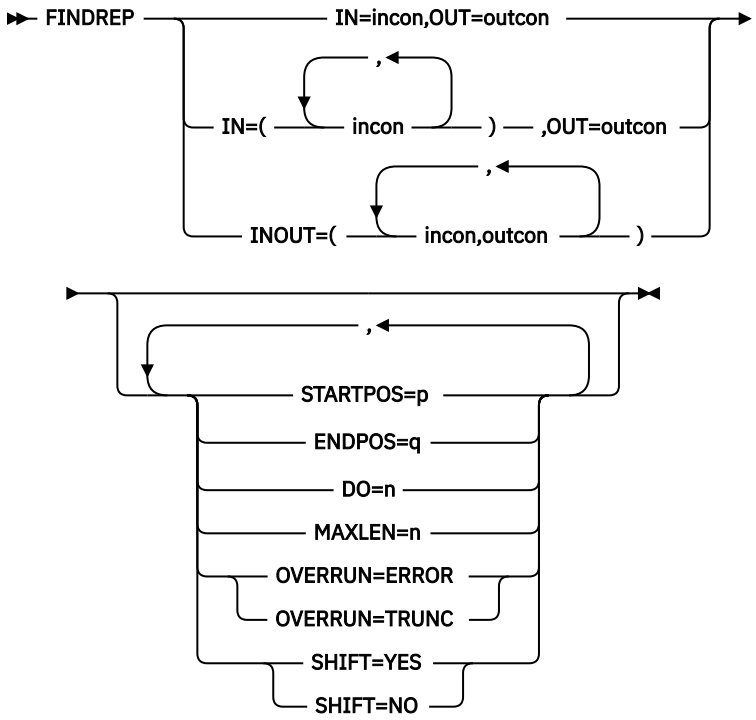
Variable input records:

```
OUTREC OVERLAY=(5:X'0001',28:C'Date is ',YDDDNS=(4D),
       17:5C'*')
```

Default for OVERLAY: None; must be specified.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

FINDREP



You can use FINDREP to find constants anywhere in a record and replace them with other constants of the same or different lengths. You can find character or hexadecimal input constants anywhere in your records and replace them with character, hexadecimal or null output constants. As appropriate, bytes can be shifted left or right, blank padding can be added for fixed-length records, and the length can be changed for variable-length records.

Various options of FINDREP allow you to define one or more input constants and a corresponding output constant, define one or more pairs of input and output constants, start and end the find scan at specified positions, stop after a specified number of constants are replaced, increase or decrease the length of the output record, define the action to be taken if nonblank characters overrun the end of the record, and specify whether output constants are to replace or overlay input constants.

See FINDREP under "OUTFIL Control Statements" for complete details.

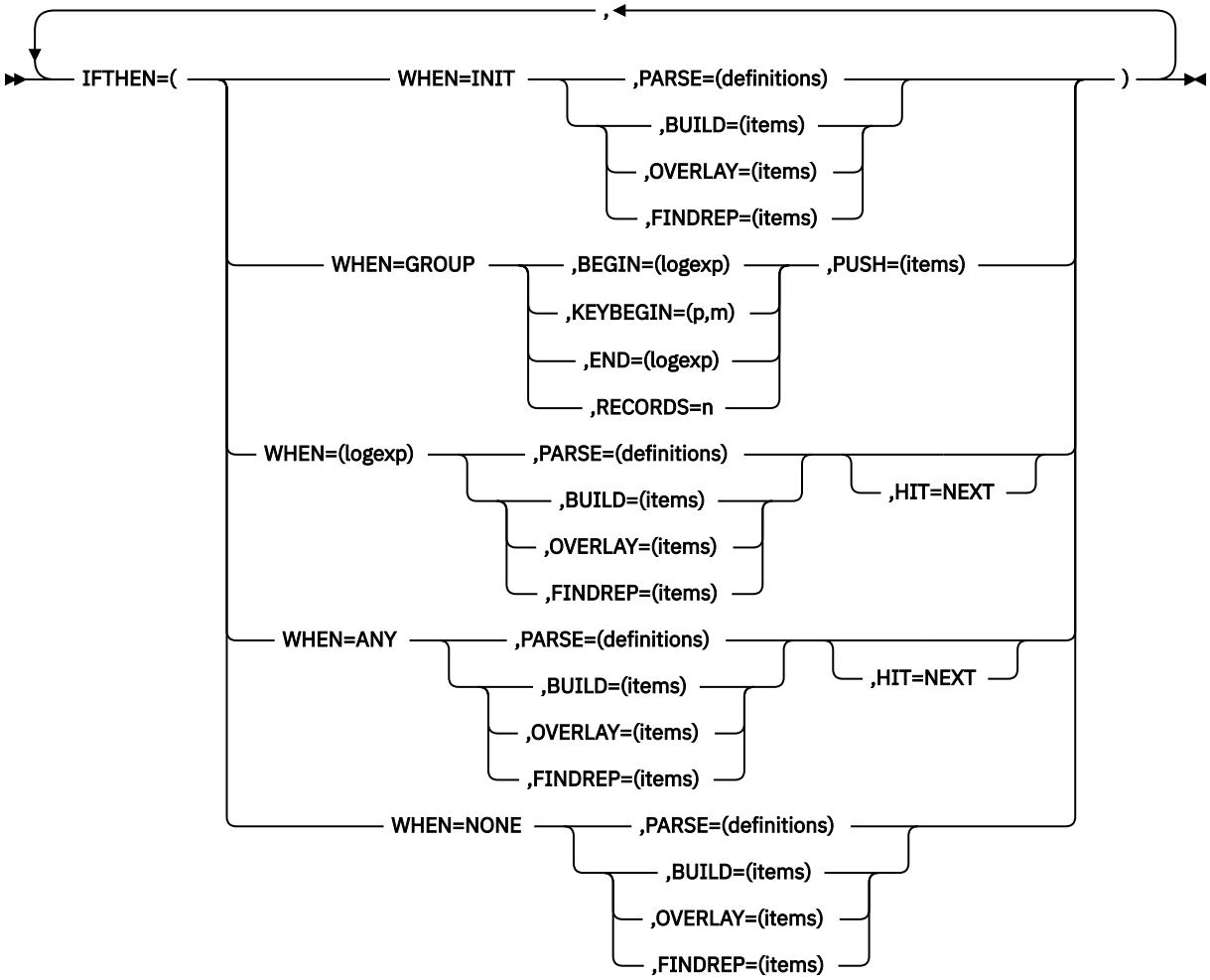
Sample Syntax:

```
OUTREC FINDREP=(INOUT=(C'Goodbye',C'Bye',C'Hello',C'Hi'))
```

Default for FINDREP: None; must be specified. See Appendix B, "Specification/Override of DFSORT Options".

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

IFTHEN



IFTHEN clauses allow you to reformat different records in different ways by specifying how build, overlay, find/replace or group operation items are to be applied to records that meet given criteria. IFTHEN clauses let you use simple or complex conditional logic to choose how different record types are reformatted.

If you want to insert, rearrange, or delete fields in the same way for every record, use BUILD or FIELDS rather than IFTHEN. If you want to do find and replace operations in the same way for every record, use FINDREP rather than IFTHEN. Use IFTHEN clauses if you want to insert, rearrange, delete or overlay fields, or perform find/replace operations in different ways for different records, or if you want to perform operations on groups of records.

You can use five types of IFTHEN clauses as follows:

- **WHEN=INIT:** Use one or more WHEN=INIT clauses to apply build, overlay or find/replace items to all of your input records. WHEN=INIT clauses and WHEN=GROUP clauses are processed before any of the other IFTHEN clauses.
- **WHEN=GROUP**: Use one or more WHEN=GROUP clauses to propagate fields, identifiers and sequence numbers within specified groups of records. WHEN=INIT clauses and WHEN=GROUP clauses are processed before any of the other IFTHEN clauses.
- **WHEN=(logexp):** Use one or more WHEN=(logexp) clauses to apply build, overlay or find/replace items to your input records that meet specified criteria. A WHEN=(logexp) clause is satisfied when the logical expression evaluates as true.

- **WHEN=ANY:** Use a WHEN=ANY clause after multiple WHEN=(logexp) clauses to apply additional build, overlay or find/replace items to your input records if they satisfied the criteria for any of the preceding WHEN=(logexp) clauses.
- **WHEN=NONE:** Use one or more WHEN=NONE clauses to apply build, overlay or find/replace items to your input records that did not meet the criteria for any of the WHEN=(logexp) clauses. WHEN=NONE clauses are processed after any of the other IFTHEN clauses. If you do not specify a WHEN=NONE clause, only the WHEN=INIT and WHEN=GROUP changes (if any) are applied to input records that do not meet the criteria for any of the WHEN=(logexp) clauses.

IFTHEN clauses are processed in the following order:

- WHEN=INIT clauses or WHEN=GROUP clauses
- WHEN=(logexp) clauses and WHEN=ANY clauses
- WHEN=NONE clauses

Processing of IFTHEN clauses continues unless one of the following occurs:

- A WHEN=(logexp) or WHEN=ANY clause is satisfied, and HIT=NEXT is not specified.
- There are no more IFTHEN clauses to process. When processing of IFTHEN clauses stops, the IFTHEN record created so far is used as the output record.

Example:

```
  OUTREC IFTHEN=(WHEN=(12,1,BI,ALL,X'3F'),OVERLAY=(18:C'Yes')),
         IFTHEN=(WHEN=(35,2,PD,EQ,+14),BUILD=(1,40,45,3,HEX),HIT=NEXT),
         IFTHEN=(WHEN=(35,2,PD,GT,+17),BUILD=(1,40,41,5,HEX),HIT=NEXT),
         IFTHEN=(WHEN=ANY,BUILD=(1,55,C'ABC',70:X)),
         IFTHEN=(WHEN=(63,2,CH,EQ,C'AB'),OVERLAY=(18:C'No')),
         IFTHEN=(WHEN=NONE,BUILD=(1,40,51,8,TRAN=LTOU))
```

For this example, the IFTHEN clauses are processed as follows:

- If IFTHEN clause 1 is satisfied, its overlay item is applied and IFTHEN processing stops.
- If IFTHEN clause 1 is not satisfied, its overlay item is not applied and IFTHEN processing continues.
- If IFTHEN clause 2 is satisfied, its build items are applied and IFTHEN processing continues.
- If IFTHEN clause 2 is not satisfied, its build items are not applied and IFTHEN processing continues.
- If IFTHEN clause 3 is satisfied, its build items are applied and IFTHEN processing continues.
- If IFTHEN clause 3 is not satisfied, its build items are not applied and IFTHEN processing continues.
- If IFTHEN clause 4 is satisfied, its build items are applied and IFTHEN processing stops.
- If IFTHEN clause 4 is not satisfied, its build items are not applied and IFTHEN processing continues.
- If IFTHEN clause 5 is satisfied, its overlay item is applied and IFTHEN processing stops.
- If IFTHEN clause 5 is not satisfied, its overlay item is not applied and IFTHEN processing continues.
- If IFTHEN clause 6 is satisfied, its build items are applied and IFTHEN processing stops.
- If IFTHEN clause 6 is not satisfied, its build items are not applied and IFTHEN processing stops.

All of the IFTHEN clauses operate sequentially on an IFTHEN record. The IFTHEN record is created initially from the input record. Each IFTHEN clause tests and changes the IFTHEN record, as appropriate. Thus, **changes made by earlier IFTHEN clauses are "seen" by later IFTHEN clauses.** For example, if you have a 40-byte input record and specify:

```
  OUTREC IFTHEN=(WHEN=INIT,OVERLAY=(8:8,4,ZD,ADD,+1,TO=ZD,LENGTH=4)),
         IFTHEN=(WHEN=(8,4,ZD,EQ,+27),OVERLAY=(28:C'Yes')),
         IFTHEN=(WHEN=NONE,OVERLAY=(28:C'No'))
```

The WHEN=INIT clause adds 1 to the ZD value and stores it in the IFTHEN record. The WHEN=(8,4,ZD,EQ,+27) clause tests the incremented ZD value in the IFTHEN record rather than the original ZD value in the input record.

The IFTHEN record is adjusted as needed for the records created or changed by the IFTHEN clauses. For fixed-length records, blanks are filled in on the left as needed. For variable-length records, the RDW length is adjusted as needed each time the IFTHEN record is changed.

Missing bytes in specified input fields are replaced with blanks so the padded fields can be processed.

DFSORT sets an appropriate LRECL (or reformatted record length if the OUTREC record is further modified) for the output records based on the build, overlay, find/replace and group operation items specified by the IFTHEN clauses. However, DFSORT does not analyze the possible results of WHEN=(logexp) conditions when determining an appropriate LRECL. When you use OUTREC IFTHEN clauses, you can override the OUTREC LRECL determined by DFSORT with the OUTREC IFOUTLEN parameter.

If SEQNUM is used in multiple IFTHEN clauses, the sequence number will be incremented for each record that satisfies the IFTHEN clause, that is, a separate SEQNUM counter will be kept for each IFTHEN clause. For example, if your input is:

```
RECORD A 1
RECORD B 1
RECORD B 2
RECORD C 1
RECORD A 2
RECORD C 2
RECORD B 3
RECORD D 1
```

and you specify:

```
  OUTREC IFTHEN=(WHEN=(8,1,CH,EQ,C'A'),OVERLAY=(15:SEQNUM,4,ZD)),
         IFTHEN=(WHEN=(8,1,CH,EQ,C'B'),OVERLAY=(16:SEQNUM,4,ZD)),
         IFTHEN=(WHEN=NONE,OVERLAY=(17:SEQNUM,4,ZD))
```

your output will be:

```
RECORD A 1    0001
RECORD B 1     0001
RECORD B 2     0002
RECORD C 1      0001
RECORD A 2    0002
RECORD C 2      0002
RECORD B 3     0003
RECORD D 1      0003
```

Separate SEQNUM counters are kept for the 'A' record, for the 'B' record, and for the NONE records.

The IFTHEN clauses of the OUTREC statement apply to all input records whereas the IFTHEN clauses of the OUTFIL statement only apply to the OUTFIL input records for its OUTFIL group.

WHEN=INIT clause

See "WHEN=INIT clause" under OUTFIL IFTHEN for details. Note that / cannot be used to create blank records or new records.

Sample Syntax:

```
  OUTREC IFTHEN=(WHEN=INIT,
           BUILD=(1,20,21:C'Department',31:3X,21,60)),
         IFTHEN=(WHEN=(5,2,CH,EQ,C'D1'),OVERLAY=(31:8,3)),
         IFTHEN=(WHEN=(5,2,CH,EQ,C'D2'),OVERLAY=(31:12,3))
```

WHEN=GROUP clause

See "WHEN=GROUP clause" under OUTFIL IFTHEN for details.

Sample Syntax

```
  OUTREC IFTHEN=(WHEN=GROUP,
    BEGIN=(1,40,SS,EQ,C'J82',OR,1,40,SS,EQ,C'M72'),
    PUSH=(41:ID=5))
```

WHEN=(logexp) clause

See "WHEN=(logexp) clause" under OUTFIL IFTHEN for details. Note that although / can be used to create blank records and new records with OUTFIL, it cannot be used with OUTREC.

Sample Syntax:

```
OUTREC IFTHEN=(WHEN=(1,3,CH,EQ,C'T01',AND,
     18,4,ZD,LE,+2000),OVERLAY=(42:C'Type1 <= 2000'),HIT=NEXT),
       IFTHEN=(WHEN=(1,3,CH,EQ,C'T01',AND,6,1,BI,BO,X'03'),
          BUILD=(1,21,42,13)),
       IFTHEN=(WHEN=(1,3,CH,EQ,C'T01',AND,
          18,4,ZD,GT,+2000),OVERLAY=(42:C'Type1 > 2000 '),HIT=NEXT),
       IFTHEN=(WHEN=(1,3,CH,EQ,C'T01',AND,6,1,BI,BO,X'01'),
          BUILD=(1,25,42,13))
```

WHEN=ANY clause

See "WHEN=ANY clause" under OUTFIL IFTHEN for details. Note that although / can be used to create blank records and new records with OUTFIL, it cannot be used with OUTREC.

Sample Syntax:

```
OUTREC IFTHEN=(WHEN=(1,3,SS,EQ,C'T01,T02,T03'),
               BUILD=(C'Group A',X,1,80),HIT=NEXT),
       IFTHEN=(WHEN=(1,3,SS,EQ,C'T04,T05,T06'),
               BUILD=(C'Group B',X,1,80),HIT=NEXT),
       IFTHEN=(WHEN=(1,3,SS,EQ,C'T07,T08,T09,T10'),
               BUILD=(C'Group C',X,1,80),HIT=NEXT),
       IFTHEN=(WHEN=ANY,OVERLAY=(16:C'Group Found')
```

WHEN=NONE clause

See "WHEN=NONE clause" under OUTFIL IFTHEN for details. Note that although / can be used to create blank records and new records with OUTFIL, it cannot be used with OUTREC.

Sample Syntax:

```
OUTREC IFTHEN=(WHEN=INIT,BUILD=(1,20,21:C'Department',31:3X,21,60)),
       IFTHEN=(WHEN=(5,2,CH,EQ,C'D1'),OVERLAY=(31:8,3)),
       IFTHEN=(WHEN=(5,2,CH,EQ,C'D2'),OVERLAY=(31:12,3)),
       IFTHEN=(WHEN=NONE,OVERLAY=(31:C'***'))
```

Default for IFTHEN clauses: None; must be specified.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

IFOUTLEN

Overrides the OUTREC LRECL (or reformatted record length if the OUTREC record is further modified) determined by DFSORT from your OUTREC IFTHEN clauses. DFSORT sets an appropriate LRECL for the output records based on the build, overlay, find/replace and group operation items specified by the IFTHEN clauses. However, DFSORT does not analyze the possible results of WHEN=(logexp) conditions when determining an appropriate OUTREC LRECL. When you use OUTREC IFTHEN clauses, you can override the OUTREC LRECL determined by DFSORT with the OUTREC IFOUTLEN parameter.

Fixed-length records longer than the IFOUTLEN length are truncated to the IFOUTLEN length. Fixed-length records shorter than the IFOUTLEN are padded with blanks to the IFOUTLEN length. Variable-length records longer than the IFOUTLEN length are truncated to the IFOUTLEN length.

n

specifies the length to use for the OUTREC LRECL (or for the reformatted record length if the OUTREC record is further modified) . The value for n must be between 1 and 32767, but must not be larger than the maximum LRECL allowed for the RECFM, and must not conflict with the specified or retrieved LRECL for the fixed-length output data set.

Sample Syntax:

```
OUTREC IFOUTLEN=70,
       IFTHEN=(WHEN=(5,1,CH,EQ,C'1',AND,8,3,ZD,EQ,+10),
```

```
            BUILD=(1,40,C'T01-GROUP-A',65)),
        IFTHEN=(WHEN=(5,1,CH,EQ,C'2',AND,8,3,ZD,EQ,+12),
            BUILD=(1,40,C'T02-GROUP-B',65))
```

Default for IFOUTLEN: The LRECL determined from the IFTHEN clauses.

OUTREC statement notes

-
- If input records are reformatted by INREC or E15, OUTREC must refer to fields in the appropriate reformatted record (see “INREC statement notes” on page 150).
- When you specify OUTREC, you must be aware of the change in record size and layout of the resulting reformatted output records.
- If the SORTOUT LRECL is specified or available, DFSORT will use it even if it does not match the reformatted OUTREC record length; this can cause padding or truncation of the reformatted OUTREC records, or termination. If the SORTOUT LRECL is not specified or available, DFSORT can automatically use the reformatted OUTREC record length as the SORTOUT LRECL, when appropriate. See the discussion of the SOLRF and NOSOLRF options in “OPTION control statement” on page 173 for details.

  For VSAM data sets, the maximum record size defined in the cluster is equivalent to the LRECL when processing fixed-length records, and is four bytes more than the LRECL when processing variable-length records. See “VSAM considerations ” on page 15 for more information.
- For variable-length records, the first entry in the FIELDS, BUILD, or IFTHEN BUILD parameter must specify or include the unedited 4-byte RDW, that is, the first field must be 1,4 or 1,m with m greater than 4. DFSORT sets the length of the reformatted record in the RDW.

  If the first field in the data portion of the input record is to appear unedited in the reformatted record immediately following the RDW, the entry in the FIELDS, BUILD, or IFTHEN BUILD parameter can specify both RDW and data field in one. (1,m,...). Otherwise, the RDW must be specifically included in the reformatted record (for example, 1,4,1,4,HEX)..
- For variable-length records, OVERLAY, IFTHEN OVERLAY or IFTHEN PUSH items must not overlay the RDW in bytes 1-4. You must ensure that 1:, 2:, 3: or 4: is not specified or defaulted for any OVERLAY or PUSH item. Note that the default for the first OVERLAY or PUSH item is 1:, so you must override it.
- With FIELDS, BUILD or IFTHEN BUILD, the variable part of the input record (that part beyond the minimum record length) can be included in the reformatted record as the last part. In this case, a value must be specified for p*n* that is less than or equal to the minimum record length (RECORD statement L4 value) plus 1 byte, and m*n* and a*n* must be omitted. For example,

  ```
  OUTREC FIELDS=(1,8,20C'*',9)
  ```

  With OVERLAY, the variable part of the input record must not be included in the reformatted record.
- If INREC with FIELDS or BUILD and OUTREC with FIELDS and BUILD are specified, either both must specify position-only for the last part, or neither must specify position-only for the last part. OVERLAY or IFTHEN, and FIELDS or BUILD, can differ with respect to position-only. See “INREC statement notes” on page 150 for more details.
- If the reformatted record includes only the RDW and the variable part of the input record, "null" records containing only an RDW may result.
- The reformatted output records are in the format specified by OUTREC regardless of whether INREC was specified.
- Fields referenced in OUTREC statements can overlap each other or control fields.
- If input is variable records, the output is also variable. This means that each record is given the correct RDW by DFSORT before output.
- When OUTREC is specified, your E35 user exit routine must refer to fields in the reformatted output record.
- DFSORT issues a message and terminates processing if an OUTREC statement is specified for a tape work data set sort or conventional merge application.

- When you specify OUTREC, VLSHRT is not used. If VLSHRT is specified, it is ignored.
- If SZERO is in effect, -0 is treated as negative and +0 is treated as positive for edited or converted input fields, decimal constants, and the results of arithmetic expressions. If NOSZERO is in effect, -0 and +0 are treated as positive for edited or converted input fields, decimal constants, and the results of arithmetic expressions.
- If SZERO is in effect, -0 compares as less than +0 when numeric fields and constants are used. If NOSZERO is in effect, -0 compares as equal to +0 when numeric fields and constants are used.

  Note: OPTION SZERO or OPTION NOSZERO is ignored for OUTREC IFTHEN=(WHEN=(logexp),...) unless the OPTION statement is found in a higher source (for example, DFSPARM is a higher source than SYSIN) or before the OUTREC statement in the same source. For example, NOSZERO will be used in both of the following cases:

  – Case 1:

```
//DFSPARM DD *
  OPTION COPY,NOSZERO
/*
//SYSIN DD *
  OUTREC IFTHEN=(WHEN=(1,2,FS,EQ,+0),OVERLAY=(22:C'Yes')),
         IFTHEN=(WHEN=NONE,OVERLAY=(22:C'No '))
/*
```

  – Case 2:

```
//SYSIN DD *
  OPTION COPY,NOSZERO
  OUTREC IFTHEN=(WHEN=(1,2,FS,EQ,+0),OVERLAY=(28:C'A')),
         IFTHEN=(WHEN=NONE,OVERLAY=(28:C'B'))
/*
```

Reformatting records after processing — examples

Both INREC and OUTREC can be used to reformat records, either separately or together. The two control statements perform similar functions, although INREC reformats records before sort, merge, or copy processing, whereas OUTREC reformats records after sort, merge, or copy processing. This section shows OUTREC examples. See “Reformatting records before processing — examples” on page 152 for INREC examples.

Example 1

```
  OUTREC FIELDS=(11,32)
```

This statement specifies that the output record is to contain 32 bytes beginning with byte 11 of the input record. This statement can be used only with fixed-length input records, because it does not include the first 4 bytes.

Example 2

```
  OUTREC FIELDS=(1,4,11,32,D,101)
```

This statement is for variable-length records of minimum length 100 bytes, and specifies that the output record is to contain an RDW plus 32 bytes of the input record starting at byte 11 (aligned on a doubleword boundary, relative to the start of the record) plus the entire variable portion of the input record.

Example 3

```
  OUTREC FIELDS=(1,42,D,101)
```

This statement is for variable-length records of minimum length 100 bytes, and specifies that the output record should contain an RDW plus the first 38 data bytes of the input record plus the entire variable portion of the input record.

The 'D' parameter has no effect because the first field is always placed at the beginning of the output record.

Example 4

```
SORT FIELDS=(20,4,CH,D,10,3,CH,D)
OUTREC FIELDS=(7:20,4,C' FUTURE ',20,2,10,3,1Z,1,9,13,7,
   24,57,TRAN=LTOU,6X'FF')
```

This example illustrates how a fixed-length input data set can be sorted and reformatted for output. The SORTIN LRECL is 80 bytes.

The reformatted output records are fixed length with a record size of 103 bytes. SOLRF (the IBM-supplied default) is in effect, so unless the SORTOUT LRECL is specified or available, it will automatically be set to the reformatted record length of 103. The reformatted records look as follows:

Position
Contents

1-6
EBCDIC blanks for column alignment

7-10
Input positions 20 through 23

11-18
Character string: C' FUTURE '

19-20
Input positions 20 through 21

21-23
Input positions 10 through 12

24
Binary zero

25-33
Input positions 1 through 9

34-40
Input positions 13 through 19

41-97
Input positions 24 through 80 with lowercase EBCDIC letters converted to uppercase EBCDIC letters

98-103
Hexadecimal string: X'FFFFFFFFFFFF'

Example 5

```
SORT FIELDS=(12,4,PD,D)
RECORD TYPE=V,LENGTH=(,,,100)
OUTREC FIELDS=(1,7,5Z,5X,28,8,6X,101)
```

This example illustrates how a variable-length input data set can be sorted and reformatted for output. The variable part of the input records is included in the output records. The minimum input record size is 100 bytes and the maximum input record size (SORTIN LRECL or maximum record size for VSAM) is 200 bytes.

The reformatted output records are variable-length with a maximum record size of 131 bytes. The reformatted records look as follows:

Position
Contents

1-4
RDW (input positions 1 through 4)

5-7
Input positions 5 through 7

8-12
Binary zeros

13-17
EBCDIC blanks

18-25
Input positions 28 through 35

26-31
EBCDIC blanks

32-n
Input positions 101 through n (variable part of input records)

Example 6

```
MERGE FIELDS=(28,4,BI,A)
OUTREC BUILD=(1,4,5Z,5X,5,3,28,8,6Z,DATE3-1)
```

This example illustrates how input files can be merged and reformatted for output, with yesterday's date included. The variable part of the input records is not to be included in the output records. The SORTINnn LRECL is 50 bytes.

The reformatted output records are variable-length with a maximum record size of 38 bytes. The reformatted records look as follows:

Position
Contents

1-4
RDW (input positions 1 through 4)

5-9
Binary zeros

10-14
EBCDIC blanks

15-17
Input positions 5 through 7

18-25
Input positions 28 through 35

26-31
Binary zeros

32-38
Yesterday's date in the form C'yyyyddd'

Example 7

```
OPTION COPY,Y2PAST=1985
  OUTREC FIELDS=(SEQNUM,8,ZD,START=1000,INCR=100,
                 11:8,4,PD,M12,
                 31:15,4,Y2V(/),
                 51:2,1,CHANGE=(3,
                          X'01',C'L92',X'02',C'M72',X'03',C'J42'),
                        NOMATCH=(C'???'))
```

This example illustrates how a sequence number can be generated, how numeric and date values can be edited, and how a lookup table can be used.

The reformatted output records look as follows:

Position
Contents

1-8
A zoned decimal sequence number that starts at 1000 and increments by 100.

11–20
A CH field containing the PD field from input positions 8 through 11 edited according to the M12 edit mask.

31-40
A C'yyyy/mm/dd' date field containing the P'yymmdd' date field from input positions 15-18 transformed according to the specified century window of 1985-2084.

51–53
A CH field containing C'L92', C'M72', C'J42' or C'???' as determined by using a lookup table for the input field in position 2.

Example 8

```
SORT FIELDS=(11,4,CH,D)
OUTREC FIELDS=(1,20,
  (5,4,FI,ADD,3,2,FI,ADD,23,2,FI),DIV,+1000,
     EDIT=(STTTTTTT),SIGNS=(,-),2X,
  9,5,ZD,MIN,16,5,FS,TO=ZD,LENGTH=5,2X,
  21,40)
```

This example illustrates how input records can be reformatted for output to contain the results of arithmetic expressions involving input fields, decimal constants, operators and parentheses.

The reformatted output records look as follows:

Position
Contents

1-20
Input positions 1 through 20

21-28
A CH field containing the total of the FI fields from positions 5 through 8, 3 through 4 and 23 through 24, divided by 1000, and edited according to the specified edit pattern.

29-30
EBCDIC blanks

31-35
A ZD field containing the minimum of the ZD field in positions 9 through 13 and the FS field in positions 16 through 20.

36-37
EBCDIC blanks

38-77
Input positions 21-60

Example 9

```
OPTION COPY
OUTREC IFTHEN=(WHEN=(3,2,SS,EQ,C'FR,MX,GR'),
                OVERLAY=(11:DATE=(DM4.),TIME1(.))),
       IFTHEN=(WHEN=(3,2,SS,EQ,C'CN,US,EN'),
                OVERLAY=(11:DATE=(4MD/),TIME1(:)))
```

This example illustrates how you can use IFTHEN clauses with OUTREC to reformat different records in different ways.

For records with 'FR', 'GR' or 'MX' in positions 3-4, positions 11-20 of the reformatted output records are overlaid with a date of the form 'dd.mm.yyyy' and positions 21-28 are overlaid with a time of the form 'hh.mm.ss'. The data before positions 11-28 and after positions 11-28 are not affected.

For records with 'CN', 'US' or 'EN' in positions 3-4, positions 11-20 of the reformatted output records are overlaid with a date of the form 'yyyy/mm/dd' and positions 21-28 are overlaid with a time of the form 'hh:mm:ss'. The data before positions 11-28 and after positions 11-28 are not affected.

Since an IFTHEN clause with WHEN=NONE is not specified, records without 'FR', 'GR', 'MX', 'CN', 'US' or 'EN' in positions 3-4 are not changed.

Example 10

```
OPTION COPY
OUTREC OVERLAY=(31:11,10,ZD,DIV,+1200,TO=PD,LENGTH=6,
                37:11,10,ZD,MOD,+1200,TO=PD,LENGTH=4)
```

This example illustrates how you can use the OVERLAY parameter with OUTREC to change certain columns in your records without affecting other columns.

Positions 31-36 of the reformatted input records are overlaid with a 6-byte PD value equal to the quotient of the 10-byte ZD value at positions 11-20 divided by +1200. Positions 37-40 of the reformatted input records are overlaid with a 4-byte PD value equal to the remainder of the 10-byte ZD value at positions 11-20 divided by +1200. The data before positions 31-40 and after positions 31-40 are not affected.

Example 11

```
OPTION COPY
OUTREC IFTHEN=(WHEN=INIT,
          OVERLAY=(81:11,10,
            11:81,10,ZD,DIV,+1200,TO=PD,LENGTH=6,
            17:81,10,ZD,MOD,+1200,TO=PD,LENGTH=4)),
       IFOUTLEN=80
```

In the previous example (Example 10), we used OVERLAY to overlay positions 31-40 with PD fields for the quotient and remainder of the 10-byte ZD value at positions 11-20 divided by +1200. In this example, we want to overlay positions 11-20 with the quotient and remainder. The input records are 80 bytes and fixed-length and we want the output records to be 80 bytes and fixed-length as well.

If we just overlaid the fields directly as before, we would "ruin" the ZD value before we could use it to get the remainder; the PD quotient would overlay positions 11-16, so positions 11-20 would no longer contain the ZD value we need to get the remainder.

In order to overlay the ZD value itself with the PD values, we make a copy of the 10 byte ZD value after the end of the record, and then overlay the original ZD value with the quotient and remainder derived from the copy of the ZD value.

By making a copy of the 10 byte ZD value at the end of the record, we extend the record length from 80 bytes to 90 bytes. Since we want the final record length to be 80 bytes, we must remove the extra 10 bytes. So instead of just using the OVERLAY parameter, we use an IFTHEN WHEN=INIT clause with OVERLAY, and IFOUTLEN=80. Alternatively, we could use an OVERLAY parameter on the OUTREC statement, followed by an OUTFIL statement, to remove the extra 10 bytes like this:

```
OPTION COPY
OUTREC OVERLAY=(81:11,10,
             11:81,10,ZD,DIV,+1200,TO=PD,LENGTH=6,
             17:81,10,ZD,MOD,+1200,TO=PD,LENGTH=4)
OUTFIL OUTREC=(1,80)
```

Example 12

```
OPTION COPY
OUTREC OVERLAY=(11:11,16,JFY=(SHIFT=RIGHT,LEAD=C'(',
  TRAIL=C')',LENGTH=18))
```

This example illustrates how you can right-justify fields within your records.

The 50-byte FB input records might look like this:

```
0001      9-1-632-731
0002      011-276-321-7836
0003      753-218-307
0004      528-314
```

Note that the second field has left-justified numeric values in various forms. We want to right-justify these values and surround them with a left parenthesis and right parenthesis.

The 50-byte FB output records look like this:

```
0001           (9-1-632-731)
0002      (011-276-321-7836)
0003           (753-218-307)
0004               (528-314)
```

We use OUTFIL OVERLAY to limit the changes to the second field. We use JFY to right-justify the second field with surrounding parentheses. SHIFT=RIGHT shifts the characters to the right. LEAD=C'(' adds a left parenthesis before the first non-blank character. TRAIL=C')' adds a right parenthesis after the last non-blank character. LENGTH=18 increases the output length by 2 bytes to allow for the parentheses (overriding the default of 16 from the input length).

Example 13

```
  OPTION COPY
  OUTREC PARSE=(%01=(ABSPOS=2,FIXLEN=13,ENDBEFR=C'","'),
                %=(ENDBEFR=C'","'),
                %03=(FIXLEN=6,ENDBEFR=C'","'),
                %04=(FIXLEN=6,ENDBEFR=C'","'),
                %05=(FIXLEN=12,ENDBEFR=C'"')),
    BUILD=(%01,20:%03,SFF,ADD,%04,SFF,EDIT=(SIIT.T),SIGNS=(,-),
           31:%05)
```

This example illustrates how you can reformat records containing variable position/length fields, such as comma separated values.

The 80-byte input records might look like this:

```
"Buffy Summers","F","+725.8","-27.3","Sunnydale"
"Bruce Wayne","M","-5.3","-173.2","Gotham City"
"Clark Kent","M","+21.3","-15.8","Metropolis"
"Diana Prince","F","-16.4","+128.9","Gateway City"
```

Note that each record has five variable fields, each enclosed in quotes and separated by a comma. The fields do not start and end in the same position in every record and they have different lengths in different records.

The 42-byte output records should look like this:

```
Buffy Summers       698.5     Sunnydale
Bruce Wayne        -178.5     Gotham City
Clark Kent            5.5     Metropolis
Diana Prince        112.5     Gateway City
```

The first fixed-length output field corresponds to the first variable input field. The second fixed-length output field corresponds to the total of the third and fourth variable input fields. The third fixed-length output field corresponds to the fifth variable input field.

In order to reformat the input records for output, we use PARSE and BUILD to create the fixed parsed fields we need from the variable position/length fields; the quotes around each value are removed. %00 is used for the first input field. % is used to ignore the second input field. %03 and %04 are used for the third and fourth input fields which are added together using SFF format. %05 is used for the fifth input field.

Example 14

```
OPTION COPY
OUTREC IFTHEN=(WHEN=INIT,
         PARSE=(%100=(FIXLEN=3,ENDBEFR=C'.',REPEAT=3),
                %103=(FIXLEN=3)),
         BUILD=(%100,4:C'.',5:%101,8:C'.',9:%102,12:C'.',13:%103)),
  IFTHEN=(WHEN=(1,3,CH,EQ,C'167',AND,13,3,CH,EQ,C'99'),
          OVERLAY=(5:C'113',9:C'75 ',1:1,15,SQZ=(SHIFT=LEFT))),
  IFTHEN=(WHEN=NONE,
          BUILD=(1:1,15,SQZ=(SHIFT=LEFT)))
```

This example illustrates how you can modify variable position/length fields, such as ftp addresses.

The 15-byte input records might look like this:

```
167.113.117.99
167.90.18.99
167.80.118.98
165.250.89.562
167.125.890.95
168.250.89.99
167.125.890.99
0.0.0.0
167.90.580.99
```

We want to convert each FTP address of the form 167.x.y.99 to 167.113.75.99. x and y can be any 1-3 digit value. Thus, the 15-byte output records should look like this:

```
167.113.75.99
167.113.75.99
167.80.118.98
165.250.89.562
167.125.890.95
168.250.89.99
167.113.75.99
0.0.0.0
167.113.75.99
```

In order to reformat the input records for output, we use IFTHEN clauses as follows:

- IFTHEN WHEN=INIT clause: We PARSE the four variable numeric values into four 3-byte fixed parsed fields using %100, %101, %102 and %103 respectively. Note that we use REPEAT=3 with the %100 parsed field to repeat it for the %101 and %102 parsed fields. REPEAT=n is useful when you have several contiguous parsed fields with the same operands).

  We reformat the record with %100, a period, %101, a period, %102, a period and %103. At this point, the reformatted records look like this:

  ```
  167.113.117.99
  167.90 .18 .99
  167.80 .118.98
  165.250.89 .562
  167.125.890.95
  168.250.89 .99
  167.125.890.99
  0  .0  .0  .0
  167.90 .580.99
  ```

- IFTHEN WHEN=(logexp) clause: If the first fixed-length field is '167' and the fourth fixed-length field is '99 ', we overlay the second fixed-length field with '113' and the third fixed-length field with '75'. Then we squeeze the fields to the left to remove the blanks.
- IFTHEN WHEN=NONE clause: If the first fixed-length field is not '167' or the fourth fixed-length field is not '99 ', we squeeze the fields to the left to remove the blanks.

Example 15

```
  SORT FIELDS=(1,2,CH,A)
  OUTREC FINDREP=(IN=C'*',OUT=C' ')
```

This example illustrates how you can replace a character with another character anywhere in FB or VB records.

The FB input records might look like this:

```
05 ***** JUNE ***************
02         * APRIL *
01* * * * DAISY * * * *
03BETTY ***********
```

We want to replace every asterisk with a blank. We use IN=C'*' to indicate we want to find each asterisk, and OUT=C' ' to indicate we want to replace it with a blank.

The sorted output records look like this:

```
01        DAISY
02           APRIL
03BETTY
05       JUNE
```

Example 16

```
  OPTION COPY
  OUTREC IFTHEN=(WHEN=GROUP,BEGIN=(1,3,CH,EQ,C'HDR'),
    END=(1,3,CH,EQ,C'TRL'),PUSH=(31:ID=1))
  OUTFIL INCLUDE=(31,1,CH,NE,C' '),BUILD=(1,30)
```

This example illustrates how you can INCLUDE groups of FB records between a header and a trailer. We add an ID after the end of each record to indicate whether it's part of a group or not, INCLUDE on the ID, and then remove it.

The 30-byte FB input records might look like this:

```
C33  Not in a group
HDR   Start Group 1
A01    Group 1 record
B02    Group 1 record
C03    Group 1 record
TRL   End Group 1
R24  Not in a group
T02  Not in a group
HDR   Start Group 2
D04    Group 2 record
E05    Group 2 record
TRL   End Group 2
F97  Not in a group
```

In the output data set we only want to include groups of records that start with 'HDR' and end with 'TRL'.

We use an IFTHEN WHEN=GROUP clause to put a non-blank character in each record that is part of a group. BEGIN indicates a group starts with a record that has 'HDR' in positions 1-3. END indicates a group ends with a record that has 'TRL' in positions 1-3. PUSH overlays a 1-byte ID character at position 31 in each record of a group (after the end of the record). After the IFTHEN GROUP clause is executed, the intermediate records look like this:

```
C33  Not in a group
HDR   Start Group 1           1
A01    Group 1 record         1
B02    Group 1 record         1
C03    Group 1 record         1
TRL   End Group 1             1
R24  Not in a group
T02  Not in a group
HDR   Start Group 2           2
D04    Group 2 record         2
E05    Group 2 record         2
TRL   End Group 2             2
F97  Not in a group
```

Note that the records within a group have a non-blank character in position 31 whereas the records outside groups have a blank character in position 31. The ID starts at 1 for the first group and is incremented by 1 for each subsequent group. Since we are only allowing one character for the ID, when the ID counter gets to 10, a '0' will appear in position 31. That's fine since we are just looking for a non-blank to indicate a record within a group, or a blank to indicate a record outside of a group.

We use an OUTFIL statement to only INCLUDE records with a non-blank in position 31, and to remove the ID character so the included output records will be identical to the input records. After the OUTFIL statement is executed, the final output records look like this:

```
HDR   Start Group 1
A01    Group 1 record
B02    Group 1 record
C03    Group 1 record
TRL   End Group 1
HDR   Start Group 2
D04    Group 2 record
E05    Group 2 record
TRL   End Group 2
```

Example 17

```
  OPTION COPY
  OUTREC IFTHEN=(WHEN=INIT,BUILD=(1,4,6:5)),
    IFTHEN=(WHEN=GROUP,BEGIN=(6,3,CH,EQ,C'HDR'),
      END=(6,3,CH,EQ,C'TRL'),PUSH=(5:ID=1))
  OUTFIL INCLUDE=(5,1,CH,NE,C' '),BUILD=(1,4,5:6)
```

This example illustrates how you can INCLUDE groups of VB records between a header and a trailer. It's similar to Example 16, but here the records are variable-length. For the FB records, we could add the ID after the end of each record and then remove it without changing the records. But we can't add the ID at the end of each VB record because that would pad all of the records to a fixed length. So, instead we insert the ID between the RDW and the first data byte of each record, and later remove it.

The VB input records might look like this:

```
Len|Data
 23|C33  Not in a group
 23|HDR   Start Group 1
 25|A01    Group 1 record
 25|B02    Group 1 record
 25|C03    Group 1 record
 21|TRL   End Group 1
 23|R24  Not in a group
 23|T02  Not in a group
 23|HDR   Start Group 2
 25|D04    Group 2 record
 25|E05    Group 2 record
 21|TRL   End Group 2
 25|F97  Not in a group
```

In the output data set we only want to include groups of records that start with 'HDR' and end with 'TRL'.

We use an IFTHEN WHEN=INIT clause to reformat each record so it has room for the ID byte between the RDW and the first data byte. After the WHEN=INIT clause is executed, the intermediate records look like this:

```
Len|Data
 24| C33  Not in a group
 24| HDR   Start Group 1
 26| A01    Group 1 record
 26| B02    Group 1 record
 26| C03    Group 1 record
 22| TRL   End Group 1
 24| R24  Not in a group
 24| T02  Not in a group
 24| HDR   Start Group 2
 26| D04    Group 2 record
 26| E05    Group 2 record
```

```
 22| TRL   End Group 2
 26| F97  Not in a group
```

Note that position 5 is blank and the 'HDR' and 'TRL' characters have been shifted over to positions 6-8.

We use an IFTHEN WHEN=GROUP clause to put a non-blank character in each record that is part of a group. BEGIN indicates a group starts with a record that has 'HDR' in positions 6-8. END indicates a group ends with a record that has 'TRL' in positions 6-8. PUSH overlays a 1-byte ID character at position 5 in each record of a group. After the IFTHEN GROUP clause is executed, the intermediate records look like this:

```
Len|Data
 24| C33  Not in a group
 24|1HDR   Start Group 1
 26|1A01    Group 1 record
 26|1B02    Group 1 record
 26|1C03    Group 1 record
 22|1TRL   End Group 1
 24| R24  Not in a group
 24| T02  Not in a group
 24|2HDR   Start Group 2
 26|2D04    Group 2 record
 26|2E05    Group 2 record
 22|2TRL   End Group 2
 26| F97  Not in a group
```

Note that the records within a group have a non-blank character in position 5 whereas the records outside groups have a blank character in position 5. The ID starts at 1 for the first group and is incremented by 1 for each subsequent group. Since we are only allowing one character for the ID, when the ID counter gets to 10, a '0' will appear in position 5. That's fine since we are just looking for a non-blank to indicate a record within a group, or a blank to indicate a record outside of a group.

We use an OUTFIL statement to only INCLUDE records with a non-blank in position 5, and to remove the ID character so the included output records will be identical to the input records. After the OUTFIL statement is executed, the final output records look like this:

```
Len|Data
 23|HDR   Start Group 1
 25|A01    Group 1 record
 25|B02    Group 1 record
 25|C03    Group 1 record
 21|TRL   End Group 1
 23|HDR   Start Group 2
 25|D04    Group 2 record
 25|E05    Group 2 record
 21|TRL   End Group 2
```

Example 18

```
  SORT FIELDS=(1,12,CH,A)
  OUTREC OVERLAY=(30:16,8,Y4T,TOGREG=Y4T)
  OUTFIL INCLUDE=(30,1,CH,EQ,C'*')
```

This example illustrates how to list records with dates outside of the valid range (for example, a month not between 01-12).

Note: Dates with an invalid digit (A-F) can result in a data exception (0C7 ABEND).

The input records might be as follows:

```
Betten         20091021
Vezinaw        20091101
Casad          00000000
Boenig         20091325
Kolusu         20090931
Yaeger         20090731
```

The SORT statement sorts the records by the name in positions 1-12. After the SORT statement is processed, the sorted records will look like this:

```
Betten         20091021
Boenig         20091325
Casad          00000000
Kolusu         20090931
Vezinaw        20091101
Yaeger         20090731
```

The OUTREC statement uses TOGREG to convert each ccyymmdd value in positions 16-23 to a ccyymmdd value in positions 30-37; the third column will be identical to the second column for valid dates and special indicators, but will contain asterisks for invalid dates. After the OUTREC statement is processed, the reformatted records will look like this:

```
Betten         20091021      20091021
Boenig         20091325      ********
Casad          00000000      00000000
Kolusu         20090931      ********
Vezinaw        20091101      20091101
Yaeger         20090731      20090731
```

The OUTFIL statement selects the records that have an asterisk in position 30, that is, the records with an invalid date. The Boenig record is invalid because mm is 13, and the Kolusu record is invalid because mm is 09 but dd is 31 (September only has 30 days). Note that the special indicator of all 0s for the Casad record is valid. The output records would be as follows:

```
Boenig         20091325      ********
Kolusu         20090931      ********
```

Example 19

```
  OPTION COPY,Y2PAST=1990
  OUTREC IFTHEN=(WHEN=INIT,
    BUILD=(1:1,8,UFF,TO=ZD,LENGTH=6,8:11,6,UFF,TO=ZD,LENGTH=5)),
   IFTHEN=(WHEN=INIT,
     BUILD=(1,6,Y2W,DATEDIFF,8,5,Y2T))
```

This example illustrates how you can calculate the difference in days between two different types of date fields, each of which has separators.

The SORTIN data set has these input records with a C'mm/dd/yy' date field in positions 1-8 and a C'yy/ddd' date field in positions 11-16:

```
03/05/07 07/052
12/13/07 08/193
02/19/08 08/365
09/01/08 09/001
11/22/09 09/015
07/22/98 01/121
01/15/09 09/015
09/15/10 09/322
06/30/10 08/050
```

The OUTREC statement calculates the number of days for date1-date2 and puts the result in the output record in positions 1-8.

The first IFTHEN clause removes the / separators from date1 and date2 so we can use them in DATEDIFF. After the first IFTHEN clause, the records look like this, with the C'mmddyy' date in positions 1-6 and the C'yyddd' date in positions 8-12:

```
030507 07052
121307 08193
021908 08365
090108 09001
112209 09015
072298 01121
011509 09015
091510 09322
063010 08050
```

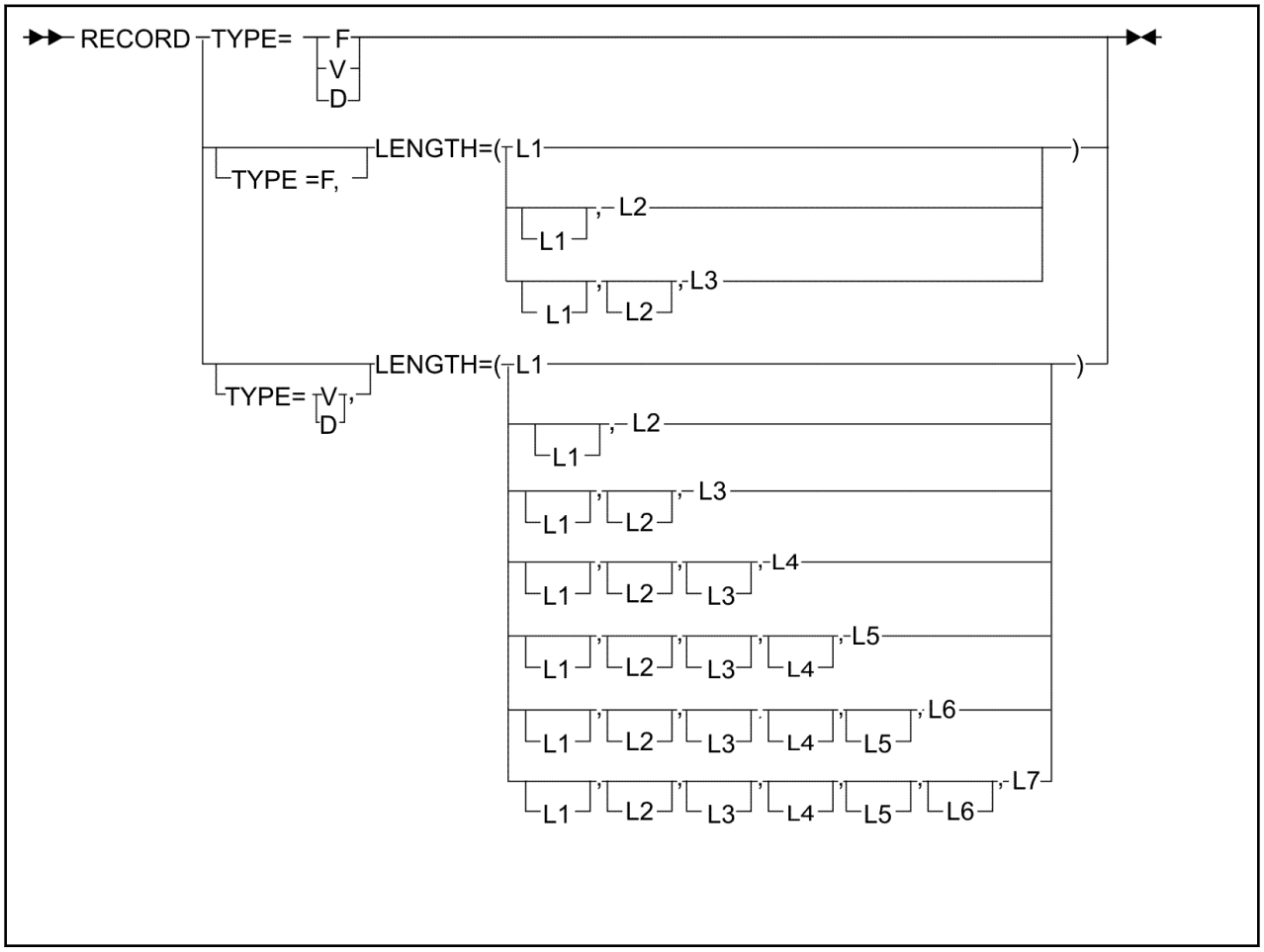
The second IFTHEN clause uses DATEDIFF to get the number of days between date1 and date2. We use 1,6,Y2W to match the C'mmddyy' date and 8,5,Y2T to match the C'yyddd' date.

SORTOUT will have these records:

```
+0000012
-0000211
-0000315
-0000122
+0000311
-0001014
+0000000
+0000301
+0000862
```

Note that when date1>=date2, the result is a positive value, and when date1<date2, the result is a negative value.

RECORD control statement

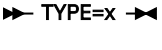


The RECORD control statement can be used to specify the type and lengths of the records being processed, and the minimum and average record lengths for a variable-length sort.

The RECORD control statement is required when:

- A user exit changes record lengths.
- A user exit supplies all of the input records.
- A Conventional merge or tape work data set sort uses VSAM input.

TYPE



Can be used to specify the record type when input is VSAM, or an E15 or E32 exit supplies all of the input records. The record type can be:

- **Fixed-length (F).** The records are processed without an RDW, so the data starts in position 1. Control statement positions should be specified accordingly.

  An RRDS can always be processed as fixed-length. A KSDS, ESDS or VRRDS used for input should only be processed as fixed-length if all of its records have a length equal to the maximum record size defined for the cluster. Otherwise, input records which are shorter than the maximum record size are padded with bytes that may or may not be zeros (that is, "garbage" bytes).
- **Variable-length (V).** The records are processed with an RDW in positions 1-4, so the data starts in position 5. Control statement positions should be specified accordingly.

  An RRDS, KSDS, ESDS or VRRDS can always be processed as variable-length. For VSAM input, DFSORT reads each record and prepends an RDW to it. For VSAM output, DFSORT removes the RDW before writing each record.

TYPE is only required for a Conventional merge or tape work data set sort that uses VSAM input or an E15 or E32 exit that supplies all of the input records.

If input is non-VSAM, DFSORT determines the record type from the RECFM of the input data set and ignores TYPE.

If input is VSAM, or an E15 or E32 exit supplies all of the input records, DFSORT determines or assigns the record type as follows, using the information in the order listed:

1. F or V from RECORD TYPE if specified.
2. F or V from SORTOUT RECFM if available.
3. V if OUTFIL VTOF, CONVERT or VLFILL is specified, or F if OUTFIL FTOV is specified.
4. F or V from OUTFIL RECFM if available.
5. V if SORTIN is VSAM and SORTOUT is VSAM; otherwise F.

   Note:

   a. If the selected record type is not what you want DFSORT to use, specify RECORD TYPE=F or RECORD TYPE=V as appropriate.

   b. For a Conventional merge or tape work data set sort, you must specify RECORD TYPE=F or RECORD TYPE=V as appropriate.

x can be one of the following:

F
fixed-length record processing.

Note: FB can be used instead of F.

V
variable-length record processing.

Note: VB can be used instead of V.

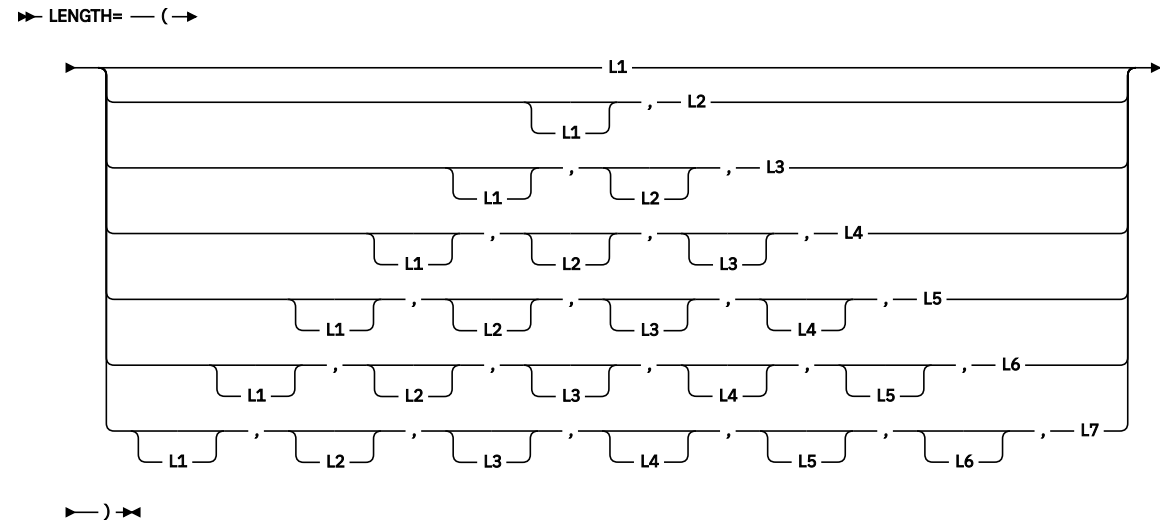
D
ASCII variable-length record processing.

Note: DB can be used instead of D.

Default: F or V as described previously in this section. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

LENGTH



Can be used to specify various record lengths. L1 through L3 apply to fixed-length and variable-length record processing. L4 and L5 apply to variable-length record processing. L6 and L7 are accepted, but not used.

LENGTH is required only if:

- A user exit changes record lengths.
- A user exit supplies all of the input records.

L1

Input record length. For variable-length records, maximum input record length.

Note:

1. L1 is ignored if the input record length is available from SORTIN.
2. L1 is required if there is no SORTIN or SORTINnn data set, unless L2 is specified.

Default: The SORTIN or SORTINnn record length. For VSAM data sets, the maximum record size (RECSZ value).

L2

Record length after E15. For variable-length records, maximum record length after E15.

Note:

1. L2 is ignored if E15 is not used.
2. An accurate value for L2 must be specified if E15 changes the record length.
3. L2 must be at least 18 bytes if tape work data sets are used.
4. L2 is ignored if there is no SORTIN or SORTINnn data set, unless L1 is not specified.

Default: L1.

L3

Output record length. For variable—length records, maximum output record length.

Note: L3 is ignored if the record length (LRECL or VSAM RECSZ) is available from SORTOUT, or if NOSOLRF is in effect and E35, INREC, OUTREC, and OUTFIL are *not* used.

Default: One of the following, in the order listed:

1. SORTOUT record length if available
2. OUTREC record length if SOLRF is in effect
3. INREC record length if SOLRF is in effect

4. L2 if specified providing an E15 is used
5. SORTIN or SORTINnn record length if available
6. L1

L4
Minimum record length.

Note:

1. L4 is not used if the Blockset technique is selected
2. L4 is only used for variable-length record sort applications.
3. Specifying L4 may improve performance, but if L4 is too large, DFSORT could fail with message ICE015A.

Default: The minimum length needed to contain all control fields. This number must be at least 18 bytes if the maximum input record length is greater than 18 bytes; otherwise, DFSORT sets L4 to 18 bytes.

L5
Average record length.

Note:

1. L5 is not used if the Blockset technique is selected
2. L5 is overridden by the AVGRLEN parameter if both are specified
3. L5 is only used for variable-length sorts.

Default: None; optional.

L6, L7
Record lengths that are accepted but are reserved for future use.

Note:

1.
2. You can drop values from the right. For example, LENGTH=(80,70,70,70).
3. You can omit values from the middle or left, provided you indicate their omission by a comma or semicolon. For example, LENGTH=(,,,30,80).
4. Parentheses are optional when L1 alone is specified. If any of L2 through L7 is specified, with or without L1, parentheses are required.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

Describing the record format and length—examples

Example 1

```
MODS E15=(INEX,1000,EXIT),E35=(OUTEX,2000,EXIT)
 RECORD LENGTH=(,175,180)
```

This example illustrates how the RECORD statement can be used to indicate that E15 and E35 exits change the record length. The record type (F) and input record length (200) are obtained automatically from the RECFM and LRECL of the input data set, respectively.

LENGTH
L2 specifies that the E15 exit passes back 175 byte records. L3 specifies that the E35 exit passes back 180 byte records.

Example 2

```
MODS E15=(E15ONLY,1000,EXIT)
RECORD TYPE=V,LENGTH=60
```

This example illustrates how the RECORD statement can be used to set the record type and maximum input record length when an E15 exit supplies all of the input as variable-length records.

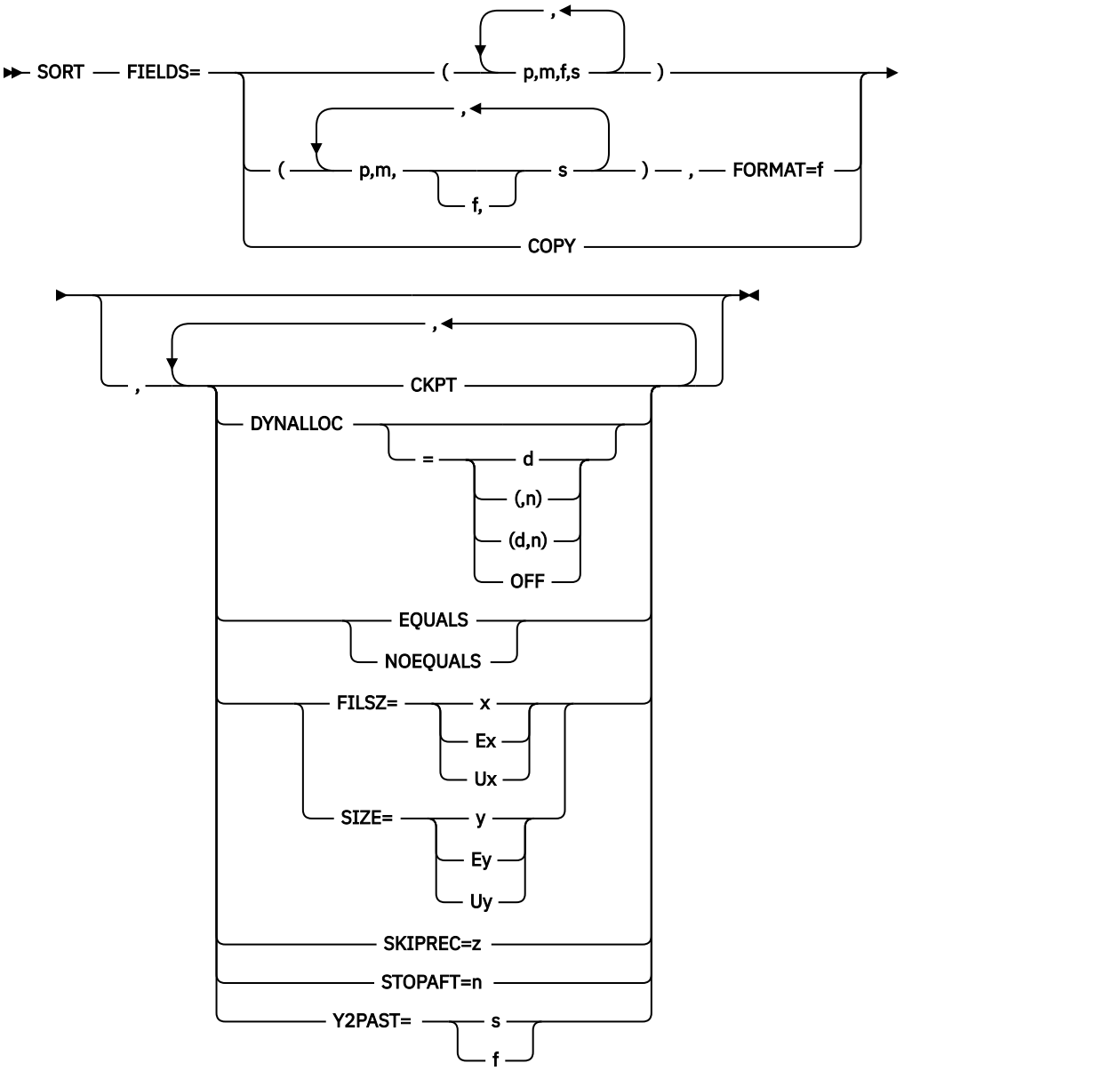
TYPE
V specifies that the E15 exit inserts variable-length records, that is, the inserted records contain an RDW in positions 1-4 and the data starts in position 5.

LENGTH
L1 specifies that the E15 exit inserts records with a maximum length of 60 bytes.

REFORMAT control statement

Used only for a JOINKEYS application. See Chapter 4, “Using a JOINKEYS application for joining two files,” on page 439 for details.

SORT control statement



The SORT control statement must be used when a sorting application is performed; this statement describes the control fields in the input records on which the program sorts. A SORT statement can also be used to specify a copy application. User labels will not be copied to the output data sets.

The way in which DFSORT processes short SORT control fields depends on the setting for VLSHRT/NOVLSHRT. A short field is one where the variable-length record is too short to contain the entire field, that is, the field extends beyond the record. For details about sorting short records, see the discussion of the VLSHRT and NOVLSHRT options in “OPTION control statement” on page 173.

The options available on the SORT statement can be specified in other sources as well. A table showing all possible sources for these options and the order of override is given in Appendix B, “Specification/override of DFSORT options,” on page 805.

When an option can be specified on either the SORT or OPTION statement, it is preferable to specify it on the OPTION statement.

DFSORT accepts but does not process the following SORT operands: WORK=value and ORDER=value.

DFSORT's collating behavior can be modified according to your cultural environment. The cultural environment is established by selecting the active locale. The active locale's collating rules affect SORT processing as follows:

- DFSORT produces sorted records for output according to the collating rules defined in the active locale. This provides sorting for single- or multi-byte character data, based on defined collating rules that retain the cultural and local characteristics of a language.

If locale processing is to be used, the active locale will only be used to process character (CH) control fields.

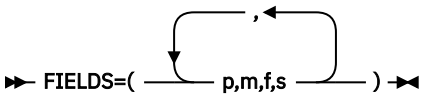
For more information on locale processing, see "Cultural environment considerations " on page 6 or LOCALE in "OPTION control statement" on page 173.

DFSORT's Unicode collation behavior can be modified according to your cultural environment. The cultural environment is established by selecting the active collation version. The active collation version rules affects SORT processing as follows:

- DFSORT produces sorted records for output according to the collating version rules defined in the active collation version. This provides sorting for single-or multi-byte character data based on defined collating version rules that retain the cultural and local characteristics of a language.

See "Unicode Environment Considerations" on page 6 or COLLKEY in "OPTION control statement" on page 173.

FIELDS



Requires four facts about each control field in the input records: the position of the field within the record, the length of the field, the format of the data in the field, and the sequence into which the field is to be sorted. These facts are communicated to DFSORT by the values of the FIELDS operand, represented by p, m, f, and s.

The value for f can optionally be specified by the FORMAT=f parameter as explained later in this section.

All control fields must be located within the first 32752 bytes of a record.

The maximum length of the collected control fields for which Blockset can be used is 4088 bytes. However, the maximum length can be less than 4088 for various situations, such as the use of certain formats (for example, PD), the EQUALS option, and so on. If this maximum is exceeded, Blockset cannot be used.

The FIELDS operand can be written in two ways.

The program examines the major control field first, and it must be specified first. The minor control fields are specified following the major control field. p, m, f, and s describe the control fields. The text that follows gives specifications in detail.

p
specifies the first byte of a control field relative to the beginning of the input record. [14]

The first data byte of a fixed-length record has relative position 1. The first data byte of a variable-length record has relative position 5. The first 4 bytes contain the record descriptor word. All

[14] If INREC is specified, p must refer to the record as reformatted by INREC. If your E15 user exit reformats the record, and INREC is not specified, p must refer to the record as reformatted by your E15 user exit.

control fields, except binary, must begin on a byte boundary. The first byte of a floating-point field is interpreted as a signed exponent; the rest of the field is interpreted as the fraction.

Fields containing binary values are described in a "bytes.bits" notation as follows:

1. First, specify the byte location relative to the beginning of the record and follow it with a period.
2. Then, specify the bit location relative to the beginning of that byte. Remember that the first (high-order) bit of a byte is bit 0 (not bit 1); the remaining bits are numbered 1 through 7.

Thus, 1.0 represents the beginning of a record. A binary field beginning on the third bit of the third byte of a record is represented as 3.2. When the beginning of a binary field falls on a byte boundary (say, for example, on the fourth byte), you can write it in one of three ways:

```
4.0
4.
4
```

Other examples of this notation are shown in Figure 8 on page 425:

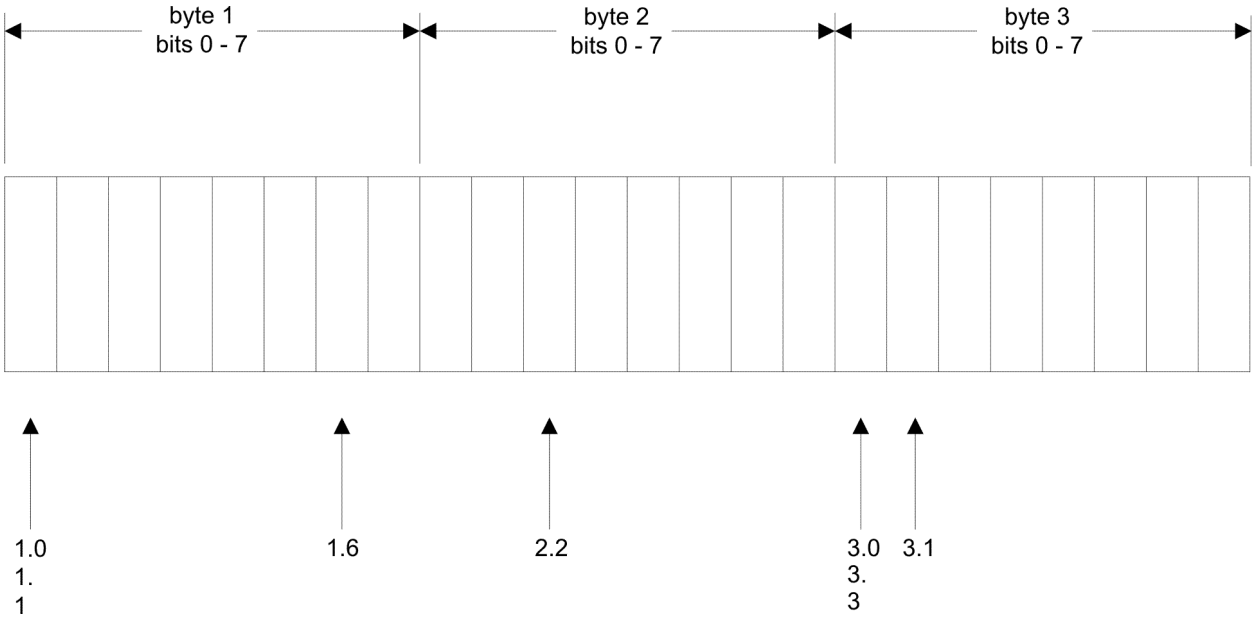


Figure 8. Examples of Notation for Binary Fields

m

specifies the length of the control field. Values for all control fields except binary fields must be expressed in integer numbers of bytes. Binary fields can be expressed in the notation “bytes.bits”. The length of a binary control field that is an integer value (d) can be expressed in one of three ways:

```
d.0
d.
d
```

The number of bits specified must not exceed 7. A control field 2 bits long would be represented as 0.2.

The total number of bytes occupied by all control fields must not exceed 4092 (or, when the EQUALS option is in operation, 4088 bytes). When you determine the total, count a binary field as occupying an entire byte if it occupies any part of it. For example, a binary field that begins on byte 2.6 and is 3 bits long occupies two bytes. All fields must be completely contained within the first 32752 bytes of the record.

f

specifies the format of the data in the control field. Acceptable control field lengths (in bytes) and available formats are shown in Table 64 on page 426.

Table 64. Control Field Formats and Lengths

Control Field Formats and LengthsFormat	Length	Description
CH	1 to 4092 bytes	Character [15]
AQ	1 to 4092 bytes	Character with alternate collating sequence
UTF8	1 to 450 Bytes	Unicode Transformation Format, a 8 bit encoding form
UTF16	1 to 450 Bytes	Unicode Transformation Format, a 16 bit encoding form
UTF32	1 to 450 Bytes	Unicode Transformation Format, a 32 bit encoding form
ZD	1 to 256 bytes	Signed zoned decimal
PD	1 to 256 bytes	Signed packed decimal
PD0	2 to 8 bytes	Packed decimal with sign and first digit ignored
FI	1 to 256 bytes	Signed fixed-point
BI	1 bit to 4092 bytes	Unsigned binary
FL	1 to 256 bytes	Signed hexadecimal floating-point
AC	1 to 4092 bytes	ASCII character
CSF or FS	1 to 32 bytes	Signed numeric with optional leading floating sign
UFF	1 to 44 bytes	Unsigned free form numeric
SFF	1 to 44 bytes	Signed free form numeric
CSL or LS	2 to 256 bytes	Signed numeric with leading separate sign
CST or TS	2 to 256 bytes	Signed numeric with trailing separate sign
CLO or OL	1 to 256 bytes	Signed numeric with leading overpunch sign
CTO or OT	1 to 256 bytes	Signed numeric with trailing overpunch sign
ASL	2 to 256 bytes	Signed ASCII numeric with leading separate sign
AST	2 to 256 bytes	Signed ASCII numeric with trailing separate sign
D1	1 to 4092 bytes	User-defined data type (requires an EFS program)
Y2T	3 to 6 bytes	Character or zoned yyx...x full date format with special indicators

[15] If CHALT is in effect, CH is treated as AQ.

Table 64. Control Field Formats and Lengths (continued)

Control Field Formats and LengthsFormat	Length	Description
Y2U	2 or 3 bytes	Packed decimal yyx and yyxxx full date format with special indicators
Y2V	3 or 4 bytes	Packed decimal yyxx and yyxxxx full date format with special indicators
Y2W	3 to 6 bytes	Character or zoned x...xyy full date format with special indicators
Y2X	2 or 3 bytes	Packed decimal xyy and xxxyy full date format with special indicators
Y2Y	3 or 4 bytes	Packed decimal xxyy and xxxxyy full date format with special indicators
Y2C or Y2Z	2 bytes	Two-digit character or zoned-decimal year data
Y2P	2 bytes	Two-digit packed-decimal year data
Y2D	1 byte	Two-digit decimal year data
Y2S	2 bytes	Two-digit character or zoned-decimal year data with special indicators
Y2B	1 byte	Two-digit binary year data
AUF	1 to 44 bytes	ASCII Unsigned free form numeric
ASF	1 to 44 bytes	ASCII Signed free form numeric

Note: See Appendix C, “Data format descriptions,” on page 823 for detailed format descriptions.

AUF, ASF, CSF, FS, UFF, SFF, Y2 and PD0 format fields can only be used if Blockset is selected.

For Y2 format fields, real dates are collated using the century window established by the Y2PAST option in effect, but the century window is not used for special indicators. Thus the Y2 formats will collate real dates and special indicators as follows:

- Y2T and Y2W:

  Ascending:
  BI zeros, blanks, CH/ZD zeros, lower century dates (for example, 19yy), upper century dates (for example, 20yy), CH/ZD nines, BI ones.

  Descending:
  BI ones, CH/ZD nines, upper century dates (for example, 20yy), lower century dates (for example, 19yy), CH/ZD zeros, blanks, BI zeros.

- Y2U, Y2V, Y2X and Y2Y:

  Ascending:
  PD zeros, lower century dates (for example, 19yy), upper century dates (for example, 20yy), PD nines.

  Descending:
  PD nines, upper century dates (for example, 20yy), lower century dates (for example, 19yy), PD zeros.

- Y2C, Y2Z, Y2P, Y2D and Y2B:

  Ascending:
  Lower century years (for example, 19yy), upper century years (for example, 20yy).

Descending:
Upper century years (for example, 20yy), lower century years (for example, 19yy).

- Y2S:

Ascending:
BI zeros, blanks, lower century years (for example, 19yy), upper century years (for example, 20yy), BI ones.

Descending:
BI ones, upper century years (for example, 20yy), lower century years (for example, 19yy), blanks, BI zeros.

The AC format sequences **EBCDIC data** using the ASCII collating sequence shown in Appendix D. For example, if AC is used with ascending sequence, the EBCDIC numbers (0-9) will collate before the EBCDIC uppercase letters (A-Z) which in turn will collate before the EBCDIC lowercase letters (a-z).

You can use p,m,s rather than p,m,f,s if you use FORMAT=f to supply the format for the field, as described later in this section.

All floating-point data must be normalized before the program can collate it properly. You can use an E15 or E61 user exit to do this during processing. If you use E61, specify the E option for the value of s in the FIELDS operand for each control field you are going to modify with this user exit.

s
specifies how the control field is to be ordered. The valid codes are:

A
ascending order

D
descending order

E
control fields to be modified

Specify E if you include an E61 user exit to modify control fields before the program sorts them. After an E61 user exit modifies the control fields, DFSORT collates the records in ascending order using the formats specified. [16]

For information on how to add a user exit, see Chapter 5, “Using your own user exit routines,” on page 467.

Default: None; must be specified. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

FORMAT

FORMAT=f can be used to specify a particular format for one or more control fields. f from FORMAT=f is used for p,m,s fields. f from FORMAT=f is ignored for p,m,f,s fields. For example, the following are all equivalent:

```
SORT FIELDS=(5,5,ZD,A,12,6,PD,D,21,3,PD,A,35,7,ZD,A)

SORT FORMAT=ZD,FIELDS=(5,5,A,12,6,PD,D,21,3,PD,A,35,7,A)

SORT FIELDS=(5,5,ZD,A,12,6,D,21,3,A,35,7,ZD,A),FORMAT=PD
```

[16] With a conventional merge or a tape work data set sort, control fields for which E is specified are treated as binary byte format regardless of the actual formats specified.

The permissible field formats are shown under the description of 'f' for fields.

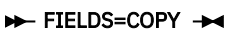
If you have specified the COPY operand, FORMAT=f cannot be specified.

Default: None; FORMAT=f must be specified if any field is specified as p,m,s rather than p,m,f,s. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

Note: DFSORT issues an informational message and ignores FORMAT=f if all of the fields are specified as p,m,f,s.

FIELDS=COPY

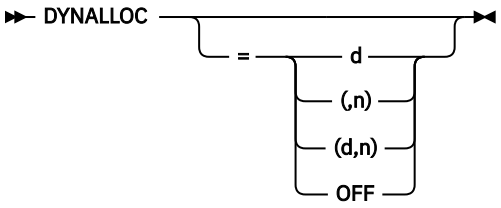


See the discussion of the COPY option discussed in “OPTION control statement” on page 173.

CKPT

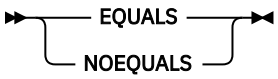
See the discussion of this option discussed in “OPTION control statement” on page 173.

DYNALLOC



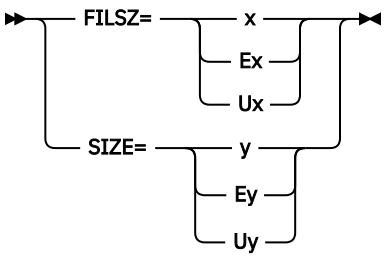
See the discussion of this option in “OPTION control statement” on page 173.

EQUALS or NOEQUALS



See the discussion of these options in “OPTION control statement” on page 173.

FILSZ or SIZE



See the discussion of these options in “OPTION control statement” on page 173.

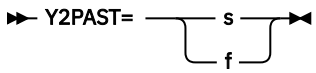
SKIPREC

See the discussion of this option in "OPTION control statement" on page 173.

STOPAFT

See the discussion of this option in "OPTION control statement" on page 173.

Y2PAST



See the discussion of this option in "OPTION control statement" on page 173.

Note: CENTURY=value and CENTWIN=value can be used instead of Y2PAST=value.

SORT/MERGE statement notes

- If records are reformatted by INREC (SORT and MERGE) or E15 (SORT), FIELDS must refer to fields in the appropriate reformatted records.
- If SZERO is in effect, -0 collates before +0 in ascending order and after +0 in descending order when numeric fields are sorted or merged. If NOSZERO is in effect, -0 collates equally with +0 when numeric fields are sorted or merged. However, SZERO is always used for a conventional merge or tape work data set sort application.
- If records are being sorted/merged using UTF8/UTF16/UTF32 data formats, then the following operands are not supported.
  - LOCALE
  - EFS
  - SUM
  - E61 exit
  - VLSHRT
  - INREC
  - OUTREC

Specifying a SORT or COPY—examples

Example 1

```
SORT   FIELDS=(2,5,FS,A),FILSZ=29483
```

FIELDS
The control field begins on the second byte of each record in the input data set, is five bytes long, and contains floating sign data. It is to be sorted in ascending order.

FILSZ
The data set to be sorted contains exactly 29483 records.

Example 2

```
SORT   FIELDS=(7,3,CH,D,1,5,FI,A,398.4,7.6,BI,D,99.0,230.2,
         BI,A,452,8,FL,A),DYNALLOC=(3390,4)
```

FIELDS
The first four values describe the major control field. It begins on byte 7 of each record, is 3 bytes long, and contains character (EBCDIC) data. It is to be sorted in descending order.

The next four values describe the second control field. It begins on byte 1, is 5 bytes long, contains fixed-point data, and is to be sorted in ascending order.

The third control field begins on the fifth bit (bits are numbered 0 through 7) of byte 398. The field is 7 bytes and 6 bits long (occupies 9 bytes), and contains binary data to be placed in descending order.

The fourth control field begins on byte 99, is 230 bytes and 2 bits long, and contains binary data. It is to be sorted in ascending order.

The fifth control field begins on byte 452, is 8 bytes long and contains normalized hexadecimal floating-point data, which is to be sorted in ascending order. If the data in this field were not normalized, you could specify E instead of A and include your own E61 user exit routine to normalize the field before the program examined it.

DYNALLOC
Four work data sets are allocated on 3390.

Example 3

```
SORT   FIELDS=(3,8,ZD,E,40,6,CH,D)
```

FIELDS
The first four values describe the major control field. It begins on byte 3 of each record, is 8 bytes long, and contains zoned decimal data that is modified by your routine before sort examines the field.

The second field begins on byte 40, is 6 bytes long, contains character (EBCDIC) data, and is sorted in descending sequence.

Example 4

```
SORT   FIELDS=(7025,4,A,5048,8,A),FORMAT=ZD,EQUALS
```

FIELDS
The major control field begins on byte 7025 of each record, is 4 bytes long, contains zoned decimal data (FORMAT=ZD), and is to be sorted in ascending sequence.

The second control field begins on byte 5048, is 8 bytes long, has the same data format as the first field, and is also to be sorted in ascending order.

FORMAT
FORMAT=ZD is used to supply ZD format for the p,m,s fields and is equivalent to specifying p,m,ZD,s for these fields.

With FORMAT=f, you can mix p,m,s and p,m,f,s fields when that's convenient such as when all or most of the fields have the same format (although you can always code p,m,f,s for all fields and not use FORMAT=f, if you prefer). For example, the following are also valid uses of the FORMAT=f parameter:

```
SORT FORMAT=BI,FIELDS=(21,4,A,5,4,PD,A,31.3,1.4,A,52,20,A)

SORT FIELDS=(16,4,A,22,8,BI,D,3,2,A),FORMAT=FI
```

EQUALS
specifies that the sequence of equal collating records is to be preserved from input to output.

Example 5

```
SORT   FIELDS=COPY
```

FIELDS
The input data set is copied to the output data set without sorting or merging.

Example 6

```
OPTION Y2PAST=1950
SORT FIELDS=(21,6,Y2T,A,13,3,Y2X,D)
```

Y2PAST
Sets a century window of 1950–2049.

FIELDS
Sorts on a C'yymmdd' (or Z'yymmdd') date in positions 21-26 in ascending order, and on a P'dddyy' date in positions 13-15 in descending order. "Real" dates are sorted using the century window of 1950-2049. Special indicators are sorted correctly relative to the "real" dates.

Example 7: Sorting UTF8 data

```
SORT FIELDS=(9,2,UTF8,D)
```

FIELDS
The control field begins on the ninth byte of each record in the input data set, is 2 bytes long, and contains 8 bit encoding Unicode Transformation Format (UTF8) data. It is to be sorted in descending order.

Example 8: Sorting UTF16 data

```
SORT FIELDS=(5,4,FI,A,345,400,UTF16,D,13,2,CH,A)
```

FIELDS
The first four values describe the major control field. It begins on byte 5 of each record, is 4 bytes long, and contains fixed-point data, and is to be sorted in ascending order.

The next four values describe the second control field. It begins on byte 345, is 400 bytes long, contains a 16 bit encoding Unicode Transformation Format (UTF16) data, and is to be sorted in descending order.

The third control field begins on byte 13 is 2 bytes long, and contains character (EBCDIC) data. It is to be sorted in ascending order.

Example 9: Sorting UTF32 data

```
OPTION COLLKEY=UCA600_LFR_RFR
```

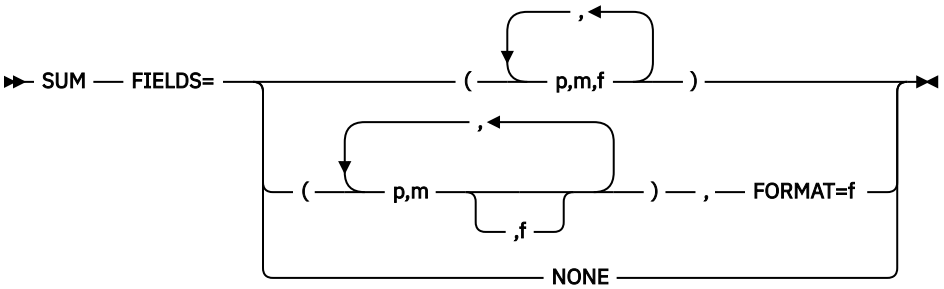
```
SORT FIELDS=(15,50,A), FORMAT=UTF32
```

COLLKEY
The collation version is UCA600 supports the Unicode Standard character suite 6.0.0 and uses Normalization Service under 6.0.0 Unicode character suite. The Language is French and the Region is France.

FIELDS
The control field begins on byte 15 of each record in the input data set, is 50 bytes long, and contains 32 bit encoding Unicode Transformation Format (UTF32) data. It is to be sorted in ascending order.

FORMAT
FORMAT=UTF32 is used to supply UTF32 format for the p,m,s fields and is equivalent to specifying p,m,UTF32,s for these fields.

SUM control statement

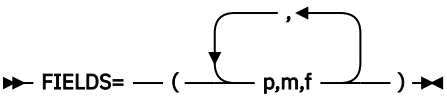


The SUM control statement specifies that, whenever two records are found with equal sort or merge control fields, the contents of their summary fields are to be added, the sum is to be placed in one of the records, and the other record is to be deleted. If the EQUALS option is in effect the **first record** of summed records is kept. If the NOEQUALS option is in effect, the record to be kept is unpredictable. For further details, see "SUM statement notes" on page 435.

If the ZDPRINT option is in effect, positive summed ZD values are printable. If the NZDPRINT option is in effect, positive summed ZD values are not printable. For further details, see "SUM statement notes" on page 435.

The way in which DFSORT processes short SUM summary fields depends on whether the VLSHRT or NOVLSHRT option is in effect. A short field is one where the variable-length record is too short to contain the entire field; that is, the field extends beyond the record. For details about sorting, merging and summing short records, see the discussion of the VLSHRT and NOVLSHRT options in "OPTION control statement" on page 173.

FIELDS



Designates numeric fields in the input record as summary fields.

p
specifies the first byte of the field relative to the beginning of the input record. [17] The first data byte of a fixed-length record has relative position 1. The first data byte of a variable-length record has relative position 5, as the first four bytes are occupied by the RDW. All fields must start on a byte boundary and no field can extend beyond byte 32752

m
specifies the length in bytes of the summary fields to be added. See Table 65 on page 433for permissible length values.

f
specifies the format of the data in the summary field:

Table 65. Summary Field Formats and Lengths

Summary Field Formats and LengthsFormat Code	Length	Description
BI	2, 4, or 8 bytes	Unsigned binary

[17] If INREC is specified, p must refer to the record as reformatted by INREC. If your E15 user exit reformats the record, and INREC is not specified, p must refer to the record as reformatted by your E15 user exit.

Table 65. Summary Field Formats and Lengths (continued)		
Summary Field Formats and LengthsFormat Code	**Length**	**Description**
FI	2, 4, or 8 bytes	Signed fixed-point
FL	4, 8, or 16 bytes	Signed hexadecimal floating-point
PD	1 to 16 bytes	Signed packed decimal
ZD	1 to 31 bytes	Signed zoned decimal

The value for f can optionally be specified by the FORMAT=f parameter as explained later in this section.

Note: See Appendix C, “Data format descriptions,” on page 823 for detailed format descriptions.

NONE
eliminates records with duplicate keys. Only one record with each key is kept and no summing is performed.

Note: The FIRST operand of ICETOOL's SELECT operator can be used to perform the same function as SUM FIELDS=NONE with OPTION EQUALS. Additionally, SELECT's FIRSTDUP, ALLDUPS, NODUPS, HIGHER(x), LOWER(y), EQUAL(v), LASTDUP, and LAST operands can be used to select records based on other criteria related to duplicate and non-duplicate keys. SELECT's DISCARD(savedd) operand can be used to save the records discarded by FIRST, FIRSTDUP, ALLDUPS, NODUPS, HIGHER(x), LOWER(y), EQUAL(v), LASTDUP, or LAST. See “SELECT operator” on page 631 for complete details on the SELECT operator.

Default: None; must be specified.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

FORMAT

FORMAT=f can be used to specify a particular format for one or more summary fields. f from FORMAT=f is used for p,m fields. f from FORMAT=f is ignored for p,m,f fields. For example, the following are all equivalent:

```
SUM FIELDS=(5,5,ZD,12,6,PD,21,3,PD,35,7,ZD)
SUM FORMAT=ZD,FIELDS=(5,5,12,6,PD,21,3,PD,35,7)
SUM FIELDS=(5,5,ZD,12,6,21,3,35,7,ZD),FORMAT=PD
```

The permissible field formats are shown under the description of 'f' for fields.

Default: None. FORMAT=f must be specified if any field is specified as p,m rather than p,m,f. See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

Applicable Functions: See Appendix B, “Specification/override of DFSORT options,” on page 805.

Note: DFSORT issues an informational message and ignores FORMAT=f if all of the fields are specified as p,m,f.

SUM statement notes

- If overflow might occur during summation, INREC can be used to create a larger SUM field in the reformatted input record (perhaps resulting in a larger record for sorting or merging) so that overflow does not occur. “Example 5” on page 437 illustrates this technique.
- An invalid PD or ZD sign or digit results in a data exception (0C7 ABEND); 0-9 are invalid for the sign and A-F are invalid for the digit. For example, a ZD value such as 3.5 (X'F34BF5') results in an 0C7 because "." (X'4B') is treated as an invalid digit. ICETOOL's DISPLAY or VERIFY operator can be used to identify decimal values with invalid digits. ICETOOL's VERIFY operator can be used to identify decimal values with invalid signs.
- Whether or not positive summed ZD results have printable numbers depends on whether NZDPRINT or ZDPRINT is in effect (as set by the ZDPRINT installation option and the NZDPRINT and ZDPRINT parameters of the OPTION statement):
  - If NZDPRINT is in effect, positive summed ZD results do not consist of printable numbers, regardless of whether the original values consisted of printable numbers or not. For example, if X'F2F3F1' (prints as '231') and X'F3F0F6' (prints as '306') are summed, the result with NZDPRINT in effect is X'F5F3C7' (prints as '53G').
  - If ZDPRINT is in effect, positive summed ZD results consist of printable numbers, regardless of whether the original values consisted of printable numbers or not. For example, if X'F2F3C1' (prints as '23A') and X'F3F0F6' (prints as '306') are summed, the result with ZDPRINT in effect is X'F5F3F7' (prints as '537').

  Thus, ZDPRINT must be in effect to ensure that positive summed ZD results are printable.

  Unsummed positive ZD values retain their original signs, regardless of whether NZDPRINT or ZDPRINT is in effect. For example, if X'F2F8C5' is not summed, it remains X'F2F8C5' (prints as '28E'). OUTFIL's OUTREC parameter can be used to ensure that all summed or unsummed ZD values are printable, as illustrated by Example 4 later in this section.
- If input records are reformatted by INREC or E15, SUM must refer to fields in the appropriate reformatted record (see the preceding description of *p*).
- Summary fields must not be control fields. They must not overlap control fields, or each other, and must not overlap the RDW.
- FL (hexadecimal floating-point) values to be summed can be normalized or unnormalized. However, the resulting FL values are always normalized. Normalization processing by the hardware can produce different sums for FL values summed in different orders.
- Exponent overflow for summed FL values results in an exponent overflow exception (0CC ABEND)
- Exponent underflow for summed FL values results in a true zero result.
- When records are summed, you can predict which record is to receive the sum (and be retained) and which record is to be deleted only when EQUALS is in effect, overflow does not occur, and the BLOCKSET technique is used. In this case, the first record (based on the sequence described under the discussion of the EQUALS or NOEQUALS parameter of the “OPTION control statement” on page 173) is chosen to contain the sum.

  Fields other than summary fields remain unchanged and are taken from the record that receives the sum.
- You can control the action that DFSORT takes when overflow occurs for BI, FI, PD or ZD values with the OVFLO parameter as described in “OPTION control statement” on page 173.
-
- DFSORT issues a message and terminates processing if a SUM statement is specified for a tape work data set sort or Conventional merge.
- DFSORT does not support the XSUM parameter provided by a competitive sort product to write records deleted by SUM processing to a SORTXSUM DD data set. However, ICETOOL's SELECT operator can perform the same function as XSUM with FIELDS=NONE. For example, this ICETOOL job:

```
//S1EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//SORTIN DD DSN=...
//SORTOUT DD DSN=...
//SORTXSUM DD DSN=...
//TOOLIN DD *
SELECT FROM(SORTIN) TO(SORTOUT)-
  ON(5,4,CH) FIRST DISCARD(SORTXSUM)
/*
```

is equivalent to this XSUM job:

```
//S1 EXEC PGM=ICEMAN
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=...
//SORTOUT DD DSN=...
//SORTXSUM DD DSN=...
//SYSIN DD *
  SORT FIELDS=(5,4,CH,A)
  SUM FIELDS=NONE,XSUM
/*
```

Tip: You can also perform additional functions with ICETOOL's SELECT operator that are not available with XSUM. See Chapter 7, “Using ICETOOL,” on page 537 for complete details of ICETOOL's SELECT operator.

Adding summary fields—examples

Example 1

```
  SUM FIELDS=(15021,8,PD,15011,4,FI)
```

This statement designates an 8-byte packed decimal field at byte 15021, and a 4-byte fixed-integer field at byte 15011, as summary fields.

Example 2

```
  SUM FIELDS=NONE
```

This statement illustrates the elimination of duplicate records.

Example 3

```
  SUM FIELDS=(41,8,49,4),FORMAT=ZD
  OPTION ZDPRINT
```

These statements illustrate the use of the FORMAT operand and the ZDPRINT option. The SUM statement designates two zoned decimal fields, one 8 bytes long starting at byte 41, and the other 4 bytes long starting at byte 49. As a result of the ZDPRINT option, the positive summed ZD values will be printable. Note, however, that the ZDPRINT option does not affect ZD values which are not summed due to overflow or unique keys. The next example shows how to use OUTFIL to make all summary fields printable.

Example 4

```
SUM FIELDS=(41,8,49,4),FORMAT=ZD
OUTFIL OUTREC=(1,40,41,8,ZD,M11,49,4,ZD,M11,53,28)
```

These statements illustrate the use of the OUTFIL statement to ensure that all positive ZD summary fields in the output data set are printable. Whereas the ZDPRINT option affects only positive summed ZD fields, OUTFIL can be used to edit positive or negative BI, FI, PD, or ZD values, whether they are summed or not.

OUTFIL can also be used to produce multiple output data sets, reports, and so on. See "OUTFIL control statements" on page 221 for complete details about OUTFIL processing.

Note: For purposes of illustration, this example assumes that the input records are 80 bytes long.

Example 5

```
* Add Z'0' before the ZD SUM field to prevent overflow.
* Add P'0' before the PD SUM field to prevent overflow.
 INREC FIELDS=(1,10,  Copy bytes before ZD SUM field
   11:C'0',12:11,4,   Add Z'0' before ZD SUM field
   16:15,6,           Copy bytes after ZD SUM field
   22:X'00',23:21,2,  Add P'0' before PD SUM field
   25:23,5,           Copy SORT field
   30:28,53)          Copy bytes after SORT field
* Sort on key in its new position.
 SORT FIELDS=(25,5,CH,A)
* Sum on the expanded ZD and PD fields in
* their new positions.
 SUM FIELDS=(11,5,ZD,22,3,PD)
```

These statements illustrate a technique for preventing overflow of summed fields by using INREC to make the fields larger before they are summed.

The fields that might overflow when they are summed are a 4 byte ZD field starting at position 11 and a 2 byte PD field starting at position 21. In order to prevent them from overflowing, we expand each field on the left with an appropriate zero byte; C'0' (Z'0') for the ZD field and P'0' (X'00') for the PD field. We can then sum on the new 5 byte ZD field and on the new 3 byte PD field.

Note that adding these extra bytes increases the length of the record and changes the starting position of various fields. In the SORT and SUM statements, we must specify the starting positions of the fields in the reformatted record rather than the starting positions of the fields in the input record. For example, although the SORT field starts in position 23 in the input record, we must use its starting position of 25 in the reformatted record.

Chapter 4. Using a JOINKEYS application for joining two files

Overview

You can perform various types of "join" applications on two files (F1 and F2) by one or more keys with DFSORT using the following statements:

JOINKEYS
You must specify two JOINKEYS statements; one for the F1 file and another for the F2 file. A separate subtask will be used to process each file and a main task will be used to process the joined records from the two files.

Each JOINKEYS statement must specify the ddname of the file it applies to and the starting position, length and sequence of the keys in that file. You can also optionally specify if the file is already sorted by the keys and if sequence checking of the keys is not needed; if the file has fixed-length or variable-length records; to stop reading the file after n records; a 2-byte id to be used for the message and control data set for the subtask used to process the file, and if a subset of the records is to be processed based on a logical expression.

JOIN
If you don't specify a JOIN statement, only paired records from F1 and F2 are kept and processed by the main task as the joined records (inner join). You can optionally specify a JOIN statement to have the main task keep and process: unpaired F1 records as well as paired records (left outer join); unpaired F2 records as well as paired records (right outer join); unpaired F1 and F2 records as well as paired records (full outer join); only unpaired F1 records; only unpaired F2 records, or only unpaired F1 and F2 records.

REFORMAT
You would normally specify a REFORMAT statement to indicate the F1 and/or F2 fields you want in the joined records. You can optionally specify an indicator of where the key was found, and a FILL character to be used for missing bytes. If a JOIN statement with ONLY is specified, the REFORMAT statement is optional.

F1 and F2 can be any type of sequential or VSAM file supported by DFSORT for SORTIN and can have different attributes (for example, F1 can have RECFM=FB and LRECL=100 and F2 can have RECFM=VB and LRECL=254, or F1 can be a VSAM ESDS and F2 can be a PDS member).

F1 will be sorted or copied by "subtask1". An E35 exit will be used to pass the needed fields from the F1 records to the "main task" (an intermediate output data set is not used or required). A subset of the DFSORT statements, such as INCLUDE, OMIT, INREC, SUM and OPTION, will be available for processing the F1 records.

F2 will be sorted or copied by "subtask2". An E35 exit will be used to pass the needed fields from the F2 records to the "main task" (an intermediate output data set is not used or required). A subset of the DFSORT statements, such as INCLUDE, OMIT, INREC, SUM and OPTION, will be available for processing the F2 records.

The "main task" will use an E15 to join the records passed from the E35 of subtask1 and E35 of subtask2. The joined records will be processed as the input records for a sort or copy application. Most of the control statements and options available for a DFSORT sort or copy application will be available for processing the joined records.

Note: Since a JOINKEYS application uses three tasks, it can require more storage than a regular DFSORT application. You may need to use REGION=0M for some JOINKEYS applications.

JOINKEYS application processing

Figure 9 on page 440 is a pictorial representation of the processing performed for a JOINKEYS application, and the order in which the various functions are performed.

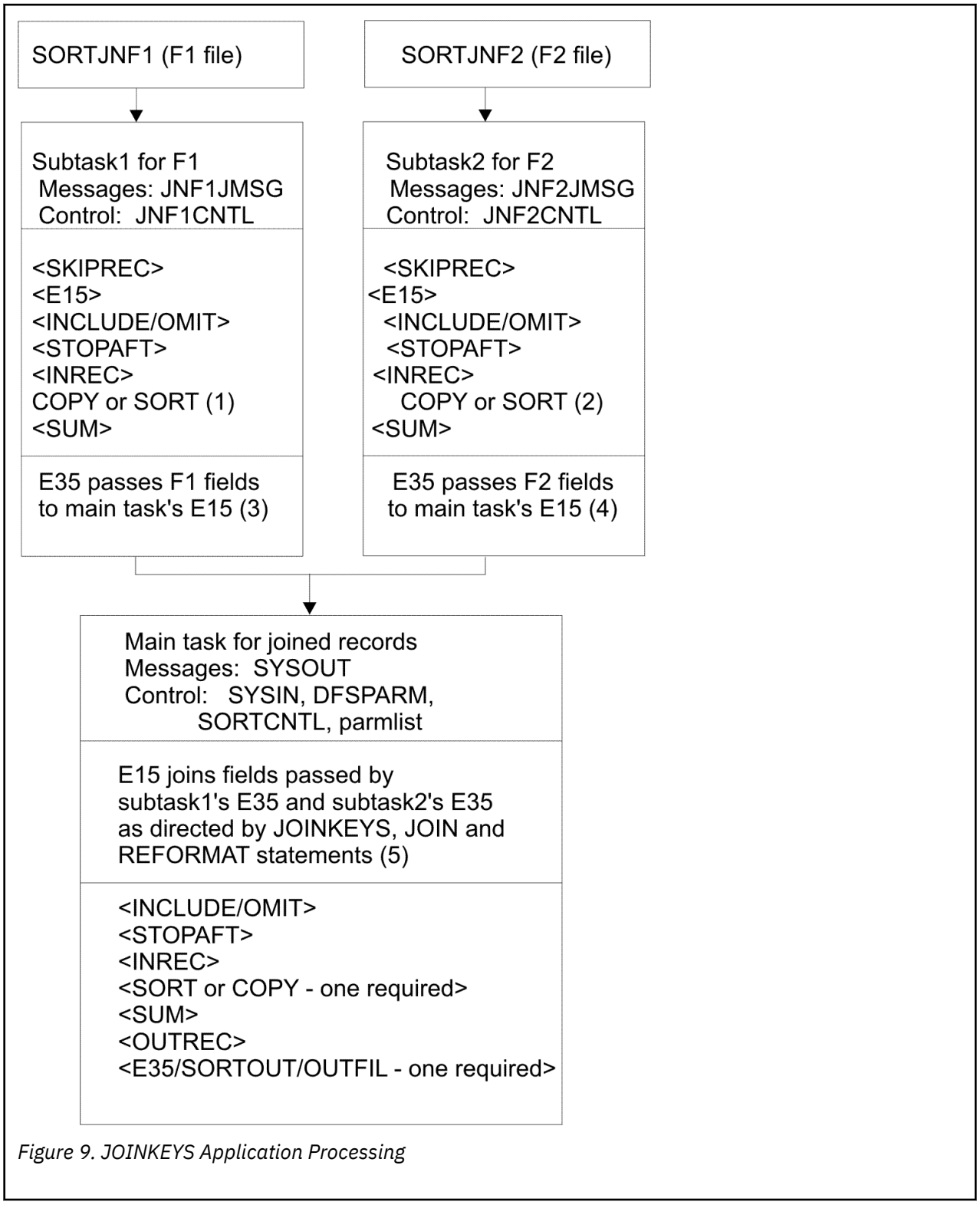


Figure 9. JOINKEYS Application Processing

Legend for Figure 9 on page 440

- (1) COPY is used automatically for subtask1 if the SORTED operand is specified in the F1 JOINKEYS statement. Otherwise, SORT is used automatically with the binary keys specified in the FIELDS operand of the F1 JOINKEYS statement.
- (2) COPY is used automatically for subtask2 if the SORTED operand is specified in the F2 JOINKEYS statement. Otherwise, SORT is used automatically with the binary keys specified in the FIELDS operand of the F2 JOINKEYS statement.
- (3) An E35 exit is used for subtask1 automatically. (An intermediate output file is not required or used.)
- (4) An E35 exit is used for subtask2 automatically. (An intermediate output file is not required or used.)
- (5) An E15 exit is used for the main task automatically. (Intermediate input files are not required or used.)

For direct invocation of DFSORT (for example, PGM=SORT), the JOINKEYS, JOIN and REFORMAT statements can be specified in SYSIN or DFSPARM. For program invocation of DFSORT (for example, LINK EP=SORT), the JOINKEYS, JOIN and REFORMAT statements can be specified in the caller's parameter list, in SORTCNTL or in DFSPARM.

Subtask1 reads the F1 file. It uses COPY or SORT, and an E35 exit (with no output data set) automatically to pass the needed F1 fields to the main task's E15 exit. Subtask1 can optionally use the other listed control statements from JNF1CNTL. Subtask1 messages are displayed in JNF1JMSG.

Subtask2 reads the F2 file. It uses COPY or SORT, and an E35 exit (with no output data set) automatically to pass the needed F2 fields to the main task's E15 exit. Subtask2 can optionally use the other listed control statements from JNF2CNTL. Subtask2 messages are displayed in JNF2JMSG.

The main task uses an E15 exit automatically. The E15 creates the joined records by accessing the F1 fields passed by subtask1's E35 and the F2 fields passed by subtask2's E35. The main task writes the output to SORTOUT and/or OUTFIL or uses a supplied E35 exit to "delete" all of the records. For direct invocation of DFSORT, the main task can optionally use the other listed control statements from SYSIN or DFSPARM. For program invocation of DFSORT, the main task can optionally use the other listed control statements from the caller's parameter list, in SORTCNTL or in DFSPARM. The main task's messages are displayed in SYSOUT.

The starting position in the fields you specify for the subtasks or main task must reflect any reformatting of the records you do at each stage. This includes using the starting positions of the joined records for main task functions.

Examples:

- If you specify INREC and SUM statements in JNF1CNTL for subtask1, the FIELDS operands in the JOINKEYS, REFORMAT, and SUM statements must specify the starting positions in the F1 records as reformatted by INREC.
- If you specify an INCLUDE statement in SYSIN for the main task, the COND operand must specify the starting positions in the joined records as reformatted by REFORMAT.

Sample JOINKEYS applications

Here are the JCL and control statements for a simple JOINKEYS application to do an inner join (also known as a cartesian join) which joins all paired records.

```
//S1   EXEC  PGM=SORT
//SYSOUT DD SYSOUT=*
//SORTJNF1 DD DSN=INPUT1,DISP=SHR
//SORTJNF2 DD DSN=INPUT2,DISP=SHR
//SORTOUT DD SYSOUT=*
//SYSIN DD *
* Control statements for JOINKEYS application
  JOINKEYS FILE=F1,FIELDS=(15,2,A,7,4,A)
  JOINKEYS FILE=F2,FIELDS=(21,2,A,23,4,A)
  REFORMAT FIELDS=(F2:1,70,F1:1,60)
* Control statements for main task (joined records)
  SORT FIELDS=COPY
/*
```

Here are the JCL and control statements for a JOINKEYS application which omits certain records and normalizes keys for F1 and F2, and retains only sorted, non-duplicate, unpaired records from F1. The F1 and F2 files are already in order by the specified JOINKEYS FIELDS. DFSORT symbols are used for various fields and constants.

```
//S2   EXEC  PGM=SORT
//SYSOUT DD SYSOUT=*
//SYMNAMES DD *
IN1_dept,11,3,ch
IN1_target,'J82'
IN1_normal_key,27,5,ZD
IN1_output,1,60
IN2_dept,14,3,ch
IN2_target,'M25'
IN2_PD_key,21,3,PD
IN2_normal_key,=,5,ZD
join_sort_key,45,8,UFF
//FIX DD DSN=F.INPUT,DISP=SHR
//VAR DD DSN=V.INPUT,DISP=SHR
//SORTOUT DD DSN=FIXOUT1,DISP=(NEW,CATLG,DELETE),
// SPACE=(CYL,(5,5)),UNIT=SYSDA
//JNF1CNTL DD *
* Control statements for subtask1 (F1)
  OMIT COND=(IN1_dept,EQ,IN1_target)
  INREC OVERLAY=(IN1_normal_key:IN1_normal_key,TO=ZDF,LENGTH=5)
//JNF2CNTL DD *
* Control statements for subtask2 (F2)
  OMIT COND=(IN2_dept,EQ,IN2_target)
  INREC OVERLAY=(IN2_PD_key:IN2_PD_key,TO=ZDF,LENGTH=5)
//SYSIN DD *
* Control statements for JOINKEYS application
  JOINKEYS F1=FIX,FIELDS=(IN1_normal_key,D),SORTED
  JOINKEYS F2=VAR,FIELDS=(IN2_normal_key,D),SORTED
  JOIN UNPAIRED,F1,ONLY
  REFORMAT FIELDS=(F1:IN1_output)
* Control statements for main task (joined records)
  SORT FIELDS=(join_sort_key,A)
  SUM FIELDS=NONE
/*
```

JCL for a JOINKEYS application

The required JCL statements used for a JOINKEYS application are as follows. See *z/OS DFSORT Application Programming Guide* for general information on DFSORT message, control, input, output, work and symbols data sets.

//stepname EXEC PGM=SORT
Invokes DFSORT. PGM=ICEMAN can also be used.

Note: Since a JOINKEYS application uses three tasks, it can require more storage than a regular DFSORT application. You may need to use REGION=0M for some JOINKEYS applications.

//SYSOUT DD
Messages from the main task. An alternate ddname can be supplied with MSGDDN=ddname in // DFSPARM.

//SORTJNF1 DD
F1 input file (read by subtask1). The ddname is SORTJNF1 (or ccccJNF1 if SORTDD=cccc is in effect) if FILE=F1 or FILES=F1 is specified on the JOINKEYS statement. An alternate ddname can be supplied with F1=ddname on the JOINKEYS statement.

Note: The F1 data set is treated as a SORTIN data set by subtask1 and is subject to the rules for a SORTIN data set documented in Chapter 2, “Invoking DFSORT with Job Control Language,” on page 27.

//SORTJNF2 DD
F2 input file (read by subtask2). The ddname is SORTJNF2 (or ccccJNF2 if SORTDD=cccc is in effect) if FILE=F2 or FILES=F2 is specified on the JOINKEYS statement. An alternate ddname can be supplied with F2=ddname on the JOINKEYS statement.

Note: The F2 data set is treated as a SORTIN data set by subtask2 and is subject to the rules for a SORTIN data set documented in Chapter 2, "Invoking DFSORT with Job Control Language," on page 27.

//SORTOUT DD or //outfil DD
Output of joined records (written by main task). An alternate ddname can be supplied with SORTOUT=ddname or SORTDD=cccc in //DFSPARM or with the OUTFIL statement in //SYSIN or //DFSPARM.

//DFSPARM DD, //SYSIN DD or //SORTCNTL DD
Control statements for the main task including JOINKEYS, JOIN, REFORMAT, OPTION, SORT, INCLUDE or OMIT, SUM, OUTREC, RECORD, ALTSEQ, MODS and OUTFIL. If SKIPREC or E15 is specified, it will not be used.

DFSPARM can be used if DFSORT is invoked directly or called from a program. SYSIN can be used if DFSORT is invoked directly. SORTCNTL can be used if DFSORT is called from a program; an alternate ddname of ccccCNTL can be supplied with SORTDD=cccc in DFSPARM.

Note: If DFSORT is called from a program, the control statements for the main task can be supplied using the parameter list.

The optional JCL statements used for a JOINKEYS application are as follows:

SYMNAMES DD
DFSORT symbols for the main task, subtask1 and subtask2.

SYMNOUT DD
Listing of symbols supplied via the SYMNAMES DD.

JNF1JMSG DD
Messages from subtask1. This message data set will be dynamically allocated with SYSOUT=*, RECFM=FBA, LRECL=121 and BLKSIZE=121 if a JNF1JMSG DD statement is not supplied. If a JNF1JMSG DD statement is supplied, this message data set will be given the same attributes as a //SYSOUT message data set. An alternate ddname of idF1JMSG can be supplied with TASKID=id on the JOINKEYS statement for F1.

JNF2JMSG DD
Messages from subtask2. This message data set will be dynamically allocated with SYSOUT=*, RECFM=FBA, LRECL=121 and BLKSIZE=121 if a JNF2JMSG DD statement is not supplied. If a JNF2JMSG DD statement is supplied, this message data set will be given the same attributes as a //SYSOUT message data set. An alternate ddname of idF2JMSG can be supplied with TASKID=id on the JOINKEYS statement for F2.

JNF1CNTL DD
Control statements for subtask1 including INCLUDE or OMIT, OPTION, MODS, RECORD, ALTSEQ, INREC and SUM. If E35 is specified, it will not be used. The following control statements must not be specified: JOINKEYS, JOIN, REFORMAT, MERGE, OUTFIL, OUTREC or SORT.

An alternate ddname of idF1CNTL can be supplied with TASKID=id on the JOINKEYS statement for F1.

JNF2CNTL DD
Control statements for subtask2 including INCLUDE or OMIT, OPTION, MODS, RECORD, ALTSEQ, INREC and SUM. If E35 is specified, it will not be used. The following control statements must not be specified: JOINKEYS, JOIN, REFORMAT, MERGE, OUTFIL, OUTREC or SORT.

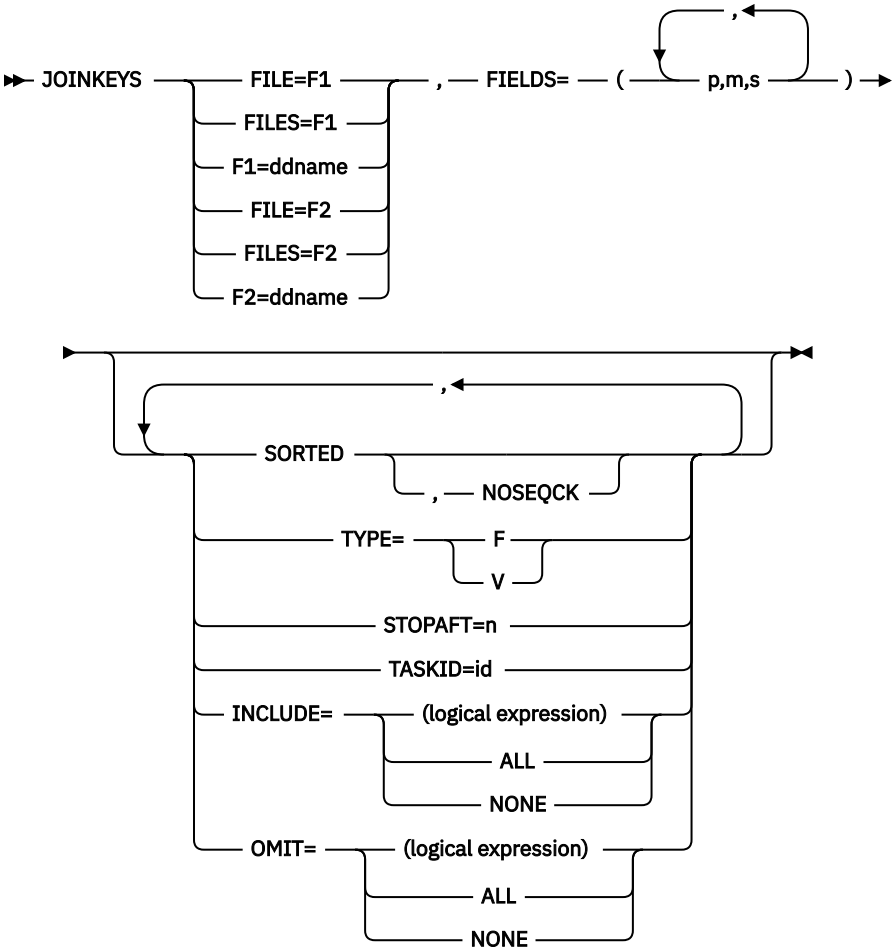
An alternate ddname of idF2CNTL can be supplied with TASKID=id on the JOINKEYS statement for F2.

SORTWKdd DD
Work data sets for the main task. Not recommended since dynamic allocation of work data sets is preferred. Alternate ddnames of ccccWKdd can be supplied with SORTDD=cccc in //DFSPARM.

JNF1WKdd DD
Work data sets for subtask1. Not recommended since dynamic allocation of work data sets will be enabled. Alternate ddnames of idF1WKdd can be supplied with TASKID=id on the JOINKEYS statement for F1.

JNF2WKdd DD
Work data sets for subtask2. Not recommended since dynamic allocation of work data sets will be enabled. Alternate ddnames of idF2WKdd can be supplied with TASKID=id on the JOINKEYS statement for F2.

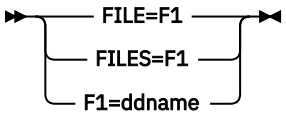
JOINKEYS statements



Two JOINKEYS statements are required for a JOINKEYS application; one for the F1 file and the other for the F2 file:

- FILE=F1 or F1=ddname must be used to indicate that the JOINKEYS statement applies to the F1 input file. FILE=F1 associates the F1 file with a ddname of SORTJNF1. You can use a different ddname for the F1 file by specifying F1=ddname. For simplicity, we will use SORTJNF1 when referring to the ddname for the F1 file.
- FILE=F2 or F2=ddname must be used to indicate that the JOINKEYS statement applies to the F2 input file. FILE=F2 associates the F2 file with a ddname of SORTJNF2. You can use a different ddname for the F2 file by specifying F2=ddname. For simplicity, we will use SORTJNF2 when referring to the ddname for the F2 file.

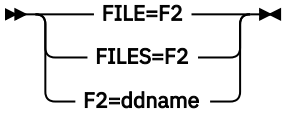
FILE=F1, FILES=F1 or F1=ddname



Must be used for the JOINKEYS statement associated with the F1 file. FILE=F1 (or FILES=F1) specifies a ddname of SORTJNF1 for the F1 file. F1=ddname can be used to specify any valid ddname for the F1 file. Do not use the same ddname for the F1 file and the F2 file.

When FILE=F1, FILES=F1 or F1=ddname is specified, the other operands of the JOINKEYS statement apply to the F1 file.

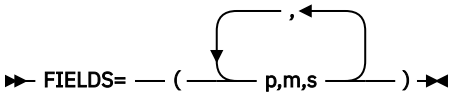
FILE=F2, FILES=F2 or F2=ddname



Must be used for the JOINKEYS statement associated with the F2 file. FILE=F2 (or FILES=F2) specifies a ddname of SORTJNF2 for the F2 file. F2=ddname can be used to specify any valid ddname for the F2 file. Do not use the same ddname for the F1 file and the F2 file.

When FILE=F2, FILES=F2 or F2=ddname is specified, the other operands of the JOINKEYS statement apply to the F2 file.

FIELDS



Must be specified to indicate the starting position, length and order (ascending or descending) of the keys in the input file. The keys will be treated as binary, so they must be "normalized". For example, if the keys are actually zoned decimal, they must have all C and D signs, or all F and D signs. You can use an INREC statement in JNF1CNTL and/or JNF2CNTL to normalize the keys for the F1 file and/or F2 file, respectively, if appropriate.

Each pair of keys for the F1 and F2 files must match with respect to length and order, but can start in different positions. For example, if the first key for the F1 file is 5 bytes ascending and the second key for the F1 file is 3 bytes descending, the first key for the F2 file must be 5 bytes ascending and the second key for the F2 file must be 3 bytes descending.

If a variable-length record is too short to contain a key you specify, the short key value will be compared using binary zeros for the missing bytes.

p
specifies the starting position of the key. The first data byte of a fixed-length record is in position 1. The first data byte of a variable-length record is in position 5 after the 4-byte RDW. p can be 1 to 32752 but all fields must be completely contained within the first 32752 bytes of the record.

m
specifies the length of the key. The total length of all keys must not exceed 4080 bytes. All fields must be completely contained within the first 32752 bytes of the record.

The length for each pair of F1 and F2 keys must match.

s
specifies the order of the key. Use A for ascending or D for descending.

The order for each pair of F1 and F2 keys must match.

For example, if you specify:

```
JOINKEYS F1=IN1,FIELDS=(22,3,A,55,9,D)
JOINKEYS F2=IN2,FIELDS=(15,3,A,1,9,D)
```

File F1 is processed using the ddname IN1, the ascending key in positions 22-24 and the descending key in positions 55-63. File F2 is processed using the ddname IN2, the ascending key in positions 15-17 and the descending key in positions 1-9.

SORTED

By default, DFSORT will sort the input file by the specified keys. If the records of the input file are already in sorted order by the specified keys, you can use the SORTED operand to tell DFSORT to copy the records rather than sort them. This can improve performance. DFSORT will terminate if the copied records are not in the order specified by the keys unless you specify the NOSEQCK operand.

For example, if you specify:

```
JOINKEYS FILE=F1,FIELDS=(22,3,A),SORTED
JOINKEYS FILE=F2,FIELDS=(15,3,A)
```

File F1 is copied using the ddname SORTJNF1 and the ascending key in positions 22-24. The SORTJNF1 records will be checked for the correct key order. File F2 is sorted using the ddname SORTJNF2 and the ascending key in positions 15-17.

If you use the SORTED operand, statements and options only available for a sort application, such as SUM, will be ignored for the subtask that copies the input file.

NOSEQCK

If you specify the SORTED operand and know that the records of the input file are already in sorted order by the specified keys, you can use the NOSEQCK operand to tell DFSORT not to check the order of the records. This can improve performance.

For example, if you specify:

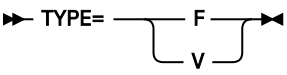
```
JOINKEYS FILE=F1,FIELDS=(22,3,A),SORTED,NOSEQCK
JOINKEYS FILE=F2,FIELDS=(15,3,A),SORTED
```

File F1 is copied using the ddname SORTJNF1 and the ascending key in positions 22-24. The SORTJNF1 records will be not be checked for the correct key order. File F2 is copied using the ddname SORTJNF2 and the ascending key in positions 15-17. The SORTJNF2 records will be checked for the correct key order.

If the records are not actually in order by the specified keys and you use NOSEQCK, the output may be incorrect.

The NOSEQCK operand is ignored if the SORTED operand is not specified.

TYPE



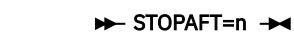
TYPE=V can be used to tell DFSORT to use variable-length processing for a VSAM input file. TYPE=F (the default) can be used to tell DFSORT to use fixed-length processing for a VSAM input file.

For example, if you specify:

```
JOINKEYS F1=VSAM1,FIELDS=(22,3,A),TYPE=V
JOINKEYS F2=VSAM2,FIELDS=(15,3,A),TYPE=F
```

VSAM file F1 is sorted as variable-length records using the ddname VSAM1 and the ascending key in positions 22-24. (Remember that for TYPE=V VSAM processing, DFSORT adds an RDW in positions 1-4 which you must account for when specifying the starting position.) VSAM file F2 is sorted as fixed-length records using the ddname VSAM2 and the ascending key in positions 15-17. (Remember that for TYPE=F VSAM processing, DFSORT does not add an RDW.)

STOPAFT



Can be used to specify the maximum number of records (n) you want the subtask for the input file to accept for sorting or copying (accepted means read from the input file and not deleted by INCLUDE or OMIT).

n
can be up to 28 digits with up to 15 significant digits.

For example, if you specify:

```
  JOINKEYS FILE=F1,STOPAFT=5,FIELDS=(32,4,A)
  JOINKEYS FILE=F2,STOPAFT=10,FIELDS=(32,4,A)
```

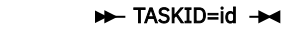
The first 5 input records from SORTJNF1 and the first 10 input records from SORTJNF2 will be processed.

You can use STOPAFT=n on the OPTION statement in JNF1CNTL (for the F1 file) or JNF2CNTL (for the F2 file) instead of specifying STOPAFT=n on the JOINKEYS statement.

For example, instead of the STOPAFT operands in the JOINKEYS statements shown previously, you could specify:

```
//SYSIN DD *
  JOINKEYS FILE=F1,FIELDS=(32,4,A)
  JOINKEYS FILE=F2,FIELDS=(32,4,A)
  ...
//JNF1CNTL DD *
  OPTION STOPAFT=5
//JNF2CNTL DD *
  OPTION STOPAFT=10
```

TASKID



By default, DFSORT uses the following ddnames for the subtasks:

- JNF1JMSG for the subtask1 (F1 file) message data set.
- JNF1CNTL for the subtask1 (F1 file) control data set.
- JNF1WKdd for the subtask1 (F1 file) work data sets.
- JNF2JMSG for the subtask2 (F2 file) message data set.
- JNF2CNTL for the subtask2 (F2 file) control data set.
- JNF2WKdd for the subtask2 (F2 file) work data sets.

The TASKID=id operand can be used to change the first two characters from JN to the specified id characters. The same id can be used for the F1 and F2 ddnames, or a different id can be used for each.

For example, if you specify:

```
  JOINKEYS F1=IN1,FIELDS=(1,10,A),TASKID=C1
  JOINKEYS F2=IN2,FIELDS=(22,10,A),TASKID=C1
```

C1F1JMSG, C1F1CNTL and C1F1WKdd will be used for subtask1 (F1 file). C1F2JMSG, C1F2CNTL and C1F2WKdd will be used for subtask2 (F2 file).

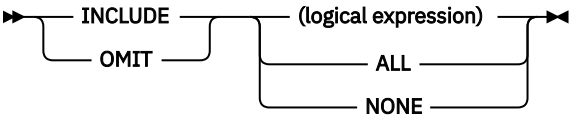
If you specify:

```
  JOINKEYS F1=IN1,FIELDS=(1,10,A),TASKID=I1
  JOINKEYS F2=IN2,FIELDS=(22,10,A),TASKID=I2
```

I1F1JMSG, I1F1CNTL and I1F1WKdd will be used for subtask1 (F1 file). I2F2JMSG, I2F2CNTL and I2F2WKdd will be used for subtask2 (F2 file).

The TASKID=id operand can be useful when you are doing multiple JOINKEYS applications and want to separate the messages for each application and/or specify different control statements or work data sets for different subtasks.

INCLUDE or OMIT



Can be used to specify criteria for the records you want the subtask for the input file to include or omit for sorting or copying. You can use the same logical expressions, ALL or NONE in the same way as for the INCLUDE or OMIT operand of the OUTFIL statement. See "OUTFIL control statements" on page 221 for details.

For example, if you specify:

```
  JOINKEYS FILE=F1,FIELDS=(35,8,A),
     OMIT=(5,4,CH,EQ,C'ABCD')
  JOINKEYS FILE=F2,FIELDS=(37,8,A),
     INCLUDE=(1,20,SS,EQ,C'NO')
```

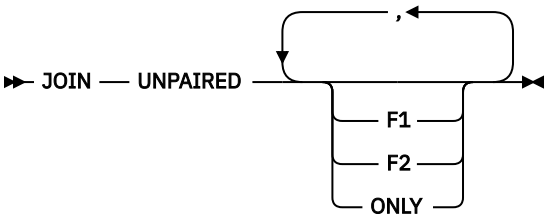
Only records without 'ABCD' in positions 5-8 will be processed from file F1. Only records with 'NO' somewhere in positions 1-20 will be processed from file F2.

Although the INCLUDE and OMIT operands are available on the JOINKEYS statement, it is recommended that you specify an INCLUDE or OMIT statement in JNF1CNTL or JNF2CNTL instead for ease of use.

For example, instead of the OMIT and INCLUDE operands in the JOINKEYS statements shown previously, you could specify:

```
//SYSIN DD *
  JOINKEYS FILE=F1,FIELDS=(35,8,A)
  JOINKEYS FILE=F2,FIELDS=(37,8,A)
  ...
//JNF1CNTL DD *
  OMIT COND=(5,4,CH,EQ,C'ABCD')
//JNF2CNTL DD *
  INCLUDE COND=(1,20,SS,EQ,C'NO')
```

JOIN statement



If you don't specify a JOIN statement for a JOINKEYS application, only paired records from F1 and F2 are kept and processed by the main task as the joined records. This is known as an inner join.

You can change which records are kept and processed by the main task as the joined records by specifying a JOIN statement. You must specify the UNPAIRED operand. The F1, F2 and ONLY operands

are optional. The JOIN operands you specify indicate the joined records to be kept and processed by the main task as follows:

UNPAIRED,F1,F2 or UNPAIRED
Unpaired records from F1 and F2 as well as paired records. This is known as a full outer join.

UNPAIRED,F1
Unpaired records from F1 as well as paired records. This is known as a left outer join.

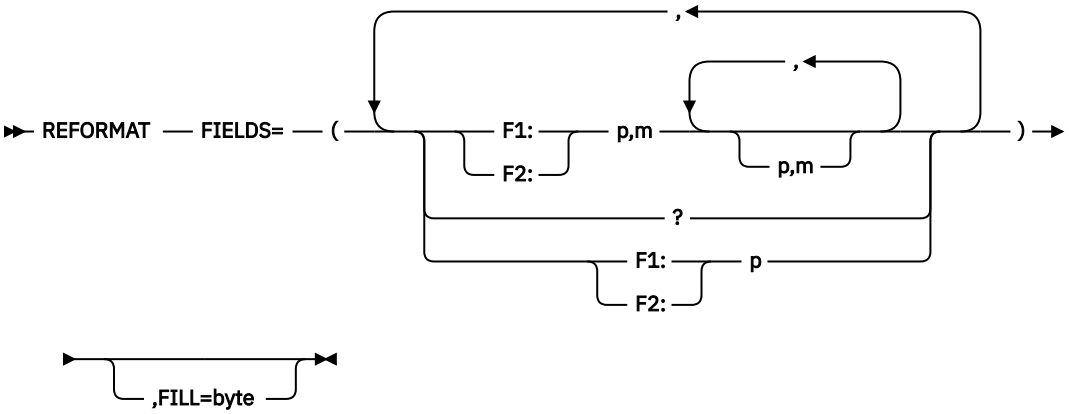
UNPAIRED,F2
Unpaired records from F2 as well as paired records. This is known as a right outer join.

UNPAIRED,F1,F2,ONLY or UNPAIRED,ONLY
Unpaired records from F1 and F2.

UNPAIRED,F1,ONLY
Unpaired records from F1.

UNPAIRED,F2,ONLY
Unpaired records from F2.

REFORMAT statement



A REFORMAT statement can always be used for a JOINKEYS application, and is required unless a JOIN statement with the ONLY operand is specified. The REFORMAT statement indicates the fields from the F1 file and/or the F2 file you want to include in the joined records, and the order in which you want the fields to appear. You can also include an indicator of where the key was found in the joined records ('B' for key found in F1 and F2, '1' for key found in F1 only, or '2' for key found in F2 only).

If the REFORMAT statement only defines position with length (p,m) fields, each joined record will be fixed-length (TYPE=F) with a LENGTH equal to the total length of all of the p,m fields. The maximum length for TYPE=F joined records is 32760 bytes. The F1 and F2 files can both be fixed-length, both be variable-length, or can be mixed fixed-length and variable-length.

If the REFORMAT statement defines a position without a length (p without m) field, each joined record will be variable-length (TYPE=V) with a LENGTH equal to the total length of all of the p,m fields plus the variable length of each p field (one for F1 and/or one for F2). The maximum length for TYPE=V joined records is 32767 bytes. The F1 and F2 files can both be variable-length or can be mixed fixed-length and variable-length. However:

- The first field must contain the RDW (1,n with n equal to or greater than 4) and must be from a variable-length file (F1 or F2).
- The position without length fields (one from the F1 file and/or one from the F2 file) must be the last fields specified and must be from a variable-length file (F1 and/or F2).

For joined records created from paired records, any F1 fields specified will be extracted from the F1 record and any F2 fields specified will be extracted from the F2 record.

For joined records created from unpaired F1 records, any F1 fields specified will be extracted from the F1 record and any F2 fields specified will be filled with the specified FILL character or blanks by default. However, if the F2 file is variable-length, any specified F2 RDW fields (1,n with n equal to or less than 4) will be filled with binary zeros.

For joined records created from unpaired F2 records, any F2 fields specified will be extracted from the F2 record and any F1 fields specified will be filled with the specified FILL character or blanks by default. However, if the F1 file is variable-length, any specified F1 RDW fields (1,n with n equal to or less than 4) will be filled with binary zeros.

If a JOIN statement with the ONLY operand is specified, the REFORMAT statement does not have to be specified. In this case, the layout of the joined records will depend on the specified JOIN statement operands as follows:

JOIN UNPAIRED,F1,ONLY
The joined records will be the original unpaired F1 records. If the F1 records are fixed-length, the joined records will be fixed-length. If the F1 records are variable-length, the joined records will be variable-length.

JOIN UNPAIRED,F2,ONLY
The joined records will be the original unpaired F2 records. If the F2 records are fixed-length, the joined records will be fixed-length. If the F2 records are variable-length, the joined records will be variable-length.

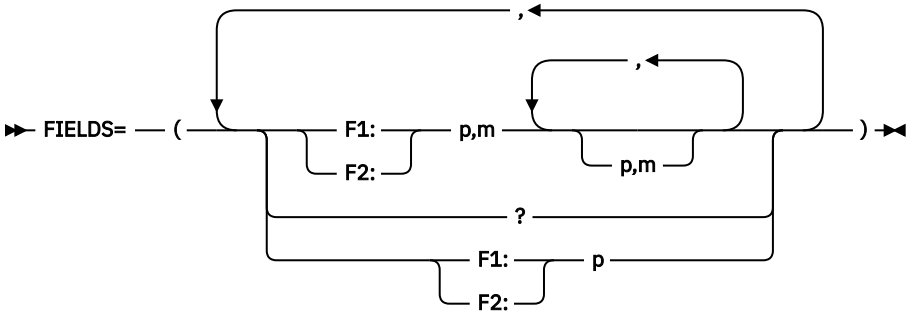
JOIN UNPAIRED,F1,F2,ONLY or JOIN UNPAIRED,ONLY
The joined records will be variable-length. If the F1 records are fixed-length, each unpaired F1 record will be variable-length with an RDW followed by the original F1 record. If the F1 records are variable-length, each unpaired F1 record will be the original F1 record. If the F2 records are fixed-length, each unpaired F2 record will be variable-length with an RDW followed by the original F2 record. If the F2 records are variable-length, each unpaired F2 record will be the original F2 record.

In all cases, the TYPE and LENGTH will be set as appropriate for the joined records.

For joined records with TYPE=F, the maximum LENGTH is 32760.

For joined records with TYPE=V, the maximum LENGTH is 32767. Note that the RECFM of the output file must be VS or VBS in order to allow output records greater than 32756.

FIELDS



Must be specified to indicate the starting position and length of each field from the F1 file and/or the F2 file to be included in the joined records, and optionally an indicator of where the key was found. The fields and indicator will be included in the joined records in the order in which they are specified.

F1:
indicates the following fields up to the next Fn: or end of the FIELDS operand are from the F1 file.

F2:
indicates the following fields up to the next Fn: or end of the FIELDS operand are from the F2 file.

p,m
specifies the starting position and length of a fixed field. p specifies the starting position of the field. The first data byte of a fixed-length record is in position 1. The first data byte of a variable-length record is in position 5 after the 4-byte RDW. p can be 1 to 32767. m specifies the

length of the field. m can be 1 to 32760. All fields must be completely contained within the first 32767 bytes of the record.

?

indicates a 1-byte indicator is to be included in each joined record. The indicator will be set to one of the following values in each paired or unpaired joined record, as appropriate:

- 'B' - the key was found in F1 and F2.
- '1' - the key was found in F1, but not in F2.
- '2' - the key was found in F2, but not in F1.

Only one ? can be specified in the FIELDS operand. If ? is not the last item, it must be followed by F1: or F2:.

For TYPE=F joined records, the indicator can appear anywhere in the record. For example, each of the following is valid:

```
* Put indicator in position 1 of each joined record.
  REFORMAT FIELDS=(?,F1:1,20,F2:5,8)

* Put indicator in position 21 of each joined record.
  REFORMAT FIELDS=(F1:1,20,?,F1:31,9)

* Put indicator in position 25 of each joined record.
  REFORMAT FIELDS=(F2:11,20,6,4,?)
```

For TYPE=V joined records, the indicator must appear in the fixed part of the record, that is, after the RDW and before the position without length fields. For example, the following is valid:

```
* Put indicator in position 5 of each joined record.
  REFORMAT FIELDS=(F1:1,4,?,F1:11)
```

p

p without m gives the starting position of a variable field. Only one p without m field can be specified for F1 and only one p without m field can be specified for F2.

If either or both p without m fields are specified, they must be the last fields in the FIELDS operand. In this case, the first field in the FIELDS operand must be from a variable-length file (F1 or F2) and must include the RDW (1,n where n is equal to or greater than 4).

Example with just p,m fields:

```
  REFORMAT FIELDS=(F1:27,5,1,8,F2:19,20,F1:1201,15)
```

Example with one p without m field:

```
  REFORMAT FIELDS=(F1:1,4,F2:6,25,92,2,F1:8,9,32)
```

Example with two p without m fields:

```
  REFORMAT FIELDS=(F2:1,9,21,3,F1:101,7,28,9,122,F2:26)
```

FILL

The FILL operand can be used to override DFSORT's default fill byte of a blank (X'40'). The fill byte is used in the following situations:

- A p,m (fixed) field is specified for a file (F1 or F2) with variable-length records, and the field extends beyond the end of a record. Each missing byte is replaced with the fill byte. For example, if a variable-length F1 record is 20 bytes long and the REFORMAT statement has:

  ```
    REFORMAT FIELDS=(F1:1,30,41),FILL=C'*'
  ```

  bytes 21-30 of the joined record are filled with asterisks.

- The REFORMAT statement has a p,m (fixed) field from the F1 file and a joined record is being created from an unpaired F2 record. For example if the following are specified:

```
JOIN UNPAIRED,F2
REFORMAT FIELDS=(F1:21,5,F2:1,10),FILL=X'00'
```

  and an unpaired F2 record is found, the joined record will have 5 binary zero bytes followed by bytes 1-10 from the F2 record.
- The REFORMAT statement has a p,m (fixed) field from the F2 file and a joined record is being created from an unpaired F1 record. For example if the following are specified:

```
JOIN UNPAIRED,F1,F2
REFORMAT FIELDS=(F1:21,5,F2:1,10),FILL=X'00'
```

  and an unpaired F1 record is found, the joined record will have bytes 21-25 of the F1 record followed by 10 binary zero bytes. (Since UNPAIRED,F1,F2 is specified, if an unpaired F2 record is found, the joined record will have 5 binary zero bytes followed by bytes 1-10 of the F2 record.)

byte
specifies the fill byte. Permissible values are C'x' and X'yy'.

C'x'
Character fill byte. x must be one EBCDIC character. If you want to use an apostrophe as the fill byte, you must specify it as C''''.

X'yy'
Hexadecimal fill byte. yy must be one pair of hexadecimal digits (00-FF).

JOINKEYS application notes

In the notes later in this section, "subtaskn" refers to subtask1 for the F1 file or subtask2 for the F2 file.

- Since a JOINKEYS application uses three tasks, it can require more storage than a regular DFSORT application. You may need to use REGION=0M for some JOINKEYS applications.
- DFSORT's normal syntax and continuation rules are used for the JOINKEYS, JOIN and REFORMAT statements. See Chapter 3, “Using DFSORT program control statements,” on page 77 for details.
- Control statements from DFSPARM, SYSIN (direct invocation) or SORTCNTL (program invocation) will be used for the main task, but not for the subtasks. Control statements from JNF1CNTL will be used for subtask1. Control statements from JNF2CNTL will be used for subtask2.
- For the main task, the normal options in effect for a DFSORT run will be used. The run-time options will be those from DFSPARM, SYSIN, SORTCNTL, EXEC PARM or the caller's parameter list, as appropriate. The installation options will be those for direct invocation or program invocation of DFSORT, as appropriate.
- For subtaskn, the options in effect for the subtask will be used with the exceptions noted later in this section. The run-time options will be those from JNFnCNTL. The installation options will be those for a calling program (INV, TSOINV, TD1-TD4).
- TYPE=F processing is used for a VSAM F1 or F2 file by default. TYPE=V can be specified in JNFnCNTL, or on the JOINKEYS statement for Fn, to override the default of TYPE=F.
- For subtaskn, if an INCLUDE or OMIT statement or a STOPAFT or TYPE option is specified in JNFnCNTL, it will override the corresponding option in a JOINKEYS statement for Fn. For ease of use, it is recommended that you use an INCLUDE or OMIT statement, and a STOPAFT or TYPE option, in JNFnCNTL rather than an INCLUDE, OMIT, STOPAFT or TYPE operand in a JOINKEYS statement.
- For the main task, override of statements and options is done in the normal way, that is, DFSPARM overrides SYSIN for direct invocation of DFSORT, and DFSPARM overrides SORTCNTL overrides the caller's parameter list for program invocation of DFSORT. See Chapter 3, “Using DFSORT program control statements,” on page 77 for details.

- The following options will be used for subtaskn and cannot be overridden: LIST, MSGPRT=ALL, NOABEND, ESTAE, NOCHECK, EQUALS and RESINV=0. DYNALLOC will be used automatically for subtaskn, but can be overridden by a DYNALLOC option in JNFnCNTL.
- If an INCLUDE or OMIT statement is specified in JNFnCNTL, DFSORT will use the following options in effect for subtaskn processing: ALTSEQ, CHALT or NOCHALT, SZERO or NOSZERO and VLSCMP or NOVLSCMP. These options can be specified in JNFnCNTL. The subtaskn LOCALE installation option in effect will also be used if appropriate.
- If an INCLUDE or OMIT operand is specified on the JOINKEYS statement for Fn, DFSORT will use these main task options in effect for subtaskn processing: ALTSEQ, CHALT or NOCHALT, SZERO or NOSZERO and VLSCMP or NOVLSCMP. If these options are specified in JNFnCNTL, they will be ignored for subtaskn processing. LOCALE will not be used.
- For subtaskn, the following statements are not allowed in JNFnCNTL: JOINKEYS, JOIN, MERGE, OUTFIL, OUTREC, REFORMAT and SORT.
- For the main task, a MERGE statement other than MERGE FIELDS=COPY is not allowed in any source.
- If appropriate, options such as FILSZ, DYNALLOC, AVGRLEN, and so on can be specified in JNFnCNTL for subtaskn, or in DFSPARM for the main task.
- If tape data sets are used for the F1 and F2 files, they must be on different drives so DFSORT can read them in parallel.

JOINKEYS application examples

Example 1 - Paired F1/F2 records without duplicates

```
//JKE1  EXEC  PGM=SORT
//SYSOUT    DD  SYSOUT=*
//SORTJNF1 DD *
Roses          03  Red
Daisies        06  Orange
Roses          04  Pink
Daisies        02  Yellow
Roses          06  Yellow
Daisies        12  Lilac
Roses          09  Blue
/*
//SORTJNF2 DD *
Red      Lilies          InStock
Red      Roses           InStock
Orange   Daisies         SoldOut
Pink     Roses           SoldOut
Yellow   Daisies         InStock
Yellow   Roses           Ordered
Lilac    Daisies         SoldOut
White    Daisies         InStock
/*
//SORTOUT DD SYSOUT=*
//SYSIN    DD    *
* Control statements for JOINKEYS application
  JOINKEYS FILE=F1,FIELDS=(1,15,A,20,8,A)
  JOINKEYS FILE=F2,FIELDS=(10,15,A,1,8,A)
  REFORMAT FIELDS=(F1:20,8,1,15,F2:26,10,F1:16,2)
* Control statements for main task (joined records)
  OPTION COPY
  OUTFIL REMOVECC,
    HEADER2=(1:'Color',11:'Flower',26:'Status',36:'Per Pot',/,
             1:7'-',11:14'-',26:7'-',36:7'-'),
    BUILD=(1:1,8,11:9,15,26:24,10,
      36:34,2,ZD,M10,LENGTH=7)
/*
```

This example illustrates how you can join paired records from two files using multiple keys. In this case, neither file has duplicates. The paired records are the records in F1 and F2 with matching keys (for example, key1=Roses and key2=Red).

Input file1 (F1) has RECFM=FB and LRECL=80. It contains the records shown for SORTJNF1 in the JCL shown previously in this section. Input file2 (F2) has RECFM=FB and LRECL=80. It contains the records shown for SORTJNF2 in the JCL shown previously in this section.

The output file will have RECFM=FB and LRECL=42. It will contain the paired records from the two files reformatted as follows:

```
Color     Flower         Status    Per Pot
-------   ------------- -------   -------
Lilac     Daisies        SoldOut        12
Orange    Daisies        SoldOut         6
Yellow    Daisies        InStock         2
Pink      Roses          SoldOut         4
Red       Roses          InStock         3
Yellow    Roses          Ordered         6
```

The first JOINKEYS statement defines the ddname and keys for the F1 file. FILE=F1 tells DFSORT that the ddname for the F1 file is SORTJNF1. FIELDS=(1,15,A,20,8,A) tells DFSORT that the first binary key is in positions 1-15 ascending and the second binary key is in positions 20-27 ascending. Since SORTED is not specified, DFSORT will sort the SORTJNF1 records by the specified binary keys.

The second JOINKEYS statement defines the ddname and keys for the F2 file. FILE=F2 tells DFSORT that the ddname for the F2 file is SORTJNF2. FIELDS=(10,15,A,1,8,A) tells DFSORT that the first binary key is in positions 10-24 ascending and the second binary key is in positions 1-8 ascending. Since SORTED is not specified, DFSORT will sort the SORTJNF2 records by the specified binary keys.

Note that corresponding keys for the two files match in length and order.

The REFORMAT statement defines the fields to be extracted for the joined records in the order in which they are to appear. FIELDS=(F1:20,8,1,15,F2:26,10,F1:16,2) tells DFSORT to create the joined records as follows:

```
Joined Record Positions     Extracted from
-----------------------     ------------------
1-8                         F1 positions 20-27
9-23                        F1 positions 1-15
24-33                       F2 positions 26-35
34-35                       F1 positions 16-17
```

Since there is no JOIN statement, only paired records are joined by default.

The OPTION COPY statement tells DFSORT to copy the joined records. The OUTFIL statement tells DFSORT to reformat the joined records, display a header at the top of each page and remove the carriage control characters. Note that the BUILD operand of the OUTFIL statement must reference the positions of fields in the joined records.

Conceptually, JOINKEYS application processing proceeds as follows:

- Subtask1 sorts the SORTJNF1 (F1 file) records as directed by its JOINKEYS statement. As a result, it passes the following records to the main task:

  ```
  Daisies        12  Lilac
  Daisies        06  Orange
  Daisies        02  Yellow
  Roses          09  Blue
  Roses          04  Pink
  Roses          03  Red
  Roses          06  Yellow
  ```

- Subtask2 sorts the SORTJNF2 (F2 file) records as directed by its JOINKEYS statement. As a result, it passes the following records to the main task:

  ```
  Lilac    Daisies         SoldOut
  Orange   Daisies         SoldOut
  White    Daisies         InStock
  Yellow   Daisies         InStock
  Red      Lilies          InStock
  Pink     Roses           SoldOut
  Red      Roses           InStock
  Yellow   Roses           Ordered
  ```

- The main task joins the records passed from subtask1 and subtask2 as directed by the specified JOINKEYS and REFORMAT statements, resulting in the following joined records:

```
Lilac   Daisies        SoldOut   12
Orange  Daisies        SoldOut   06
Yellow  Daisies        InStock   02
Pink    Roses          SoldOut   04
Red     Roses          InStock   03
Yellow  Roses          Ordered   06
```

- Finally, the main task copies and reformats the joined records according to the OUTFIL statement, and writes the resulting records to SORTOUT. Thus, SORTOUT contains these records:

```
Color     Flower         Status    Per Pot
-------   -------------- -------   -------
Lilac     Daisies        SoldOut        12
Orange    Daisies        SoldOut         6
Yellow    Daisies        InStock         2
Pink      Roses          SoldOut         4
Red       Roses          InStock         3
Yellow    Roses          Ordered         6
```

Example 2 - Paired F1/F2 records with duplicates (cartesian)

```
//JKE2  EXEC  PGM=SORT
//SYSOUT    DD  SYSOUT=*
//VBIN DD DSN=MY.VBFILE,DISP=SHR
//FBIN DD DSN=MY.FBFILE,DISP=SHR
//SORTOUT DD DSN=MY.FB.OUTPUT,DISP=(NEW,CATLG,DELETE),
// SPACE=(CYL,(5,5)),UNIT=SYSDA
//SYSIN    DD    *
* Control statements for JOINKEYS application
  JOINKEYS F1=VBIN,FIELDS=(18,16,A),SORTED
  JOINKEYS F2=FBIN,FIELDS=(1,16,A)
  REFORMAT FIELDS=(F2:22,12,F1:5,12,F2:1,16)
* Control statements for main task (joined records)
  OPTION EQUALS
  SORT FIELDS=(13,12,CH,A)
/*
```

This example illustrates how you can join paired records from two files, both of which have duplicate records. The result will be a cartesian join. The paired records are the records in F1 and F2 with matching keys (for example, key=Cats).

Input file1 has RECFM=VB and LRECL=50. It contains the following records:

```
Len|Data
 40|Eliot        Cats            Musical
 40|Lloyd-Webber Cats            Musical
 48|Hart         Pal Joey        Musical, Comedy
 48|Rodgers      Pal Joey        Musical, Comedy
 47|Hammerstein  South Pacific   Musical, Drama
 47|Rodgers      South Pacific   Musical, Drama
```

The output file will have RECFM=FB and LRECL=40. It will contain the paired cartesian product of the two files sorted as follows:

```
Start: 1982 Eliot       Cats
End:   2000 Eliot       Cats
Start: 1949 Hammerstein South Pacific
End:   1954 Hammerstein South Pacific
Start: 1940 Hart        Pal Joey
End:   1941 Hart        Pal Joey
Start: 1982 Lloyd-WebberCats
End:   2000 Lloyd-WebberCats
Start: 1940 Rodgers     Pal Joey
End:   1941 Rodgers     Pal Joey
Start: 1949 Rodgers     South Pacific
End:   1954 Rodgers     South Pacific
```

The first JOINKEYS statement defines the ddname and key for the F1 file. F1=VBIN tells DFSORT that the ddname for the F1 file is VBIN. FIELDS=(18,16,A) tells DFSORT that the key is in positions 18-33

ascending. Note that since VBIN is a VB file, the starting position of its key must take the RDW in positions 1-4 into account. Since SORTED is specified, indicating that the records are already in order by the specified binary key, DFSORT will copy the VBIN records.

The second JOINKEYS statement defines the ddname and binary key for the F2 file. F2=FBIN tells DFSORT that the ddname for the F2 file is FBIN. FIELDS=(1,16,A) tells DFSORT that the binary key is in positions 1-16 ascending. Since SORTED is not specified, DFSORT will sort the FBIN records by the specified binary key.

The REFORMAT statement defines the fields to be extracted for the joined records in the order in which they are to appear. FIELDS=(F2:22,12,F1:5,12,F2:1,16) tells DFSORT to create the joined records as follows:

```
Joined Record Positions     Extracted from
-----------------------     ------------------
1-12                        F2 positions 22-33
13-24                       F1 positions 5-16
25-40                       F2 positions 1-16
```

Note that since VBIN (F1) is a VB file, the starting position of its REFORMAT field must take the RDW in positions 1-4 into account.

Since there is no JOIN statement, only paired records are joined by default. Since there are duplicates in each input file, a cartesian join is performed.

The SORT FIELDS=(13,12,CH,A) statement tells DFSORT to sort the joined records by a different key than the one used for the join of F1 and F2 records. Note that the FIELDS operand of the SORT statement must reference the positions of fields in the joined records.

Conceptually, JOINKEYS application processing proceeds as follows:

- Subtask1 copies the VBIN (F1 file) records as directed by its JOINKEYS statement. As a result, it passes the following records to the main task:

  ```
  Len|Data
   40|Eliot        Cats            Musical
   40|Lloyd-Webber Cats            Musical
   48|Hart         Pal Joey        Musical, Comedy
   48|Rodgers      Pal Joey        Musical, Comedy
   47|Hammerstein  South Pacific   Musical, Drama
   47|Rodgers      South Pacific   Musical, Drama
  ```

- Subtask2 sorts the FBIN (F2 file) records as directed by its JOINKEYS statement. As a result, it passes the following records to the main task:

  ```
  Cats                Start: 1982  50
  Cats                End:   2000
  Pal Joey            Start: 1940  22
  Pal Joey            End:   1941
  South Pacific       Start: 1949  13
  South Pacific       End:   1954
  ```

- The main task joins the records passed from subtask1 and subtask2 as directed by the specified JOINKEYS and REFORMAT statements, resulting in the following joined records:

  ```
  Start: 1982 Eliot       Cats
  End:   2000 Eliot       Cats
  Start: 1982 Lloyd-WebberCats
  End:   2000 Lloyd-WebberCats
  Start: 1940 Hart        Pal Joey
  End:   1941 Hart        Pal Joey
  Start: 1940 Rodgers     Pal Joey
  End:   1941 Rodgers     Pal Joey
  Start: 1949 Hammerstein South Pacific
  End:   1954 Hammerstein South Pacific
  Start: 1949 Rodgers     South Pacific
  End:   1954 Rodgers     South Pacific
  ```

- Finally, the main task sorts the joined records according to the SORT statement, and writes the resulting records to SORTOUT. Thus, SORTOUT contains these records:

```
Start: 1982 Eliot       Cats
End:   2000 Eliot       Cats
Start: 1949 Hammerstein South Pacific
End:   1954 Hammerstein South Pacific
Start: 1940 Hart        Pal Joey
End:   1941 Hart        Pal Joey
Start: 1982 Lloyd-WebberCats
End:   2000 Lloyd-WebberCats
Start: 1940 Rodgers     Pal Joey
End:   1941 Rodgers     Pal Joey
Start: 1949 Rodgers     South Pacific
End:   1954 Rodgers     South Pacific
```

Example 3 - Paired F1 records

```
//JKE3  EXEC  PGM=SORT
//SYSOUT    DD  SYSOUT=*
//MASTER DD DSN=MASTER.FILE,DISP=SHR
//PULL DD DSN=PULL.FILE,DISP=SHR
//SORTOUT DD SYSOUT=*
//JNF1CNTL DD *
* Control statement for subtask1 (F1)
  INREC PARSE=(%01=(ENDBEFR=C',',FIXLEN=15),
               %=(ENDBEFR=C','),
               %02=(ENDBEFR=C',',FIXLEN=10)),
  OVERLAY=(36:%01,52:%02)
/*
//DFSPARM   DD    *
* Control statements for JOINKEYS application
  JOINKEYS F1=MASTER,FIELDS=(36,15,A,52,10,A)
  JOINKEYS F2=PULL,FIELDS=(12,15,A,1,10,A)
  REFORMAT FIELDS=(F1:1,35)
* Control statement for main task
  OPTION COPY
/*
```

This example illustrates how you can select only paired records from one of two files. In this case, we will select the F1 records that have a match in F2 on the specified keys (for example, key1=Development and key2=Master). The F1 records are in comma delimited form so we will parse them to get the binary keys we need.

Input file1 (F1) has RECFM=FB and LRECL=35. It contains the following records:

```
Marketing,William,Master
Development,John,Bachelor
Manufacturing,Louis,Master
Development,Carol,Master
Research,Angela,Master
Research,Anne,Doctorate
Development,Sara,Doctorate
Marketing,Joseph,Master
Manufacturing,Donna,Bachelor
Development,Susan,Master
Manufacturing,Nick,Master
Research,Howard,Doctorate
```

Input file2 (F2) has RECFM=FB and LRECL=30. It contains the following records:

```
Master     Development
Master     Manufacturing
Bachelor   Development
Doctorate  Research
```

The output file will have RECFM=FB and LRECL=35. It will contain the paired F1 records, that is, the records from F1 that have a match in F2 for the specified keys (the first and third fields):

```
Development,John,Bachelor
Development,Carol,Master
Development,Susan,Master
Manufacturing,Louis,Master
Manufacturing,Nick,Master
Research,Anne,Doctorate
Research,Howard,Doctorate
```

The first JOINKEYS statement defines the ddname and keys for the F1 file. F1=MASTER tells DFSORT that the ddname for the F1 file is MASTER.

The control statements in JNF1CNTL will be used to process the F1 file before it is sorted. The input records are in comma delimited form, so to use the first and third fields as binary keys, while preserving the original data, we use an INREC statement to parse out the fields we want and add them to the end of the record. The parsed first key will be in positions 36-50 and the parsed second key will be in positions 52-61. For example, the first reformatted record will look like this:

```
Marketing,William,Master           Marketing       Master
```

FIELDS=(36,15,A,52,10,A) in the JOINKEYS statement (referring to the reformatted INREC positions) tells DFSORT that the first key is in positions 36-50 ascending and the second key is in positions 52-61 ascending.

The second JOINKEYS statement defines the ddname and keys for the F2 file. F2=PULL tells DFSORT that the ddname for the F2 file is PULL.

FIELDS=(12,15,A,1,10,A) in the JOINKEYS statement tells DFSORT that the first key is in positions 12-26 ascending and the second key is in positions 1-10 ascending.

The REFORMAT statement defines the fields to be extracted for the joined records in the order in which they are to appear. FIELDS=(F1:1,35) tells DFSORT to create the joined records from the original F1 input records (positions 1-35 of the F1 records). Since we only needed the added parsed fields for sorting, we don't need to include them in the joined records. Of course, if we wanted the main task to "see" the parsed fields in the joined records, we could include the parsed fields in the REFORMAT FIELDS operand.

Since there is no JOIN statement, only paired records are joined by default.

The OPTION COPY statement tells DFSORT to copy the joined records.

Conceptually, JOINKEYS application processing proceeds as follows:

- Subtask1 performs INREC processing for the MASTER (F1 file) records as directed by the control statement in JNF1CNTL and sorts the resulting records as directed by its JOINKEYS statement. As a result, it passes the following records to the main task:

  ```
  Development,John,Bachelor          Development     Bachelor
  Development,Sara,Doctorate         Development     Doctorate
  Development,Carol,Master           Development     Master
  Development,Susan,Master           Development     Master
  Manufacturing,Donna,Bachelor       Manufacturing   Bachelor
  Manufacturing,Louis,Master         Manufacturing   Master
  Manufacturing,Nick,Master          Manufacturing   Master
  Marketing,William,Master           Marketing       Master
  Marketing,Joseph,Master            Marketing       Master
  Research,Howard,Doctorate          Research        Doctorate
  Research,Anne,Doctorate            Research        Doctorate
  Research,Angela,Master             Research        Master
  ```

- Subtask2 sorts the PULL (F2 file) records as directed by its JOINKEYS statement. As a result, it passes the following records to the main task:

  ```
  Bachelor   Development
  Master     Development
  Master     Manufacturing
  Doctorate  Research
  ```

- The main task joins the records passed from subtask1 and subtask2 as directed by the specified JOINKEYS and REFORMAT statements, resulting in the following joined records:

  ```
  Development,John,Bachelor
  Development,Carol,Master
  Development,Susan,Master
  Manufacturing,Louis,Master
  Manufacturing,Nick,Master
  Research,Anne,Doctorate
  Research,Howard,Doctorate
  ```

- Finally, the main task copies the joined records to SORTOUT. Thus, SORTOUT contains these records:

```
Development,John,Bachelor
Development,Carol,Master
Development,Susan,Master
Manufacturing,Louis,Master
Manufacturing,Nick,Master
Research,Anne,Doctorate
Research,Howard,Doctorate
```

Example 4 - Unpaired F2 records

```
//JKE4  EXEC  PGM=SORT
//SYSOUT    DD  SYSOUT=*
//IN1 DD DSN=FILE1.IN,DISP=SHR
//IN2 DD DSN=FILE2.IN,DISP=SHR
//SORTOUT DD DSN=FILE3.OUT,DISP=OLD
//JNF1CNTL DD *
* Control statements for subtask1 (F1)
  OMIT COND=(10,5,UFF,EQ,99999)
  INREC BUILD=(1,8,9:10,5,UFF,TO=ZD,LENGTH=5)
/*
//JNF2CNTL DD *
* Control statements for subtask2 (F2)
  OMIT COND=(14,5,UFF,EQ,99999)
  INREC BUILD=(1,4,5:14,5,UFF,TO=ZD,LENGTH=5,10:5)
/*
//SYSIN    DD    *
* Control statements for JOINKEYS application
  JOINKEYS F1=IN1,FIELDS=(1,8,A,9,5,D)
  JOINKEYS F2=IN2,FIELDS=(10,8,A,5,5,D)
  JOIN UNPAIRED,F2,ONLY
  REFORMAT FIELDS=(F2:1,4,10)
* Control statement for main task
  OPTION COPY
/*
```

This example illustrates how you can select only unpaired records from one of two files. In this case, we will select the F2 records that do not have a match in F1 on the specified keys (for example, key1=Molly and key2=2100). We will also omit certain records from each input file and handle unnormalized keys.

Input file1 has RECFM=FB and LRECL=15. It contains the following records:

```
Molly    145
Molly    99999
Molly    2143
Jasmine  1292
Jasmine  5
Jasmine  28
Jasmine  99999
```

Input file2 has RECFM=VB and LRECL=35. It contains the following records:

```
Len|Data
 30|Molly    145      Thursday
 31|Molly    2100     Wednesday
 28|Molly    18       Sunday
 28|Jasmine  99999    Monday
 28|Jasmine  5        Sunday
 30|Jasmine  28       Saturday
 29|Jasmine  103      Tuesday
 31|Jasmine  99999    Wednesday
```

The output file will have RECFM=VB and LRECL=35. F1 records with 99999 in positions 10-14 and F2 records with 99999 in positions 14-18 will be removed before join processing. The output file will contain unpaired F2 records (that is, records from F2 that do not have a match in F1 for the specified keys) as follows:

```
Len|Data
 29|Jasmine  103      Tuesday
 31|Molly    2100     Wednesday
 28|Molly    18       Sunday
```

The first JOINKEYS statement defines the ddname and keys for the F1 file. F1=IN1 tells DFSORT that the ddname for the F1 file is IN1.

The control statements in JNF1CNTL will be used to process the F1 file before it is sorted. The OMIT statement removes records that have 99999 in positions 10-14. We want to use the numeric field as our key. However, the numeric field is unnormalized since it is left aligned instead of right aligned, so sorting it as a binary key will not work. To handle this, we use the INREC statement to reformat the numeric field as ZD values in positions 9-13 (positive ZD values with the same sign can be sorted as binary). For example, the first reformatted FB record will look like this:

```
Molly   00145
```

Since we don't need the F1 records for output, we don't need to keep the original left aligned numeric value.

FIELDS=(1,8,A,9,5,D) in the JOINKEYS statement (referring to the reformatted INREC positions) tells DFSORT that the first key is in positions 1-8 ascending and the second key is in positions 9-13 descending.

The second JOINKEYS statement defines the ddname and keys for the F2 file. F2=IN2 tells DFSORT that the ddname for the F2 file is IN2.

The control statements in JNF2CNTL will be used to process the F2 file before it is sorted. The OMIT statement removes records that have 99999 in positions 14-18. Again, we need a ZD version of the left aligned numeric value to use for the binary key. But in this case, since we want the original F2 records for output, we need to keep the original numeric value as well. Using the INREC statement, we add the ZD value at positions 5-9 between the RDW and the first data field. That shifts the original data to start at position 10. For example, the first reformatted VB record will look like this:

```
Len|Data
 35|00145Molly    145      Thursday
```

In this case, since the input is a VB file, we specify the RDW (1,4), then the converted field, and then the rest of the record (5 without a length) in the INREC statement.

FIELDS=(10,8,A,5,5,D) in the JOINKEYS statement (referring to the reformatted INREC positions) tells DFSORT that the first key is in positions 10-17 ascending and the second key is in positions 5-9 descending.

Note that since IN2 is a VB file, all of its starting positions must take the RDW in positions 1-4 into account.

The JOIN statement tells DFSORT that the joined records should be the F2 records that do not have a match in F1 on the specified keys.

The REFORMAT statement defines the fields to be extracted for the joined records in the order in which they are to appear. We need the RDW (1,4) and the original data which starts in position 10 of the reformatted F2 records. So we use FIELDS=(F2:1,4,10). Since the last field (10) is a position without a length, it tells DFSORT to create VB records. The joined records are created as follows from the reformatted F2 records:

```
Joined Record Positions     Extracted from
-----------------------     --------------------------------
1-4                         RDW (not extracted)
5 to end                    Reformatted F2 position 10 to end
```

Conceptually, JOINKEYS application processing proceeds as follows:

- Subtask1 performs OMIT and INREC processing for the IN1 (F1 file) records as directed by the control statements in JNF1CNTL and sorts the resulting records as directed by its JOINKEYS statement. As a result, it passes the following records to the main task:

  ```
  Jasmine 01292
  Jasmine 00028
  Jasmine 00005
  ```

```
Molly   02143
Molly   00145
```

- Subtask2 performs OMIT and INREC processing for the IN2 (F2 file) records as directed by the control statements in JNF2CNTL and sorts the resulting records as directed by its JOINKEYS statement. As a result, it passes the following records to the main task:

```
Len|Data
 34|00103Jasmine  103      Tuesday
 35|00028Jasmine  28       Saturday
 33|00005Jasmine  5        Sunday
 36|02100Molly    2100     Wednesday
 35|00145Molly    145      Thursday
 33|00018Molly    18       Sunday
```

- The main task joins the records passed from subtask1 and subtask2 as directed by the specified JOINKEYS, JOIN and REFORMAT statements, resulting in the following joined records (unmatched F2 records):

```
Len|Data
 29|Jasmine  103      Tuesday
 31|Molly    2100     Wednesday
 28|Molly    18       Sunday
```

- Finally, the main task copies the joined records, and writes them to SORTOUT. Thus, SORTOUT contains these records:

```
Len|Data
 29|Jasmine  103      Tuesday
 31|Molly    2100     Wednesday
 28|Molly    18       Sunday
```

Example 5 - Paired and unpaired F1/F2 records (indicator method)

```
//JKE5  EXEC  PGM=SORT
//SYSOUT    DD  SYSOUT=*
//SORTJNF1 DD DSN=FIRST.FILE,DISP=SHR
//SORTJNF2 DD DSN=SECOND.FILE,DISP=SHR
//F1ONLY DD SYSOUT=*
//F2ONLY DD SYSOUT=*
//BOTH DD SYSOUT=*
//SYSIN    DD    *
* Control statements for JOINKEYS application
  JOINKEYS FILE=F1,FIELDS=(1,10,A),SORTED,NOSEQCK
  JOINKEYS FILE=F2,FIELDS=(7,10,A),SORTED,NOSEQCK
  JOIN UNPAIRED,F1,F2
  REFORMAT FIELDS=(F1:1,14,F2:1,20,?)
* Control statements for main task (joined records)
  OPTION COPY
  OUTFIL FNAMES=F1ONLY,INCLUDE=(35,1,CH,EQ,C'1'),
    BUILD=(1,14)
  OUTFIL FNAMES=F2ONLY,INCLUDE=(35,1,CH,EQ,C'2'),
    BUILD=(15,20)
  OUTFIL FNAMES=BOTH,INCLUDE=(35,1,CH,EQ,C'B'),
    BUILD=(1,14,/,15,20)
/*
```

This example illustrates how you can create three output files; with paired F1/F2 records, unpaired F1 records and unpaired F2 records. In this case:

- F1 records which do not have a match in F2 on the specified keys (for example, key=David) will be written to the F1ONLY output file.
- F2 records which do not have a match in F1 on the specified keys (for example, key=Karen) will be written to the F2ONLY output file.
- F1 and F2 records which have a match in F1 and F2 on the specified keys (for example, key=Carrie) will be written to the BOTH output file.

Input file1 has RECFM=FB and LRECL=14. It contains the following records:

```
Carrie    F101
David     F102
Frank     F103
Holly     F104
Vicky     F105
```

Input file2 has RECFM=FB and LRECL=20. It contains the following records:

```
No    Carrie    F201
Yes   Holly     F202
Yes   Karen     F203
No    Sri Hari  F204
Yes   Vicky     F205
```

The F1ONLY file will have RECFM=FB and LRECL=14. It will contain the unpaired F1 records as follows:

```
David     F102
Frank     F103
```

The F2ONLY file will have RECFM=FB and LRECL=20. It will contain the unpaired F2 records as follows:

```
Yes   Karen     F203
No    Sri Hari  F204
```

The BOTH file will have RECFM=FB and LRECL=20. It will contain the paired F1 and F2 records as follows:

```
Carrie    F101
No    Carrie    F201
Holly     F104
Yes   Holly     F202
Vicky     F105
Yes   Vicky     F205
```

The first JOINKEYS statement defines the ddname and key for the F1 file. FILE=F1 tells DFSORT that the ddname for the F1 file is SORTJNF1. FIELDS=(1,10,A) tells DFSORT that the key is in positions 1-10 ascending. Since SORTED is specified, indicating that the records are already in order by the specified binary key, DFSORT will copy the SORTJNF1 records. Since NOSEQCK is specified, DFSORT will not check that the records are in order by the key. (Only use NOSEQCK if you know for sure that the records are in order by the key.)

The second JOINKEYS statement defines the ddname and key for the F2 file. FILE=F2 tells DFSORT that the ddname for the F2 file is SORTJNF2. FIELDS=(7,10,A) tells DFSORT that the key is in positions 7-16 ascending. Since SORTED is specified, indicating that the records are already in order by the specified binary key, DFSORT will copy the SORTJNF2 records. Since NOSEQCK is specified, DFSORT will not check that the records are in order by the key. (Only use NOSEQCK if you know for sure that the records are in order by the key.)

The JOIN statement tells DFSORT that the joined records should include the unpaired F1 and F2 records as well as the paired F1/F2 records.

The REFORMAT statement defines the fields to be extracted for the joined records in the order in which they are to appear, and includes an indicator in the last position that will be set to '1' if the key is found only in the F1 file, '2' if the key is found only in the F2 file, or 'B' if the key is found in the F1 file and in the F2 file. FIELDS=(F1:1,14,F2:1,20,?) tells DFSORT to create the joined records as follows:

```
Joined Record Positions   Extracted from
-----------------------   ----------------
1-14                      F1 positions 1-14
15-34                     F2 positions 1-20
35                        Indicator of where key was found
```

The OPTION COPY statement tells DFSORT to copy the joined records. The OUTFIL statements use the indicator in position 35 to determine where to find the F1 or F2 fields in the joined records and where to write the fields (F1ONLY, F2ONLY or BOTH).

Conceptually, JOINKEYS application processing proceeds as follows:

- Subtask1 copies the SORTJNF1 (F1) records as directed by the JOINKEYS statement. As a result, it copies the unchanged SORTJNF1 records to the main task.
- Subtask2 copies the SORTJNF2 (F2) records as directed by the JOINKEYS statement. As a result, it copies the unchanged SORTJNF2 records to the main task.
- The main task joins the records passed from subtask1 and subtask2 as directed by the specified JOINKEYS, JOIN and REFORMAT statements, resulting in the following joined records (paired and unpaired):

```
Carrie    F101No    Carrie    F201B
David     F102                  1
Frank     F103                  1
Holly     F104Yes   Holly     F202B
              Yes   Karen     F2032
              No    Sri Hari  F2042
Vicky     F105Yes   Vicky     F205B
```

  For F1 records without a match in F2 (for example, the F102 record), the indicator in position 35 has a '1'. For F2 records without a match in F1 (for example, the F203 record), the indicator in position 35 has a '2'. For F1 records with a match in F2 (for example, the F101 and F201 records), the indicator in position 35 has a 'B'.
- The first OUTFIL statement finds records with a '1' in position 35. These are the F1 records without a match in F2. The F1 field is in positions 1-14 of the joined record, so those positions are written to the F1ONLY file. Thus, F1ONLY contains these records:

```
David     F102
Frank     F103
```

- The second OUTFIL statement finds records with a '2' in position 35. These are the F2 records without a match in F1. The F2 field is in positions 15-34 of the joined record, so those positions are written to the F2ONLY file. Thus, F2ONLY contains these records:

```
Yes   Karen     F203
No    Sri Hari  F204
```

- The third OUTFIL statement finds records with 'B' in position 35. These are the F1 and F2 records with a match. The F1 field is in positions 1-14 of the joined record and the F2 field is in positions 15-34 of the joined record, so each joined record is split into those two records and written to the BOTH file. The shorter F1 record is padded with blanks on the right to the length of the F2 record. Thus, BOTH contains these records:

```
Carrie    F101
No    Carrie    F201
Holly     F104
Yes   Holly     F202
Vicky     F105
Yes   Vicky     F205
```

Note: If you only want one record per key in BOTH, you can have the BUILD for FNAMES=BOTH specify the positions for just that record. For example, BUILD=(1,14) for the F1 records or BUILD=(15,20) for the F2 records.

Example 6 - Paired and unpaired F1/F2 records (FILL method)

```
//JKE6  EXEC  PGM=SORT
//SYSOUT    DD  SYSOUT=*
//SORTJNF1 DD DSN=FIRST.FILE,DISP=SHR
//SORTJNF2 DD DSN=SECOND.FILE,DISP=SHR
//F1ONLY DD SYSOUT=*
//F2ONLY DD SYSOUT=*
//BOTH DD SYSOUT=*
//SYSIN    DD    *
* Control statements for JOINKEYS application
  JOINKEYS FILE=F1,FIELDS=(1,10,A),SORTED,NOSEQCK
  JOINKEYS FILE=F2,FIELDS=(7,10,A),SORTED,NOSEQCK
  JOIN UNPAIRED,F1,F2
  REFORMAT FIELDS=(F1:1,14,F2:1,20),FILL=C'$'
```

```
* Control statements for main task (joined records)
  OPTION COPY
  OUTFIL FNAMES=F1ONLY,INCLUDE=(15,1,CH,EQ,C'$'),
    BUILD=(1,14)
  OUTFIL FNAMES=F2ONLY,INCLUDE=(1,1,CH,EQ,C'$'),
    BUILD=(15,20)
  OUTFIL FNAMES=BOTH,INCLUDE=(15,1,CH,NE,C'$',AND,1,1,CH,NE,C'$'),
    BUILD=(1,14,/,15,20)
/*
```

This example illustrates an alternate way to create three output files; with paired F1/F2 records, unpaired F1 records and unpaired F2 records. It produces the same output as Example 5. Whereas Example 5 uses an indicator in one position to determine where the key was found, Example 6 uses a fill character in different positions to determine where the key was found. Another drawback of the Example 6 method is that you must use a FILL character that does not appear in either input record. The Explanation for Example 6 is the same as for Example 5 up to the REFORMAT statement and then it differs as follows:

The REFORMAT statement defines the fields to be extracted for the joined records in the order in which they are to appear. FIELDS=(F1:1,14,F2:1,20) tells DFSORT to create the joined records as follows:

```
Joined Record Positions   Extracted from
-----------------------   ----------------
1-14                      F1 positions 1-14
15-34                     F2 positions 1-20
```

FILL=C'$' tells DFSORT to use $ as the fill character for the F1 field in an unpaired F2 record and for the F2 field in an unpaired F1 record. We use the FILL character to identify the unpaired records from each file; when used for this purpose, the default of FILL=X'40' is usually not a good choice. You must select a FILL character that will not appear in either input file.

The OPTION COPY statement tells DFSORT to copy the joined records. The OUTFIL statements use the presence or absence of the $ fill character in certain positions to determine where to find the F1 or F2 fields in the joined records and where to write the fields (F1ONLY, F2ONLY or BOTH).

Conceptually, JOINKEYS application processing proceeds as follows:

- Subtask1 copies the SORTJNF1 (F1) records as directed by the JOINKEYS statement. As a result, it copies the unchanged SORTJNF1 records to the main task.
- Subtask2 copies the SORTJNF2 (F2) records as directed by the JOINKEYS statement. As a result, it copies the unchanged SORTJNF2 records to the main task.
- The main task joins the records passed from subtask1 and subtask2 as directed by the specified JOINKEYS, JOIN and REFORMAT statements, resulting in the following joined records (paired and unpaired):

  ```
  Carrie    F101No    Carrie    F201
  David     F102$$$$$$$$$$$$$$$$$$$$
  Frank     F103$$$$$$$$$$$$$$$$$$$$
  Holly     F104Yes   Holly     F202
  $$$$$$$$$$$$$$$Yes   Karen     F203
  $$$$$$$$$$$$$$$No    Sri Hari  F204
  Vicky     F105Yes   Vicky     F205
  ```

  For F1 records without a match in F2 (for example, the F102 record), the F2 field is filled with the FILL character. For F2 records without a match in F1 (for example, the F203 record), the F1 field is filled with the FILL character. For F1 records with a match in F2 (for example, the F101 and F201 records), no FILL characters are used.
- The first OUTFIL statement finds records with the FILL character in position 15. These are the F1 records without a match in F2. The F1 field is in positions 1-14 of the joined record, so those positions are written to the F1ONLY file. Thus, F1ONLY contains these records:

  ```
  David     F102
  Frank     F103
  ```

- The second OUTFIL statement finds records with the FILL character in position 1. These are the F2 records without a match in F1. The F2 field is in positions 15-34 of the joined record, so those positions are written to the F2ONLY file. Thus, F2ONLY contains these records:

```
Yes   Karen     F203
No    Sri Hari  F204
```

- The third OUTFIL statement finds records without the FILL character in position 1 or position 15. These are the F1 and F2 records with a match. The F1 field is in positions 1-14 of the joined record and the F2 field is in positions 15-34 of the joined record, so each joined record is split into those two records and written to the BOTH file. The shorter F1 record is padded with blanks on the right to the length of the F2 record. Thus, BOTH contains these records:

```
Carrie    F101
No    Carrie    F201
Holly     F104
Yes   Holly     F202
Vicky     F105
Yes   Vicky     F205
```

Note: If you only want one record per key in BOTH, you can have the BUILD for FNAMES=BOTH specify the positions for just that record. For example, BUILD=(1,14) for the F1 records or BUILD=(15,20) for the F2 records.

Chapter 5. Using your own user exit routines

User exit routine overview

DFSORT can pass program control to your own routines at points in the executable code called *user exits*. Your user exit routines can perform a variety of functions including deleting, inserting, altering, and summarizing records.

If you need to perform these tasks, you should be aware that DFSORT already provides extensive facilities for working with your data in the various DFSORT program control statements. See the discussions of the INCLUDE, OMIT, INREC, OUTFIL, OUTREC, and SUM program control statements in Chapter 3, "Using DFSORT program control statements," on page 77. You might decide that using a program control statement to work with your records is more appropriate to your needs.

Although this chapter discusses only routines written in assembler or COBOL, you can write your exit routines in any language that can:

- Pass and accept the address into general register 1 of a:
  - Record
  - Full word of zeros
  - Parameter list.
- Pass a return code in register 15.

You can easily activate user exit routines at run-time with the MODS program control statement (see "MODS control statement" on page 167). Alternatively, under certain circumstances you can also activate a user exit routine by passing the address of your exit routine in the invocation parameter list. See Chapter 6, "Invoking DFSORT from a program," on page 517 for details.

Parameters that affect the way user exit routines are handled include:

- The MODS statement, explained in "MODS control statement" on page 167
- The E15=COB and E35=COB PARM options of the EXEC statement, explained in "Specifying EXEC/DFSPARM PARM options" on page 32
- The COBEXIT option of the OPTION statement, explained in "OPTION control statement" on page 173
- The EXITCK option of the OPTION statement, explained in "OPTION control statement" on page 173.

Note: To avoid ambiguity in this chapter, it is assumed that the IBM default, EXITCK=STRONG, was selected at your site.

Certain user exit routines can be written in COBOL, using a special interface. If you write your exit routines in PL/I, you must use the PL/I subroutine facilities.

You might need to reserve space to be used by your exits. See "Use main storage efficiently" on page 755 for more information about storage.

DFSORT program phases

A DFSORT program phase is a large DFSORT component designed to perform a specific task such as writing the output file. Various user exits are contained in the input and output phases and are activated at a particular time during DFSORT processing. The input phase is used only for a sort or copy. When the output phase is completed, DFSORT returns control to the operating system or invoking program. Figure 10 on page 468 is a representation of DFSORT input/output logic.

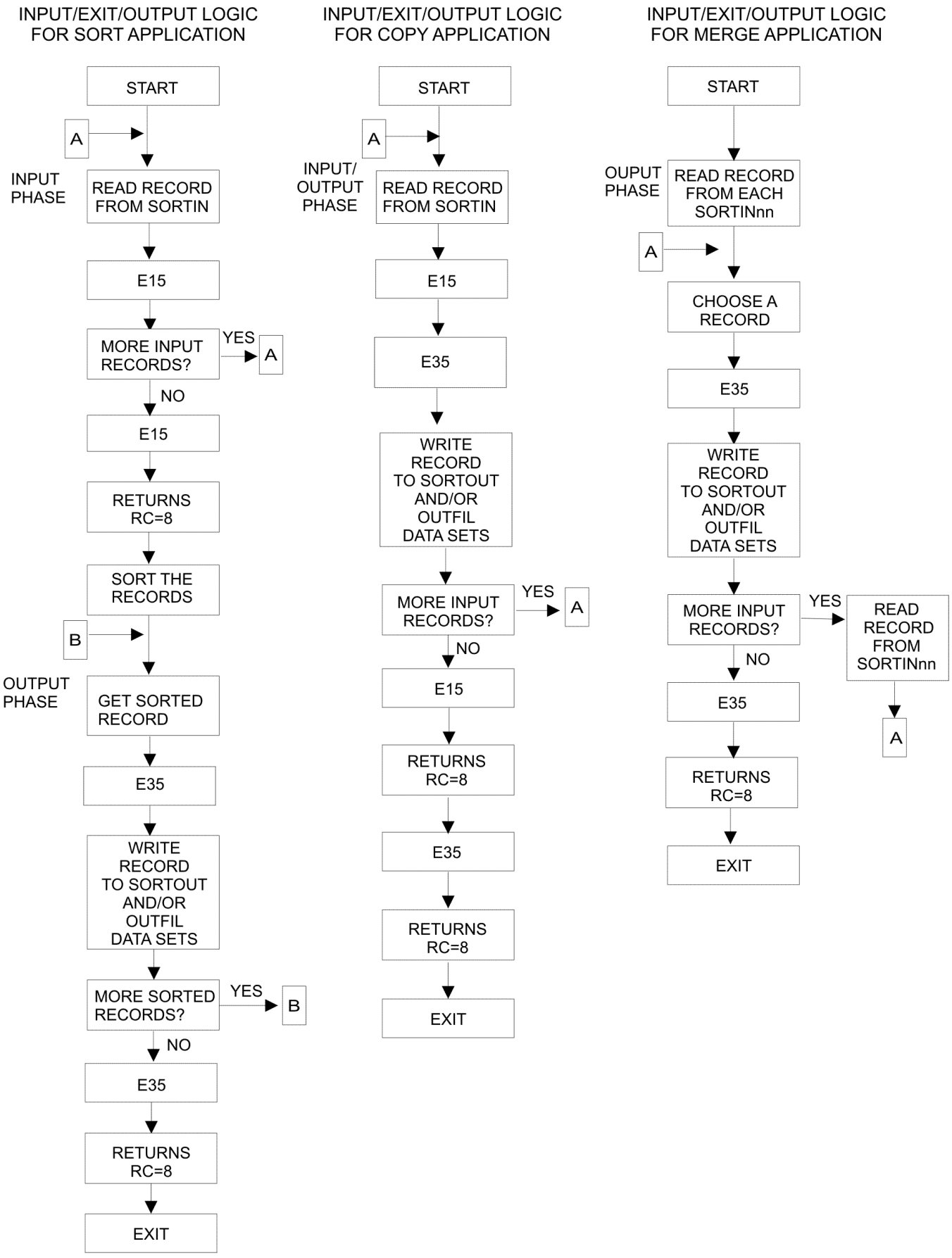


Figure 10. Examples of DFSORT Input/User Exit/Output Logic

Functions of routines at user exits

You can use exit routines to accomplish a variety of tasks:

- Open and initialize data sets
- Modify control fields
- Insert, delete, or alter records
- Sum records
- Handle special I/O conditions
- Determine action when intermediate storage is insufficient
- Close data sets
- Terminate DFSORT.

Figure 10 on page 468, Table 66 on page 470, and Table 67 on page 470 summarize the functions of user exit routines and the exits and phases with which they can be associated.

The following restrictions apply to the applications for which specific user exits can be used:

- E11. E16, E17, E18 and E19 can be used for sort applications, but are ignored for copy or merge applications.
- E15 can be used for sort or copy applications, but is ignored for merge applications.
- E32 can be used for merge applications, but is ignored for copy or sort applications.
- E61 can be used for sort or merge applications, but is ignored for copy applications.
- E31, E35, E37, E38 and E39 can be used for sort, copy or merge applications.

DFSORT input/user exit/output logic examples

Figure 10 on page 468 gives examples of the logic flow for sort, copy, and merge applications as it relates to SORTINnn, E15 or E35 user exits, and SORTOUT. The intent is to show how your E15 and E35 user exits fit into the logic of an application. All possible paths are not covered. For simplicity, it is assumed that all of the applicable data sets and exits are present and that records are not inserted or deleted. (For a merge, similar logic would be used if an E32 user exit supplied the records rather than SORTINnn data sets.)

Figure 10 on page 468 illustrates the following logic:

- E15 and E35 user exits continue to be entered until they pass back a return code of 8. If your user exit passes a return code of 8 to DFSORT when input records still remain to be processed, the records are processed by DFSORT *without* being passed to your exit.
- During a sort, each record is read from SORTIN and passed to E15 user exit. When *all* of the records have been processed in this manner, they are sorted by DFSORT, then each sorted record is passed to E35 and written to the output data sets.
- During a copy, each record is read from SORTIN, passed to E15 and E35 user exits, and written to the output data sets.
- During a merge, one record is initially read from each SORTINnn data set. The record to be output is chosen, passed to E35, and written to the output data sets. The chosen record is then replaced by reading a record from the same SORTINnn data set and the process continues.

  Note: For a merge application, records deleted during an E35 user exit routine are not sequence-checked. If you use an E35 user exit routine without an output data set, sequence checking is not performed at the time the records are passed to the E35 user exit; therefore, you must ensure that input records are in correct sequence.

Table 66. Functions of Routines at Program User Exits (Sort)

Functions	Sort Input Phase	Sort Output Phase
Open/Initialize	E11, E15 user exits	E31 user exit
Modify control fields	E61 user exit	N/A
Insert, Delete/Alter	E15 user exit	E35 user exit
Sum records		E35 user exit"1" on page 470
Handle special I/O conditions: QSAM/BSAM and VSAM SORTIN QSAM/BSAM SORTOUT VSAM SORTOUT	 E18 user exit E19 user exit"2" on page 470 N/A	 E38 user exit"2" on page 470 E39 user exit"3" on page 470 E39 user exit"3" on page 470
Determine action when intermediate storage is insufficient	E16 user exit"4" on page 470	N/A
Close/housekeeping	E15, E17 user exits	E35, E37 user exits
Terminate DFSORT	E15 user exit	E35 user exit

Note:

1. The SUM control statement can be used instead of your own routine to sum records.
2. Applies only to a tape work data set sort.
3. E39 can be used for SORTOUT, but not for OUTFIL data sets.
4. Applies only to a tape work data set sort or a Peerage/Vale sort without work data sets.

Table 67. Functions of Routines at Program User Exits (Copy and Merge)

Functions	Copy	Merge
Open/Initialize	E15, E31 user exits	E31 user exit
Modify control fields	N/A	E61 user exit
Insert	E15, E35 user exits	E32, E35 user exits
Delete/alter	E15, E35 user exits	E35 user exit
Sum records	E35 user exit	E35 user exit"1" on page 470
Handle special I/O conditions: QSAM/BSAM and VSAM SORTIN(nn) QSAM/BSAM and VSAM SORTOUT	 E38 user exit E39 user exit	 E38 user exit E39 user exit
Close/housekeeping	E35, E37 user exits	E35, E37 user exits
Terminate DFSORT	E15, E35 user exits	E32, E35 user exits

Note:

1. The SUM control statement can be used instead of your own routine to sum records.

Opening and initializing data sets

You can write your own routines to open data sets and perform other forms of initialization; you must associate these routines with the E11, E15, E31 and E35 user exits.

To check labels on input files, use the E18 and E38 user exits.

Modifying control fields

You can write a routine to alter control fields before DFSORT compares them. This allows you, for example, to normalize floating-point control fields. It also allows you to modify the order in which the records are finally sorted or merged, a function for which you would usually use DFSORT's ALTSEQ program control statement. You must associate this routine with the E61 user exit.

When an E61 user exit is used, the subsequent comparisons always arrange the modified control fields in ascending order.

Note: Although you are altering control fields before a compare, your original records are not altered.

Inserting, deleting, and altering records

You can write your own routines to delete, insert, or alter records. You must associate these routines with the E15, E32, and E35 user exits.

Note: DFSORT also provides INCLUDE and OMIT statements, and OUTFIL INCLUDE and OMIT parameters that automatically include or delete records based on your field criteria. For more information on these control statements, refer to Chapter 3, “Using DFSORT program control statements,” on page 77.

Summing records

You can sum records for output by using the E35 user exit. However, you can also use DFSORT's SUM program control statement to accomplish this without a user exit. See “SUM control statement” on page 433.

Handling special I/O

DFSORT contains four exits to handle special I/O conditions: E18 and E38 user exits for SORTIN and SORTINnn, and E19 and E39 user exits for SORTOUT (but not for OUTFIL data sets). They are particularly useful for a tape work data set sort. With all disk work data set sorts, E19 and E38 user exits are ignored.

You can use these exits to incorporate your own or your site's I/O error recovery routines into DFSORT. Your read and write error routines must reside in a partitioned data set (library). Your library routines are brought into main storage with their associated phases. When DFSORT encounters an uncorrectable I/O error, it passes the same parameters as those passed by QSAM/BSAM or VSAM. If no user routines are supplied and an uncorrectable read or write error is encountered, DFSORT issues an error message and then terminates.

With QSAM/BSAM, the following information is passed to your synchronous error routine:

- General registers 0 and 1 are unchanged; they contain the information passed by QSAM/BSAM, as documented in the data management publications.
- General register 14 contains the return address of DFSORT.
- General register 15 contains the address of your error routine.

VSAM will go directly to any routine specified in the EXLST macro you passed to DFSORT via the E18, E38, or E39 user exit, as appropriate. Your routine must return to VSAM via register 14. For details, see *z/OS DFSMS Macro Instructions for Data Sets* or *z/OS DFSMS Using Data Sets*.

Routines for read errors

You must associate these routines with the E18 and E38 user exits. They must pass certain control block information back to DFSORT to tell it whether to accept the record as it is, skip the block, or request termination. They can also attempt to correct the error.

Routines for write errors

You must associate these routines with the E19 and E39 user exits. These routines can perform any necessary abnormal end-of-task operations for SORTOUT before DFSORT is terminated.

VSAM user exit functions

There are three user exits that can be used with VSAM SORTIN, SORTINnn, and SORTOUT data sets (but not with OUTFIL data sets), to supply passwords or a user exit list to journal a VSAM data set. They can carry out other VSAM exit functions except EODAD. The user exits are E18 for sort SORTIN, E38 for merge SORTINnn or copy SORTIN, and E39 for SORTOUT.

Determining action when intermediate storage is insufficient

You can write a routine to direct DFSORT program action if DFSORT determines that insufficient intermediate storage is available to handle the input data set. You must associate this routine with the E16 user exit for sorts using tape work data sets. For a sort that uses tape data sets, you can choose between sorting current records only, trying to complete the sort, or terminating DFSORT. For more details, see “Exceeding tape work space capacity” on page 804.

Closing data sets

You can write your own routines to close data sets and perform any necessary housekeeping; you must associate these routines with the E15, E17, E35, and E37 user exits. To write SORTOUT labels, use the E19 and E39 user exits. If you have an end-of-file routine you want to use for SORTIN, include it at the E18 user exit.

Terminating DFSORT

You can write an exit routine to terminate DFSORT before all records have been processed. You must associate these routines with the E15, E16, E32, and E35 user exits.

32-bit and 64-bit parameter lists

For the E15, E32 and E35 exits, DFSORT supports both a 32-bit parameter list and a 64-bit parameter list. For all of the other exits, DFSORT supports only a 32-bit parameter list.

If you want DFSORT to use the 64-bit parameter list for your E15, E32 or E35, you must call DFSORT with the 64-bit invocation list, and:

- use a MODS control statement for E15 or E35 with the N64 parameter or
- set on bit 4 (for E15 or E32) or bit 5 (for E35) of byte 9 in the 64-bit invocation list

Otherwise, DFSORT will use the 32-bit parameter list for your E15, E32 or E35.

See Chapter 6, “Invoking DFSORT from a program,” on page 517 for details of using the 64-bit invocation parameter list.

See the sections that follow on E15, E32 and E35 for complete details on using the 32-bit and 64-bit parameter lists.

64-bit address terminology

When referring to 64-bit addresses in this chapter:

- A **clean 24-bit address** refers to binary zeros in the high-order 5 bytes followed by a 24-bit address in the low-order 3 bytes, that is, X'0000000000aaaaaa'
- A **clean 31-bit address** refers to binary zeros in the high-order 4 bytes followed by a 31-bit address in the low-order 4 bytes, that is, X'00000000aaaaaaaa' where Bit 32 of the 31-bit address must be 0.
- A 64 bit address uses all 64-bits for the address.

The following summarizes the previous information:

Table 68. Summary of 64-bit Terminology for Exits

Summary of 64-bit Terminology for ExitsAddress	Bits 0-31	Bit 32	Bits 33-39	Bits 40-63
24-bit	0	0	0	Address
31-bit	0	0	Address	
64-bit	Address			

Addressing and residence modes for user exits

To allow user exits called by Blockset or Peerage/Vale to reside above or below 16MB virtual, and use either 24-bit, 31-bit or 64-bit addressing, DFSORT supplies these features:

- To ensure that DFSORT enters your user exit with the correct addressing mode, you must observe these rules:
  - If the user exit name is specified in a MODS control statement, the user exit is entered with the addressing mode indicated by the linkage editor attributes of the routine (for example, 31-bit addressing in effect if AMODE 31 is specified).
  - If the address of the exit is passed to DFSORT (preloaded exit) via the 24-bit list, the user exit is entered with 24-bit addressing in effect.
  - If the address of the user exit is passed to DFSORT via the extended parameter list (preloaded exit), the user exit is entered with 24-bit addressing in effect if bit 0 of the user exit address in the list is 0 or with 31-bit addressing in effect if bit 0 of the user exit address in the list is 1.
  - If the address of the user exit is passed to DFSORT via the 64-bit parameter list (preloaded exit), the user exit is entered with the addressing mode in effect as specified by the appropriate flag in byte 8 or 9 of the parameter list as follows:

Table 69. Addressing Mode Flags for Exits in 64-bit Parameter List

Byte	Bit	Meaning when on
8	0	Enter E15 or E32 in AMODE 24
	1	Enter E15 or E32 in AMODE 31
	2	Enter E15 or E32 in AMODE 64
	3	Enter E35 in AMODE 24
	4	Enter E35 in AMODE 31
	5	Enter E35 in AMODE 64
	6	Enter E18 in AMODE 24
	7	Enter E18 in AMODE 31
9	1	Enter E39 in AMODE 24
	2	Enter E39 in AMODE 31

- User exits can return to DFSORT with either 24-bit, 31-bit, or 64-bit addressing in effect. The return address that DFSORT placed in register 14 must be used.

  Note: For a conventional merge or tape work data set sort application, user exits:

  – must reside below 16MB virtual
  – must use 24-bit addressing mode
  – must not use a user exit address constant.

How user exit routines affect DFSORT performance

Before writing a user exit routine, consider the following factors:

- Your routines occupy main storage that would otherwise be available to DFSORT. Because its main storage is restricted, DFSORT might need to perform extra passes to sort the data. This, of course, increases sorting time.
- User exit routines increase the overall run-time.

  Attention: Several of the user exits give your routine control once for each record until you pass a "do not return" return code to DFSORT. You must remember this when designing your routines.
- Using INCLUDE, OMIT, INREC, OUTFIL, OUTREC, and SUM instead of user exit routines allows DFSORT to perform more efficiently.

Summary of rules for user exit routines

When preparing your routines, remember that:

- User-written routines must follow standard linkage conventions and use the required interfaces. COBOL E15 and E35 user exits must use the special interface provided.
- To use an E32 user exit, your invoking program must pass its address to DFSORT in the parameter list.
- To use any other user exit, you must associate your routine with the appropriate user exits using the MODS control statement. See “MODS control statement” on page 167.
- Your invoking program can alternatively pass the address of an E15, E18, E35, and E39 user exit to DFSORT in the parameter list.
- When Blockset or Peerage/Vale is used and your user exits are reenterable, the entire DFSORT program is reenterable.
- If you are using ASCII input, remember that data presented to your user exits at user exits are in EBCDIC format. If the E61 user exit is used to resolve ASCII collating for special alphabetic characters, substituted characters must be in EBCDIC, but the sequencing result depends on the byte value of the ASCII translation for the substituted character.

Loading user exit routines

You must assemble or compile each user exit as a separate program. If your user exit operates independently, bind or link-edit it separately into a partitioned data set (library) with the member name to be used in the MODS statement. If your user exit operates in conjunction with other user exits in the same phase (for example, E11, E15, and E17 user exits all use the same DCB), you can request DFSORT to dynamically bind or link-edit them together (see MODS statement). Alternatively, you can bind or link-edit them together into a partitioned data set following these rules:

1. Specify RENT as a bind or linkage editor parameter.
2. Include an ALIAS statement for each user exit using the external entry name of the routine (for example, the CSECT name).
3.
4. Specify the appropriate ALIAS name for each user exit on the MODS statement.

DFSORT includes the names and locations of your user exits in the list of modules to be run during each phase. No user exit is loaded more than once in a program phase, but the same user exit can appear in different phases. For example, you can use the same Read Error user exit in both phases, but not twice in one phase.

The length you specify for a user exit must include storage for the user exit itself as well as any storage used by the user exit outside of the load modules such as I/O buffers or COBOL library subroutines. If you specify a ddname for a user exit in the MODS statement, it must match the DD statement that defines the library containing that user exit. For example:

```
//MYLIB DD   DSNAME=MYRTN, etc.
         .
         .
         .
   MODS   E15=(MODNAME,500,MYLIB,N)
```

User exit linkage conventions

To enter a user exit, DFSORT loads the address of the DFSORT return point in register 14 and the address of the user exit routine in register 15. A branch to the address in register 15 is then performed.

The general registers used by DFSORT for linkage and communication of parameters observe operating system conventions. When your routine gets control, the general registers have the following contents:

Register
Contents

1
DFSORT places the address of a parameter list in this register.

13
DFSORT places the address of a standard save area in this register. The area can be used to save contents of registers used by your user exit. The first word of the area contains the characters SM1 in its three low-order bytes.

14
Contains the address of DFSORT return point.

15
Contains the address of your user exit. This register can be used as a base register for your user exit; your user exit can also use it to pass return codes to DFSORT.

You can return control to DFSORT by performing a branch to the DFSORT return point address in register 14 or by using a RETURN macro instruction. The RETURN instruction can also be used to set return codes when multiple actions are available at a user exit.

Your user exit must save all the general registers it uses. You can use the SAVE macro instruction to do this. If you save registers, you must also restore them; you can do this with the RETURN macro instruction.

Linkage examples

When calling your user exit, DFSORT places the return address in general register 14 and your routine's entry point address in general register 15. DFSORT has already placed the register's save area address in general register 13. DFSORT then makes a branch to your routine.

Your routine for the E15 user exit might incorporate the following assembler instructions:

```
     ENTRY  E15
     .
     .
E15  SAVE   (5,9)
     .
```

```
      .
      RETURN (5,9)
```

This coding saves and restores the contents of general registers 5 through 9. The macro instructions are expanded into the following assembler language code:

```
      ENTRY  E15
      .
      .
E15   STM    5,9,40(13)
      .
      .
      LM     5,9,40(13)
      BR     14
```

If multiple actions are available at a user exit, your routine sets a return code in general register 15 to inform DFSORT of the action it is to take. The following macro instruction can be used to return to DFSORT with a return code of 12 in register 15:

```
RETURN  RC=12
```

A full explanation of linkage conventions and the macro instructions discussed in this section is in *z/OS MVS Programming: Assembler Services Guide*.

Dynamically binding or link-editing user exit routines

You can dynamically bind or link-edit any user exit routine written in any language that has the ability to pass the location or address of a record or parameter in general register 1 and a return code in register 15 (see MODS statement). This does not include E15 and E35 user exits written in COBOL.

Dynamic binding or link-editing does not support AMODE 31 or RMODE 31 for the option T. The user exits that are bound or link-edited *together* by DFSORT are not loaded above 16MB virtual and cannot be entered in 31-bit addressing mode. User exits bound or link-edited with the S option retain the AMODE and RMODE attributes of the object modules and are loaded above or below 16MB virtual depending upon the load module's RMODE; they are entered in the addressing mode of the user exit.

Note:

1. The Blockset technique is not used for dynamic binding or link-editing.
2. Dynamic binding or link-editing cannot be used with copy.

When the option T is specified for a user exit routine, that routine *must* contain an entry point whose name is that of the associated program user exit. This is to accommodate special DFSORT dynamic binder or link-edit requirements. For example, when the option T is specified on the MODS statement for E35, the following assembler instructions must be included in the user exit routine associated with the E35 user exit:

```
      ENTRY E35
E35    .
       .
```

or

```
E35  CSECT
       .
       .
```

In all other circumstances, the user exit is *not* required to have an entry point that has the same name as that of the associated program user exit.

Assembler user exit routines (input phase user exits)

You can use these program user exits in the DFSORT input phase:

- E11

- E15
- E16
- E17
- E18
- E19
- E61

These user exits are discussed in sequence. To determine whether a particular user exit can be used for your application, refer to Table 66 on page 470 and Table 67 on page 470.

E11 user exit: opening data sets/initializing routines

You might use routines at this user exit to open data sets needed by your other routines in the input phase. It can also be used to initialize your other routines. Return codes are not used, however.

Note: To avoid special linkage editor requirements (see "Summary of rules for user exit routines" on page 474), you can include these functions in your E15 user exit rather than in a separate E11 user exit routine.

E15 user exit: passing or changing records for sort and copy applications

If you write your E15 user exit in COBOL, see "COBOL user exit routines" on page 499 and "COBOL E15 user exit: passing or changing records for sort" on page 501.

The EXITCK option affects the way DFSORT interprets certain return codes from user exit E15. To avoid ambiguity, this section assumes that the IBM default, EXITCK=STRONG, was selected at your site. For complete information about E15 return codes in various situations with EXITCK=STRONG and EXITCK=WEAK, see "E15/E35 return codes and EXITCK" on page 512.

DFSORT enters the E15 user exit routine each time a new record is brought into the input phase. DFSORT continues to enter E15 (even when there are no input records) until the user exit tells DFSORT, with a return-code of 8, not to return.

See Figure 10 on page 468 for logic flow details.

Some uses for the E15 user exit are:

- Adding records to an input data set
- Passing an entire input data set to DFSORT
- Deleting records from an input data set
- Changing records in an input data set.

Note:

1. If your E15 user exit is processing variable-length records, include a 4-byte RDW at the beginning of each record you change or insert, before you pass it back to DFSORT. The format of an RDW is described in *z/OS DFSMS Using Data Sets* or *System Programming Reference*. (Alternatively, you can pad records to the maximum length and process them as fixed-length.)
2. DFSORT uses the specified or defaulted value for L2 in the RECORD statement to determine the length of the records your E15 user exit passes back to DFSORT. For fixed-length records, be sure that the length of each record your E15 user exit changes or inserts corresponds to the specified or defaulted L2 value. For variable-length records, be sure that the RDW of each record your E15 user exit changes or inserts indicates a length that is less than or equal to the specified or defaulted L2 value. Unwanted truncation or abends may occur if DFSORT uses the wrong length for the records passed to it by your E15 user exit.

   For details of the L2 value, see "RECORD control statement" on page 418

   .
3. If you use the E15 user exit to pass all your records to DFSORT, you can omit the SORTIN DD statement, in which case you must include a RECORD statement in the program control statements.

4. If you invoke DFSORT from an assembler program and pass the address of your E15 user exit in the parameter list, DFSORT ignores the SORTIN data set and terminates if you specify E15 in a MODS statement.
5. If you omit the SORTIN DD statement, or it is ignored, all input records are passed to DFSORT through your routine at user exit E15. The address of each input record in turn is placed in general register 1, and you return to DFSORT with a return code of 12. When DFSORT returns to the E15 user exit after the last record has been passed, you return to DFSORT with a return code of 8 in register 15, which indicates "do not return."
6. DFSORT continues to reenter your E15 user exit until a return code of 8 is received. However, if STOPAFT is in effect, no additional records are inserted to DFSORT after the STOPAFT count is satisfied (even if you pass back a return code of 12).
7. An RDW must be built for variable-length VSAM records (see *z/OS DFSMS Using Data Sets*).

Information DFSORT passes to your routine at E15 user exit

Your E15 user exit routine is entered each time a new record is brought into the input phase. DFSORT passes two fields to your routine each time it is entered:

- **The address of the new record**. End of input is reached when there are no more records to pass to your E15 user exit; DFSORT indicates end of input by setting this address to zero before entering your E15 user exit. If there are no records in the input data set (or no input data set), this address is zero the first time your E15 is entered.

  After end of input is reached, DFSORT continues to enter your user exit routine until you pass back a return code of 8.

  Your E15 user exit must not change the address of the new record.
- **The user exit address constant**. If you invoked DFSORT with a user exit address constant in the parameter list, the address constant is passed to your E15 user exit the first time it is entered. This address constant can be changed by your E15 user exit any time it is entered; the address constant is passed along on subsequent entries to your E15 user exit and also on the first entry to your E35 user exit. For example, you can obtain a dynamic storage area, use it in your E15 user exit, and pass its address to your E35 user exit.

Note: The user exit address constant must not be used for a tape work data set sort application.

Block Support for E15.

This support provides a mechanism for inserting block of records or single records for all DFSORT functions (SORT, COPY, and MERGE) for both fixed length records (FLR) and variable length records (VLR). Inserting block of records will significantly reduce the number of calls of the user exit, and will reduce the transfer of records between user's storage and DFSORT storage. Furthermore, this block support does not affect the existing support of single records interface. In other words in one run it is possible to combine the use of block of records processing with single record processing. Using the 64-bit parameter list for E15 users can now

- Insert of block of records from the E15 exit
- Insert single records from the E15 exit
- Insert of block of records or single record from the E15 exit at each invocation of the E15 exit from DFSORT
- Input of records from SORTIN data set only
- Input of records from SORTIN data set and insert block of records or single record from the E15 exit in one run.

The E15 block list contains the information on the number of blocks, the sizes of blocks, and addresses of blocks where DFSORT will receive records for inserting from the E15 exit.

Table 70. Block list parameters for E15 exit

Block list parameters for E15 exitLocation	Contents	Description
Bytes 1 through 8	X'00000000'	Address of user block list area
Bytes 9 through 16	X'00000000'	Maximum length of block of records for block list type 2
Bytes 17 through 24	X'00000000'	Maximum number of block for block list type 2 or Maximum number of records for block list type 1
Bytes 25 through 28	X'00000000'	Maximum length of record
Bytes 29 through 29	X'00'	Format of record (F =FLR V=VLR)
Bytes 30 through 30	X'00'	Block List Type **Note :** 1 = Single record 2 = Block of records
Bytes 31 through 32	X'00'	Reserved

Notes:

It is the user's responsibility for allocating the areas of virtual storage to form blocks that will be used to populate the records to be inserted to DFSORT through the E15 exit (input block of records) and the areas to receive the sorted records from DFSORT (output block of records) through the E35 exit.

Any users who want to use block support should specify in DFSORT's 64-bit invocation parameters list the indicator of block support (offset X'17': X'80' for E15 exit and X'40' for E35 exit) and the address of the user block list parameters Area(offset X'60') containing characteristics of E15/E35 block lists.

The 64-bit invocation parameters list can be created by the user in any user's virtual storage (24/31/64 bit).

The user exit can have any addressing mode (AMODE 24/31/64).

The user invokes DFSORT using one of the names ICEMAN64, SORT64, or ICEDFS64 and passes the address of the 64-bit invocation parameters list in the 64-bit general register 1.

Which E15 parameter list will be used

For the E15 exit, DFSORT supports both a 32-bit parameter list and a 64-bit parameter list. If you want DFSORT to use the 64-bit parameter list for your E15, you must call DFSORT with the 64-bit invocation list, and:

- use a MODS control statement for E15 with the N64 parameter or
- set on bit 4 of byte 9 in the 64-bit invocation list.

Otherwise, DFSORT will use the 32-bit parameter list for your E15.

See Chapter 6, “Invoking DFSORT from a program,” on page 517 for details of using the 64-bit invocation parameter list.

32-bit E15 parameter list

In 32-bit general register 1, DFSORT places the address of a parameter list that contains the record address and the user address constant. The list is two fullwords long and begins on a fullword boundary. The format of the parameter list is:

Table 71. 32-bit E15 User Exit Parameter List

32-bit E15 User Exit Parameter ListLocation	Contents
Bytes 1 through 4	Address of the new record
Bytes 5 through 8	User exit address constant

DFSORT provides a 72-byte Format 0 save area pointed to by 32-bit register 13 in which the exit can save the 32-bit registers.

Before returning control to DFSORT, you must:

- Place the address of the record in 32-bit register 1.
- Put the return code in 32-bit register 15.

64-bit E15 parameter list

In 64-bit general register 1, DFSORT places the address of a parameter list that contains the record address or the address of block list and the user address constant. The list is three doublewords long and begins on a doubleword boundary. The format of the parameter list is:

Table 72. 64-bit E15 User Exit Parameter List

64-bit E15 User Exit Parameter ListLocation	Value	Contents
Bytes 1 through 8	X'00000000'	Address of the new record or Address of block list
Bytes 9 through 16	X'00000000'	User exit address constant
Bytes 17 through 24	X'00000000'	Address of user block list parameter area

DFSORT provides a 144-byte Format 4 save area pointed to by 64-bit register 13 in which the exit can save the 64-bit registers.

The E15 should expect and use 64-bit addresses.

Before returning control to DFSORT, you must:

- Place the address of the single record or the block list address in 64-bit register 1. This must be a 64-bit address, a clean 31-bit address or a clean 24-bit address.
- Put the return code in 64-bit register 15.

The DFSORT target library, SICEUSER, contains a mapping macro called ICEPL64, which provides a separate Assembler DSECT for the E15 64-bit parameter list.

E15 return codes

Your E15 routine must pass a return code to DFSORT. Following are the return codes for the E15 user exit:

Return Code
Description

00 (X'00')
No Action/Record Altered

04 (X'04')
Delete Record

08 (X'08')
Do Not Return

12 (X'0C')
Insert single record

16 (X'10')
Terminate DFSORT

20 (X'14')
Insert of block of records

0: No Action
If you want DFSORT to retain the record unchanged, place the address of the record in general register 1 and return to DFSORT with a return code of 0 (zero).

0: Record Altered
If you want to change the record before passing it back to DFSORT, your routine must move the record into a work area, perform whatever modification you want, place the address of the modified record in general register 1, and return with a return code of 0 (zero).

4: Delete Record
If you want DFSORT to delete the record from the input data set, return to DFSORT with a return code of 4. You need not place the address of the record in general register 1.

8: Do Not Return
DFSORT continues to return control to the user routine until it receives a return code of 8. After that, the user exit is not used again during the DFSORT application. You need not place an address in general register 1 when you return with a return code of 8. *Unless you are inserting records after the end of the data set, you must pass a return code of 8 when the program indicates the end of the data set*. It does this by passing your routine a zero address in the parameter list.

If your user exit routine passes a return code of 8 to DFSORT when input records still remain to be processed, the remaining records are processed by DFSORT, but are *not* passed to your user exit.

12: Insert Record
To add a record before the record whose address was just passed to your routine, place the address of the record to be added in general register 1 and return to DFSORT with a return code of 12. DFSORT keeps returning to your routine with the same record address as before so that your routine can insert more records at that point or alter the current record. You can make insertions after the last record in the input data set (after DFSORT places a zero address in the parameter list). *DFSORT keeps returning to your routine until you pass a return code of 8*.

16: Terminate DFSORT
If you want to terminate DFSORT, return with a code of 16. DFSORT then returns to its calling program or to the system with a return code of 16.

20: Insert Block of records
To add a block of records before the record whose address was just passed to your routine, place the address of the block of records to be added in 64-bit general register 1 and return to DFSORT with a return code of 20. DFSORT keeps returning to your routine until you pass a return code of 8.

See "E15/E35 return codes and EXITCK" on page 512 for complete details of the meanings of return codes in various situations.

Storage usage for E15 user exit

DFSORT obtains storage (using GETMAIN or STORAGE OBTAIN) for the parameter list and the records it passes to your E15 user exit routine. You must not attempt to modify or free the storage obtained by DFSORT.

If you need to obtain storage for use by your E15 user exit routine, such as to pass altered records to DFSORT, you can use the following strategy:

1. The first time your exit is called, obtain the storage you need
2. Use the storage you obtained each time your exit is called
3. Free the storage before you pass back return code 8 to DFSORT

Note: When you obtain your storage you can save its address in the user exit address constant and restore it on each subsequent call to your exit.

E16 user exit: handling intermediate storage miscalculation

For a tape work data set sort or a Peerage/Vale sort without work data sets, you would use a routine at this user exit to decide what to do if the sort exceeds its calculated estimate of the number of records it can handle for a given amount of main storage and intermediate storage. This user exit is ignored for a sort with work data sets because DFSORT uses the WRKSEC option to determine whether secondary allocation is allowed. See "SORTWKdd DD statement" on page 69. See also "Exceeding tape work space capacity" on page 804.

Note: When using magnetic tape, remember that the system uses an assumed tape length of 2400 feet. If you use tapes of a different length, the Nmax figure is not accurate; for shorter tapes, capacity can be exceeded before "NMAX EXCEEDED" is indicated.

E16 return codes

Your E16 routine must pass a return code to DFSORT. Following are the return codes for the E16 user exit:

Return Code
Description

00 (X'00')
Sort Current Records Only

04 (X'04')
Try to Sort Additional Records

08 (X'08')
Terminate DFSORT

0: Sort Current Records Only
If you want DFSORT to continue with only that part of the input data set it estimates it can handle, return with a return code of 0 (zero). Message ICE054I contains the number of records with which sort is continuing. You can sort the remainder of the data set on one or more subsequent runs, using SKIPREC to skip over the records already sorted. Then you can merge the sort outputs to complete the operation.

4: Try to Sort Additional Records
If you want DFSORT to continue with all of the input data set, return with a return code of 4. If tapes are used, enough space might be available for DFSORT to complete processing. If enough space is not available, DFSORT generates a message and terminates. Refer to "Exceeding tape work space capacity" on page 804.

8: Terminate DFSORT
If you want DFSORT to terminate, return with a return code of 8. DFSORT then returns to its calling program or to the system with a return code of 16.

E17 user exit: closing data sets

Your E17 user exit routine is entered once at the end of the input phase. It can be used to close data sets used by your other routines in the phase or to perform any housekeeping functions for your routines.

Note: To avoid special linkage editor requirements (see "Summary of rules for user exit routines" on page 474), you can include these functions in your E15 user exits rather than in a separate E17 user exit routine.

E18 user exit: handling input data sets

You can use this user exit to handle special I/O conditions for QSAM/BSAM and VSAM input data sets.

Using E18 user exit with QSAM/BSAM

Your routines at this user exit can pass DFSORT a parameter list containing the specifications for three data control block (DCB) fields: SYNAD, EXLST, and EROPT. Your E18 user exit routine can also pass a fourth DCB field (EODAD) to DFSORT.

Note: If you are using a disk sorting technique, the EROPT option is ignored.

Your routines are entered at the beginning of each phase so that DFSORT can obtain the parameter lists. The routines are entered again during processing of the phase at the points indicated in the parameter lists. For example, if you choose the EXLST option, DFSORT enters your E18 user exit routine early in the sort (input) phase. DFSORT picks up the parameter list including the EXLST address. Later in the phase, DFSORT enters your routine again at the EXLST address when the data set is opened.

Information your routine passes to DFSORT at E18 user exit

Before returning control to DFSORT, your routine passes the DCB fields in a parameter list by placing the parameter list address in general register 1. The parameter list must begin on a fullword boundary and be a whole number of fullwords long. The high-order byte of each word must contain a character code that identifies the parameter. One or more of the words can be omitted. A word of all zeros marks the end of the list.

If VSAM parameters are specified, they are accepted but ignored.

The format of the list is shown as follows:

Byte 1	Byte 2	Byte 3	Byte 4
01	SYNAD field		
02	EXLST field		
03	00	00	EROPT code
04	EODAD field		
00	00	00	00

SYNAD
Contains the location of yourread synchronous error routine. This routine is entered only after the operating system has tried unsuccessfully to correct the error. The routine must be assembled as part of your E18 user exit routine. When the routine receives control, it must *not* store registers in the save area pointed to by register 13.

EXLST
Contains the location of a list of pointers to routines that you want used to check labels and accomplish other tasks not handled by data management. The list, and the routines to which it points, must be included in your read error routine. This parameter can only be used for EXLST routines associated with opening the first SORTIN data set.

EROPT
Indicates what action DFSORT must take when it encounters an uncorrectable read error. The three possible actions and the codes associated with them are:

X'80'
Accept the record (block) as is

X'40'
Skip the record (block)

X'20'
Terminate the program.

If you include this parameter in the DCB field list, you must place one of the previous codes in byte 4 of the word. Bytes 2 and 3 of the word must contain zeros.

When you use the EROPT option, the SYNAD field and the EODAD field must contain the appropriate address in bytes 2 through 4. Or, if no routine is available, bytes 2 and 3 must contain zeros, and byte 4 must contain X'01'. You can use the assembler instruction DC AL3(1) to set up bytes 2 through 4.

EODAD
Contains the address of your end-of-file routine. If you specify EODAD, you must include the end-of-file routine in your own routine.

A full description of the DCB fields are contained in *z/OS DFSMS Macro Instructions for Data Sets*.

Using E18 user exit with VSAM

If input to DFSORT is a VSAM data set, you can use the E18 user exit to perform various VSAM user exit functions and to insert passwords in VSAM input ACBs.

E18 user exit restrictions

If passwords are to be entered through a user exit and Blockset is not selected, the data set cannot be opened during the initialization phase. This means that MAINSIZE|SIZE=MAX must not be used because the program cannot make the necessary calculations.

Information your routine passes to DFSORT at E18 user exit

When you return to DFSORT, you must place the address of a parameter list in general register 1:

When you return to DFSORT, you must place the address of a parameter list in general register 1:Byte 1	Bytes 2 through 4
05	Address of VSAM user exit list
06	Address of password list
00	000000

If QSAM parameters are passed instead, they are accepted but ignored.

Either address entry can be omitted; if they both are included, they can be in any order.

E18 password list

A password list included in your routine must have the following format:

Two bytes on a halfword boundary:

Two bytes on a halfword boundary:Contents
Number of entries in list

Followed by the 16-byte entries:

Followed by the 16-byte entriesContents
8 bytes: ddname
8 bytes: Password

The last byte of the ddname field is destroyed by DFSORT. This list must not be altered at any time during the program. MAINSIZE|SIZE=MAX must not be used if this function is used.

E18 user exit list

The VSAM user exit list must be built using the VSAM EXLST macro instruction giving the addresses of your routines handling VSAM user exit functions. VSAM branches directly to your routines which must return to VSAM via register 14.

Any VSAM user exit function available for input data sets can be used except EODAD. If you need to do EODAD processing, write a LERAD user exit and check for X'04' in the FDBK field of the RPL. This will indicate input EOD. This field must not be altered when returning to VSAM because it is also needed by DFSORT.

For details, see *z/OS DFSMS Macro Instructions for Data Sets*.

Figure 11 on page 485 shows an example of code your program can use to return control to DFSORT.

```
   ENTRY   E18
           .
           .
E18        LA      1,PARMLST
           RETURN
           CNOP    0,4
PARMLST    DC      X'01'
           DC      AL3(SER)
           DC      X'02'
           DC      AL3(LST)
           DC      X'03'
           DC      X'000080'        EROPT CODE
           DC      A(0)
           DC      X'04'
           DC      AL3(QSAMEOD)
           DC      X'05'
           DC      AL3(VSAMEXL)
           DC      X'06'
           DC      AL3(PWDLST)
           DC      A(0)
           .
           .
VSAMEXL    EXLST   SYNAD=USYNAD,LERAD=ULERAD
PWDLST     DC      H'1'
           DC      CL8'SORTIN'      SORTIN DDNAME
           DC      CL8'INPASS'      SORTIN PASSWORD
USYNAD     ...                      VSAM SYNCH ERROR RTN
ULERAD     ...                      VSAM LOGIC ERROR RTN
SER        ...                      QSAM ERROR RTN
LST        DC      X'85',AL3(RTN)   EXLST ADDRESS LIST[1]
RTN        ...                      EXLST ROUTINE
QSAMEOD    ...                QSAM END OF FILE ROUTINE
```

Figure 11. E18 User Exit Example

[1] X'85'= X'80' plus X' 05', where:

- X'80' means this entry is the LAST ENTRY of the list.
- X'05' means this user exit is the data control block user exit.

For more information, refer to *z/OS DFSMS Using Data Sets*.

E19 user exit: handling output to work data sets

This user exit is used to handle write error conditions in the input phase when DFSORT is unable to correct a write error to a work data set. It is used only for a tape work data set sort.

Using E19 user exit with QSAM/BSAM

Your routines at this user exit can pass DFSORT a parameter list containing the specifications for two DCB fields (SYNAD and EXLST). Your routines are entered first early in the input phase so that DFSORT can obtain the parameter lists. The routines are entered again later in the phase at the points indicated by the options in the parameter lists.

Information your routine passes to DFSORT at E19 user exit

Before returning control to DFSORT, your routine passes the DCB fields in a parameter list by placing the parameter list address in general register 1. The list must begin on a fullword boundary and must be a

whole number of fullwords long. The first byte of each word must contain a character code that identifies the parameter. Either word can be omitted. A word of all zeros indicates the end of the list.

If VSAM parameters are passed, they are accepted but ignored.

The format of the list is as follows:

Byte 1	Byte 2	Byte 3	Byte 4
01	SYNAD field		
02	EXLST field		
00	00	00	00

SYNAD

This field contains the location of your write synchronous error routine. This routine is entered only after the operating system has unsuccessfully tried to correct the error. It must be assembled as part of your own routine.

EXLST

The EXLST field contains the location of a list of pointers. These pointers point to routines that are used to process labels and accomplish other tasks not handled by data management. This list, and the routines to which it points, must be included as part of your own routine.

A full description of these DCB fields can be found in *z/OS DFSMS Macro Instructions for Data Sets*.

E61 user exit: modifying control fields

You can use a routine at this user exit to lengthen, shorten, or alter any control field within a record. The E option for the s parameter on the SORT or MERGE control statement must be specified for control fields changed by this routine as described in "MERGE control statement" on page 163 and "SORT control statement" on page 423. After your routine modifies the control field, DFSORT collates the records in ascending order using the format(s) specified.[18]

Note:

1. Routine E61 will not be used with EFS fields that have a D1 format.
2. Routine E61 will not be used with Unicode format fields that have a UTF8/UTF16/UTF32 format.
3. Although your E61 routine alters control fields before a compare, your original records are not altered.
4. If locale processing is used for SORT or MERGE fields, an E61 user exit must not be used. DFSORT's locale processing may eliminate the need for an E61 user exit. See "OPTION control statement" on page 173 for information related to locale processing.
5. If Unicode processing is used for SORT or MERGE fields, an E61 user exit must not be used. DFSORT's COLLKEY processing may eliminate the need for an E61 user exit. See "OPTION control statement" on page 173 for information related to COLLKEY processing.

Some uses of E61 user exit

Your routine can normalize floating-point control fields or change any other type of control field in any way that you desire. You need to be familiar with the standard data formats used by the operating system before modifying control fields.

If you want to modify the collating sequence of EBCDIC data, for example, to permit the alphabetic collating of national characters, you can do so without the need for an E61 user exit routine by using the ALTSEQ control statement (as described in "ALTSEQ control statement" on page 84).

[18] With a conventional merge or a tape work data set sort, control fields for which E is specified are treated as binary byte format regardless of the actual format(s) specified.

Information DFSORT passes to your routine at E61 user exit

DFSORT places the address of a parameter list in general register 1. The list begins on a fullword boundary and is three fullwords long. The parameter list for the E61 user exit is as follows:

Byte 1	Byte 2	Byte 3	Byte 4
00	00	00	Control Field No.
00	Address of Control Field Image		
Not Used		Control Field Length	

The control field length allows you to write a more generalized modification routine.

To alter the control field, change the control field image at the indicated address (changing the address itself will have no effect).

The control field number is relative to all fields in the SORT or MERGE statement. For example, if you specify:

```
SORT FIELDS=(4,2,CH,A,8,10,CH,E,25,2,BI,E)
```

field numbers 2 and 3 will be passed to user exit E61.

For all fields except binary, the total number of bytes DFSORT passes to your routine is equal to the length specified in the *m* parameter of the SORT or MERGE statement.

All binary fields passed to your routine contain a whole number of bytes; all bytes that contain *any bits* of the control field are passed. If the control field is longer than 256 bytes, DFSORT splits it into fields of 256 bytes each and passes them one at a time to your routine.

Your routine cannot physically change the length of the control field. If you must increase the length for collating purposes, you must previously specify that length in the *m* parameter of the SORT or MERGE statement. If you must shorten the control field, you must pad it to the specified length before returning it to DFSORT. Your routine must return the field to DFSORT with the same number of bytes that it contained when your routine was entered.

When user exit E61 is used, records are always ordered into ascending sequence. If you need some other sequence, you can modify the fields further; for example, if after carrying out your planned modification for a binary control field, and before handing back control to DFSORT, you reverse all bits, the field is, in effect, collated in descending order as illustrated by the E61 example in Figure 17 on page 499.

Note that if E61 is used to resolve ASCII collating for special alphabetic characters, substituted characters must be in EBCDIC, but the sequencing depends upon the byte value of the ASCII translation for the substituted character.

Assembler user exit routines (output phase user exits)

You can use these program user exits located in the DFSORT output phase:

- E31
- E32
- E35
- E37
- E38
- E39

The functions of these user exits are discussed in sequence.

E31 user exit: opening data sets/initializing routines

You might use routines at this user exit to open data sets needed by your other routines in the output phase or to initialize your other routines. Return codes are not used.

Note: To avoid special linkage editor requirements (see "Summary of rules for user exit routines" on page 474), you can include these functions in your E35 user exit rather than in a separate E31 routine.

E32 user exit: handling input to a merge only

This user exit can be used only in a merge operation invoked from a program and cannot be specified on the MODS statement. When an E32 user exit is activated, it must supply all input to the merge. DFSORT ignores SORTIN**nn** data sets when an E32 user exit is used.

You must indicate the number of input files you want to merge using either (1) the FILES=n option on the MERGE control statement, or (2) the X'04' entry in the 24-bit parameter list. Your E32 user exit routine must insert records for these files as DFSORT requests them.

If input is variable-length records, you must be sure the beginning of each record contains a 4-byte RDW before merged. The format of an RDW is described in *z/OS DFSMS Macro Instructions for Data Sets* (Alternatively, you can declare the records as fixed-length and pad them to the maximum length.)

See Figure 10 on page 468 for logic flow details.

Information DFSORT passes to your routine at E32 user exit

Your E32 user exit routine is entered each time the merge program requires a new input record. DFSORT passes three fields to your routine:

- **The increment of the next file to be used for input**. The file increment is 0,4,8,...,N–4, where N is four times the number of input files. Thus, the increment 0 (zero) represents the first input file, 4 the second file, 8 the third, and so on.
- **The address of the next input record**. Your routine must provide a separate input buffer for each input file used. An input buffer containing the first record for a file must not be altered until you have passed the first record from each file to DFSORT.
- **The user exit address constant**. If you invoked DFSORT with a user exit address constant in the parameter list, the address constant is passed to your E32 user exit the first time it is entered. This address constant can be changed by your E32 user exit any time it is entered; the address constant is passed along on subsequent entries to your E32 user exit and E35 user exit. For example, you can obtain a dynamic storage area, use it in your E32 user exit, and pass its address to your E35 user exit.

Note: The user exit address constant must not be used for a Conventional merge application.

Which E32 parameter list will be used

For the E32 exit, DFSORT supports both a 32-bit parameter list and a 64-bit parameter list. If you want DFSORT to use the 64-bit parameter list for your E32, you must:

- Call DFSORT with the 64-bit invocation list
- Set on bit 4 of byte 9 in the 64-bit invocation list

Otherwise, DFSORT will use the 32-bit parameter list for your E32.

See Chapter 6, "Invoking DFSORT from a program," on page 517 for details of using the 64-bit invocation parameter list.

32-bit E32 parameter list

In 32-bit general register 1, DFSORT places the address of a parameter list that contains the file increment, the record address and the user address constant. The list is three fullwords long and begins on a fullword boundary. The format of the parameter list is:

Table 73. 32-bit E32 User Exit Parameter List

32-bit E32 User Exit Parameter ListLocation	Contents
Bytes 1 through 4	Increment of next file to be used for input
Bytes 5 through 8	Address of next input record
Bytes 9 through 12	User exit address constant

DFSORT provides a 72-byte Format 0 save area pointed to by 32-bit register 13 in which the exit can save the 32-bit registers

Before returning control to DFSORT, you must:

- Place the address of the next input record from the requested input file in the second word of the parameter list
- Put the return code in 32-bit register 15.

64-bit E32 parameter list

In 64-bit general register 1, DFSORT places the address of a parameter list that contains the file increment, the record address and the user address constant. The list is three doublewords long and begins on a doubleword boundary. The format of the parameter list is:

Table 74. 64-bit E32 User Exit Parameter List

64-bit E32 User Exit Parameter List Location	Contents	Description
Bytes 1 through 8	X'00000000'	Increment of next file to be used for input
Bytes 9 through 16	Address of next input record	
Bytes 17 through 24	X'00000000'	User exit address constant
Bytes 25 through 32	X'00000000'	Reserved

DFSORT provides a 144-byte Format 4 save area pointed to by 64-bit register 13 in which the exit can save the 64-bit registers.

The E32 should expect and use 64-bit addresses.

Before returning control to DFSORT, you must:

- Place the address of the next input record from the requested input file in the second doubleword of the parameter list.
- Put the return code in 32-bit register 15.

The DFSORT target library, SICEUSER, contains a mapping macro called ICEPL64, which provides a separate Assembler DSECT for the E32 64-bit parameter list.

E32 return codes

Your E32 routine must pass a return code to DFSORT. Following are the return codes for the E32 user exit:

Return Code
Description

08 (X'08')
End of input for requested file

12 (X'0C')
Insert record

16 (X'10')
Terminate DFSORT

8: End of input for requested file
DFSORT continues to return control to the user routine until it receives a return code of 8 for every input file. After that, the user exit is not used again during the DFSORT application. You need not place an address in the second field of the parameter list when you return with a return code of 8.

12: Insert Record
To add a record from the requested input file, place the address of the next record in the parameter list and return to DFSORT with a return code of 12. *DFSORT keeps returning to your routine until you pass a return code of 8 for every input file*.

16: Terminate DFSORT
If you want to terminate DFSORT, return with a code of 16. DFSORT then returns to its calling program with a return code of 16.

E35 user exit: changing records

If you write your E35 user exit in COBOL, see "COBOL user exit routines" on page 499 and "COBOL E35 user exit: changing records" on page 506.

The EXITCK option affects the way DFSORT interprets certain return codes from user exit E35. To avoid ambiguity, this section assumes that the IBM default, EXITCK=STRONG, was selected at your site. For complete details of the meaning of E35 return codes in various situations with EXITCK=STRONG and EXITCK=WEAK, see "E15/E35 return codes and EXITCK" on page 512.

DFSORT enters the E35 user exit routine each time it prepares to place a record in the output area.

See Figure 10 on page 468 for logic flow details.

Some uses for the E35 user exit are:

- Adding records for output data sets
- Omitting records for output data sets
- Changing records for output data sets

Note:

1. If your E35 user exit is processing variable-length records, include a 4-byte RDW at the beginning of each record you change or insert, before you pass it back to DFSORT. The format of an RDW is described in *z/OS DFSMS Using Data Sets* or *System Programming Reference*. (Alternatively, you can pad records to the maximum length and process them as fixed-length.)
2. DFSORT uses the specified or defaulted value for L3 in the RECORD statement to determine the length of the records your E35 user exit passes back to DFSORT. For fixed-length records, be sure that the length of each record your E35 user exit changes or inserts corresponds to the specified or defaulted L3 value. For variable-length records, be sure that the RDW of each record your E35 user exit changes or inserts indicates a length that is less than or equal to the specified or defaulted L3 value. Unwanted truncation or abends may occur if DFSORT uses the wrong length for the records passed to it by your E35 user exit.

   For details of the L3 value, see "RECORD control statement" on page 418.
3. If you use the E35 user exit to dispose of all your output records, you can omit the SORTOUT DD statement.
4. If you invoke DFSORT from a program and you pass the address of your E35 user exit in the parameter list:
   - DFSORT ignores the SORTOUT data set (but not any OUTFIL data sets).
   - DFSORT terminates if you specify E35 in a MODS statement.
5. If you omit the SORTOUT DD statement or it is ignored, and you do not specify any OUTFIL data sets, your E35 user exit routine must dispose of each output record and return to DFSORT with a return

code of 4. When DFSORT returns to your routine after you have disposed of the last record, return to DFSORT with a return code of 8 to indicate"do not return."

6. Remember that if input records are variable-length from a VSAM data set, they will have been prefixed by a 4-byte RDW.
7. After records have been put into the output area, their lengths cannot be increased.
8. For a merge application, records deleted by an E35 user exit routine are not sequence-checked. If you use an E35 user exit routine without an output data set, sequence checking is not performed. In this case, you must ensure that the records are sequenced correctly.

Information DFSORT passes to your routine at E35 user exit

Your E35 user exit routine is entered each time DFSORT prepares to place a record (including the first record) in the output area. DFSORT passes three fields to your routine:

- **The address of the record leaving DFSORT**, which usually follows the record in the output area. End of input is reached when there are no more records to pass to your E35 user exit; DFSORT indicates end of input by setting this address to zero before entering your E35 user exit.

  After end of input is reached, DFSORT continues to enter your user exit routine until a return code of 8 is passed back.

  Your E35 user exit must not change the address of the record leaving DFSORT.
- **The address of a record in the output area** is zero the first time your routine is entered because there is no record in the output area at that time. It remains zero provided you pass a return code of 4 (delete record) to DFSORT.

  Note: If the record pointed to is variable-length, it has an RDW at this point even if output is to a VSAM data set.
- **The user exit address constant** is passed to your user exit exactly as it was set by your E15 or E32 user exit or invoking program's parameter list.

Note: The user exit address constant must not be used for a Conventional merge or tape work data set sort application.

Block Support for E35

This support provides a mechanism to transfer a block of records or single record for all DFSORT functions (SORT, COPY and MERGE) for both fixed length records (FLR) and variable length records (VLR). Transferring block of records will significantly reduce the number of calls of the user exit, and will reduce the transfer of records between user's storage and DFSORT storage. Furthermore, this block support does not affect the existing support of single records interface. In other words in one run it is possible to combine the use of a block of records processing with single record processing. Using the 64-bit parameter list for E35 users can now

- Output a block of records from DFSORT to the E35 exit.
- Output a single record from DFSORT to the E35 exit.
- Output a block of records and single record from DFSORT to the E35 exit in one run.

The E35 Block List contains the information on the number of blocks, the sizes of blocks, and addresses of blocks where DFSORT will accumulate records for transfer to the E35 exit.

Table 75. Block list parameters for E35 exit

Block list parameters for E35 exit Location	Contents	Description
Bytes 1 through 8	X'00000000'	Address of user block list area
Bytes 9 through 16	X'00000000'	Maximum length of block of records for block list type 2

Table 75. Block list parameters for E35 exit (continued)

Block list parameters for E35 exit Location	Contents	Description
Bytes 17 through 24	X'00000000'	Maximum number of blocks for block list type 2 or Maximum number of records for block list type 1
Bytes 25 through 28	X'00000000'	Maximum length of record
Bytes 29 through 29	X'00'	Format of record (F =FLR V=VLR)
Bytes 30 through 30	X'00'	Block List Type **Note:** 1 = Single record 2 = Block of records
Bytes 31 through 32	X'00'	Reserved

See "Notes" under “Block Support for E15.” on page 478 for complete details of the parameter lists and restrictions.

Which E35 parameter list will be used

For the E35 exit, DFSORT supports both a 32-bit parameter list and a 64-bit parameter list. If you want DFSORT to use the 64-bit parameter list for your E35, you must call DFSORT with the 64-bit invocation list, and:

- use a MODS control statement for E35 with the N64 parameter or
- set on bit 5 of byte 9 in the 64-bit invocation list

Otherwise, DFSORT will use the 32-bit parameter list for your E35.

See Chapter 6, “Invoking DFSORT from a program,” on page 517 for details of using the 64-bit invocation parameter list.

32-bit E35 parameter list

In 32-bit general register 1, DFSORT places the address of a parameter list that contains the two record addresses and the user address constant. The list is three fullwords long and begins on a fullword boundary. The format of the parameter list is:

Table 76. 32-bit E35 User Exit Parameter List

32-bit E35 User Exit Parameter ListLocation	Contents
Bytes 1 through 4	Address of record leaving DFSORT
Bytes 5 through 8	Address of record in output area
Bytes 9 through 12	User exit address constant

DFSORT provides a 72-byte Format 0 save area pointed to by 32-bit register 13 in which the exit can save the 32-bit registers.

Before returning control to DFSORT, you must:

- Place the address of the record in 32-bit register 1.

- Put the return code in 32-bit register 15.

64-bit E35 parameter list

In 64-bit general register 1, DFSORT places the address of a parameter list that contains the two record addresses and the user address constant. The list is four doublewords long and begins on a doubleword boundary. The format of the parameter list is:

Table 77. 64-bit E35 User Exit Parameter List

64-bit E35 User Exit Parameter ListLocation	Contents	Description
Bytes 1 through 8	X'00000000'	Address of record leaving DFSORT Or Address of E35 block list
Bytes 9 through 16	X'00000000'	Address of record in output area
Bytes 17 through 24	X'00000000'	User exit address constant
Bytes 25 through 32	X'00000000'	Address of user block list parameter area

DFSORT provides a 144-byte Format 4 save area pointed to by 64-bit register 13 in which the exit can save the 64-bit registers.

The E35 should expect and use 64-bit addresses.

Before returning control to DFSORT, you must:

- Place the address of the single record or the block list address in 64-bit register 1. This must be a 64-bit address, a clean 31-bit address or a clean 24-bit address.
- Put the return code in 64-bit register 15.

The DFSORT target library, SICEUSER, contains a mapping macro called ICEPL64, which provides a separate Assembler DSECT for the E35 64-bit parameter list.

E35 return codes

Your E35 routine must pass a return code to DFSORT. Following are the return codes for the E35 user exit:

Return Code
Description

00 (X'00')
No Action/Record Altered

04 (X'04')
Delete Single record or Block of records

08 (X'08')
Do Not Return

12 (X'0C')
Insert Single Record

16 (X'10')
Terminate DFSORT

24 (X'18')
Receive block of records

0: No Action
If you want DFSORT to retain the record unchanged, load the address of the record leaving DFSORT in general register 1 and return to DFSORT with a return code of 0 (zero).

0: Record Altered
If you want to change the record before having it placed in the output data set, move the record to a work area, make the change, load the address of the modified record into general register 1, and return to DFSORT with a return code of 0 (zero).

Note:
Return code 0 is not valid for block support to add/alter block of records from E35 exit.

4: Delete Record
Your routine can delete the single record or block of records leaving DFSORT by returning to DFSORT with a return code of 4. You need not place an address in general register 1.

8: Do Not Return
DFSORT keeps returning to your routine until you pass a return code of 8. After that, the user exit is not used again during the DFSORT application. When you return with a return code of 8, you need not place an address in general register 1. *Unless you are inserting records after the end of the data set, you must pass a return code of 8 when DFSORT indicates the end of the data set*. This is done by passing a zero as the address of the record leaving DFSORT.

If you do not have an output data set and would usually return with a return code of 8 before EOF, you can avoid getting the ICE025A message by specifying NOCHECK on the OPTION control statement (if installation option CHECK=NO had not already been specified).

If your user exit routine passes a return code of 8 to DFSORT when input records still remain to be processed, the remaining records are processed by DFSORT, but are *not* passed to your user exit.

12: Insert Record
To add an output record ahead of the record leaving DFSORT, place the address of the new record in general register 1 and return to DFSORT with a return code of 12. DFSORT returns to your routine with the same address as passed on the previous call to the user exit for the record leaving DFSORT. DFSORT passes the address of the inserted record as the address of the record in the output area. You can make more insertions at that point, or delete the record leaving DFSORT.

DFSORT does not perform sequence checking for disk work data set sorts. For tape work data set sorts, DFSORT does not perform sequence checking on records that you insert unless you delete the record leaving DFSORT and insert a record to replace it. *DFSORT keeps returning to your routine until you pass a return code of 8*.

Note:

Return code 12 is not valid for block support to insert block of records from E35 exit. Set the return code to 24.

16: Terminate DFSORT
If you want to terminate DFSORT, return with a code of 16. DFSORT then returns to its calling program or to the system with a return code of 16.

24: Receive Block of records
To add a block of records ahead of the record leaving DFSORT, place the address of the new block of records in general register 1 and return to DFSORT with a return code of 24. DFSORT keeps returning to your routine until you pass a return code of 8.

See “E15/E35 return codes and EXITCK” on page 512 for complete details of the meanings of return codes in various situations.

Summing records at E35 user exit

You can use the SUM control statement to sum records. However, you can sum records for output by changing the record in the output area and then, if you want, by deleting the record leaving DFSORT. DFSORT returns to your routine with the address of a new record leaving DFSORT, and the same record remains in the output area, so that you can continue summing. If you do not delete the record leaving

DFSORT, that record is added to the output area, and its address replaces the address of the previous record in the output area. DFSORT returns with the address of a new record leaving DFSORT.

Storage usage for E35 user exit

DFSORT obtains storage (using GETMAIN or STORAGE OBTAIN) for the parameter list and the records it passes to your E35 user exit routine. You must not attempt to modify or free the storage obtained by DFSORT.

If you need to obtain storage for use by your E35 user exit routine, such as to pass altered records to DFSORT, you can use the following strategy:

1. The first time your exit is called, obtain the storage you need
2. Use the storage you obtained each time your exit is called
3. Free the storage before you pass back return code 8 to DFSORT.

Note: When you obtain your storage you can save its address in the user exit address constant and restore it on each subsequent call to your exit.

E37 user exit: closing data sets

Your E37 user exit routine is entered once at the end of the output phase. It can be used to close data sets used by your other routines in the phase or to perform any housekeeping functions for your routines.

Note: To avoid special linkage editor requirements (see “Summary of rules for user exit routines” on page 474), you can include these functions in your E35 user exit rather than in a separate E37 user exit.

E38 user exit: handling input data sets

The routine here is the same as for E18. If the Blockset or Peerage/Vale technique is selected, I/O error conditions cannot be handled through the E38 user exit.

Using E38 user exit with VSAM

This user exit can be used during a merge or copy to insert VSAM passwords into VSAM input ACBs and to perform various VSAM user exit functions. The following example shows code your program can use to return control to DFSORT.

```
       ENTRY   E38
         .
         .
E38      LA      1,PARMLST
         RETURN
         CNOP    0,4
PARMLST  DS      0H
         DC      X'05'
         DC      AL3(VSAMEXL)
         DC      X'06'
         DC      AL3(PWDLST)
         DC      A(0)
         .
         .
VSAMEXL  EXLST   SYNAD=USYNAD,LERAD=ULERAD
PWDLST   DC      H'2'
         DC      CL8'SORTIN01'    SORTIN01 DDNAME
         DC      CL8'INPASS1'     SORTIN01 PASSWORD
         DC      CL8'SORTIN02'    SORTIN02 DDNAME
         DC      CL8'INPASS2'     SORTIN02 PASSWORD
USYNAD   ...                      VSAM SYNCH ERROR RTN
ULERAD   ...                      VSAM LOGIC ERROR RTN
```

Figure 12. E38 User Exit Example

E39 user exit: handling output data sets

Your E39 user exit routine is entered for the SORTOUT data set, but not for OUTFIL data sets.

Using E39 user exit with QSAM/BSAM

The technique is the same as for E19 for QSAM/BSAM. See "E19 user exit: handling output to work data sets" on page 485 for details.

Using E39 user exit with VSAM

For VSAM, this user exit can be used to insert VSAM passwords intothe VSAM SORTOUT ACB and to perform various VSAM user exit functions. The example that follows shows code your program can use to return control to DFSORT.

```
         ENTRY   E39
         .
         .
E39      LA      1,PARMLST
         RETURN
         CNOP    0,4
PARMLST  DS      0H
         DC      X'05'
         DC      AL3(VSAMEXL)
         DC      X'06'
         DC      AL3(PWDLST)
         DC      A(0)
         .
         .
VSAMEXL  EXLST   SYNAD=USYNAD,LERAD=ULERAD
PWDLST   DC      H'1'
         DC      CL8'SORTOUT'     SORTOUT DDNAME
         DC      CL8'OUTPASS'     SORTOUT PASSWORD
USYNAD   ...                      VSAM SYNCH ERROR RTN
ULERAD   ...                      VSAM LOGIC ERROR RTN
```

Figure 13. E39 User Exit Example

Sample E15 and E35 routines using the 64-bit parameter lists

"Example 15. Sort with 64-bit parameter lists, E15, E35 and OUTFIL" on page 785 in Chapter 11, "Examples of DFSORT job streams," on page 765 shows, in assembler language, how to use the 64-bit parameter lists for E15 and E35 exit routines.

Sample routines written in assembler using the 32-bit parameter lists

This section provides some sample program user exits written in assembler using the 32-bit parameter lists.

E15 user exit: altering record length

This routine changes the variable-length input records making them all the same length.

```
E15      CSECT
* IF A RECORD IS GREATER THAN 204 BYTES, TRUNCATE IT TO 204 BYTES.
* IF A RECORD IS LESS THAN 204 BYTES, PAD IT OUT TO 204 BYTES.
* ALL OF THE RESULTING RECORDS WILL BE 204 BYTES LONG
* (4 BYTES FOR THE RDW AND 200 BYTES OF DATA).
         USING E15,12             SHOW BASE REG
         STM   14,12,12(13)       SAVE ALL REGS EXCEPT 13
         LA    12,0(0,15)         SET BASE REG
         ST    13,SAVE15+4        SAVE BACKWARD POINTER
         LA    14,SAVE15          SET FORWARD POINTER
         ST    14,8(13)           IN SAVE AREA
         LR    13,14              SET OUR SAVE AREA
         LR    2,1                SAVE PARM LIST POINTER
         L     3,0(,2)            LOAD ADDR OF RECORD
         LTR   3,3                EOF
         BZ    EOF                YES - DO NOT RETURN
         LH    4,0(,3)            GET RDW
         CH    4,CON204           IS RDW EQ 204
         BE    ACCEPT             YES-ACCEPT IT
         BL    PAD                LESS THAN 204-PAD
         LH    4,CON204           LIMIT LENGTH TO 204
         B     TRUNC              MORE THAN 204-TRUNCATE
PAD      DS    0H                 PAD OR TRUNCATE
         MVI   DATA,X'00'         ZERO OUT THE BUFFER
         MVC   DATA+1(199),DATA
TRUNC    DS    0H                 PAD OR TRUNCATE
         BCTR  4,0                DECREASE RDW FOR EXECUTE
         EX    4,MVPAD            MOVE RECORD INTO PAD/TRUNCATE BUFFER
         MVC   NEWRDW(2),CON204   SET NEW RDW TO 204
         LA    3,BUFFER           POINT TO PADDED/TRUNCATED RECORD
ACCEPT   DS    0H
         SR    15,15              SET RC=0
         LR    1,3                SET RECORD POINTER
         B     GOBACK
EOF      LA    15,8               EOF - SET RC=8
GOBACK   L     13,4(,13)
         L     14,12(,13)
         LM    2,12,28(13)        RESTORE REGS
         BR    14                 RETURN
MVPAD    MVC   BUFFER(*-*),0(3)   FOR EXECUTE
SAVE15   DS    18F
CON204   DC    H'204'
BUFFER   DS    0H
NEWRDW   DS    H                  NEW RDW OF 204
         DC    H'0'
DATA     DC    XL200'00'          BUFFER FOR PADDING/TRUNCATING
         END
```

Figure 14. E15 User Exit Example

E16 user exit: sorting current records when NMAX is exceeded

This routine tells DFSORT that, when DFSORT issues the message "NMAX EXCEEDED", it must sort only the records already read in.

```
E16    CSECT
       LA      15,0      SET RETURN CODE
       BR      14
       END
```

Figure 15. E16 User Exit Example

E35 user exit: altering record length

This routine changes the variable-length output records making them all the same length.

```
E35     CSECT
* IF A RECORD IS GREATER THAN 204 BYTES, TRUNCATE IT TO 204 BYTES.
* IF A RECORD IS LESS THAN 204 BYTES, PAD IT OUT TO 204 BYTES.
* ALL OF THE RESULTING RECORDS WILL BE 204 BYTES LONG
* (4 BYTES FOR THE RDW AND 200 BYTES OF DATA).
         USING E35,12             SHOW BASE REG
         STM   14,12,12(13)       SAVE ALL REGS EXCEPT 13
         LA    12,0(0,15)         SET BASE REG
         ST    13,SAVE15+4        SAVE BACKWARD POINTER
         LA    14,SAVE15          SET FORWARD POINTER
         ST    14,8(13)           IN SAVE AREA
         LR    13,14              SET OUR SAVE AREA
         LR    2,1                SAVE PARM LIST POINTER
         L     3,0(,2)            LOAD ADDR OF RECORD
         LTR   3,3                EOF
         BZ    EOF                YES - DO NOT RETURN
         LH    4,0(,3)            GET RDW
         CH    4,CON204           IS RDW EQ 204
         BE    ACCEPT             YES-ACCEPT IT
         BL    PAD                LESS THAN 204-PAD
         LH    4,CON204           LIMIT LENGTH TO 204
         B     TRUNC              MORE THAN 204-TRUNCATE
PAD      DS    0H                 PAD OR TRUNCATE
         MVI   DATA,X'00'         ZERO OUT THE BUFFER
         MVC   DATA+1(199),DATA
TRUNC    DS    0H                 PAD OR TRUNCATE
         BCTR  4,0                DECREASE RDW FOR EXECUTE
         EX    4,MVPAD            MOVE RECORD INTO PAD/TRUNCATE BUFFER
         MVC   NEWRDW(2),CON204   SET NEW RDW TO 204
         LA    3,BUFFER           POINT TO PADDED/TRUNCATED RECORD
ACCEPT   DS    0H
         SR    15,15              SET RC=0
         LR    1,3                SET RECORD POINTER
         B     GOBACK
EOF      LA    15,8               EOF - SET RC=8
GOBACK   L     13,4(,13)
         L     14,12(,13)
         LM    2,12,28(13)        RESTORE REGS
         BR    14                 RETURN
MVPAD    MVC   BUFFER(*-*),0(3)   FOR EXECUTE
SAVE15   DS    18F
CON204   DC    H'204'
BUFFER   DS    0H
NEWRDW   DS    H                  NEW RDW OF 204
         DC    H'0'
DATA     DC    XL200'00'          BUFFER FOR PADDING/TRUNCATING
         END
```

Figure 16. E35 User Exit Example

E61 user exit: altering control fields

This routine can be used to change the order of binary control fields passed to it (that is, those for which 'E' is specified) from ascending order to descending order.

```
* E61 PARAMETER LIST DSECT
PARML    DSECT
         DS    3C
PARMNUM  DS    C   CONTROL FIELD NUMBER
PARMPTR  DS    A   ADDRESS OF CONTROL FIELD
         DS    2C
PARMLEN  DS    H   CONTROL FIELD LENGTH
*
E61REV   CSECT
* CHANGE THE ORDER OF EACH  CONTROL FIELD PASSED TO THIS ROUTINE
* FROM ASCENDING TO DESCENDING BY REVERSING ALL OF THE BITS.
* ASSUMES THAT ONLY BI CONTROL FIELDS ARE PASSED.
         USING E61REV,12          SHOW BASE REG
         STM   14,12,12(13)       SAVE ALL REGS EXCEPT R13
         LA    12,0(0,15)         SET BASE REG
         ST    13,SAVE61+4        SAVE BACKWARD POINTER
         LA    14,SAVE61          SET FORWARD POINTER
         ST    14,8(13)           IN SAVE AREA
         LR    13,14              SET OUR SAVE AREA
         LR    3,1                SET PARM LIST POINTER
         USING PARML,3
         L     4,PARMPTR         GET POINTER TO CONTROL FIELD IMAGE
         LH    5,PARMLEN         GET LENGTH OF CONTROL FIELD
         BCTR  5,0               SUBTRACT 1 FOR EXECUTE
         EX    5,REVCF           CHANGE FROM ASCENDING TO DESCENDING
GOBACK   L     13,4(,13)
         LM    14,12,12(13)       RESTORE REGS
         BR    14                 RETURN
REVCF    XC    0(*-*,4),REVFF    REVERSE CONTROL FIELD BITS
SAVE61   DS    18F
REVFF    DC    256X'FF'
         LTORG
         END
```

Figure 17. E61 User Exit Example

COBOL user exit routines

You can perform the same tasks with E15 and E35 user exit routines written in COBOL that you can perform with E15 and E35 user exit routines written in assembler. However, COBOL routines differ from assembler routines in the way they pass information between themselves and DFSORT.

- COBOL routines must pass information through fields described in the LINKAGE SECTION of the DATA DIVISION. Assembler uses general register 1 and pointers in a parameter list.
- COBOL routines must use RETURN-CODE, a COBOL special register. Assembler uses register 15 for the return code.
- COBOL routines must use return code 20 to alter or replace a record. Assembler uses return code 0.
- COBOL routines can use the user exit area for E15/E35 communication. Assembler uses the user address constant.

COBOL user exit requirements

The following rules apply to COBOL user exits. Failure to observe these COBOL user exit rules can result in termination or unpredictable results.

- User exits written in COBOL must not use STOP RUN statements. To return to DFSORT, use the GOBACK statement.
- DFSORT may use high-half of the general registers. If the user exit routine is compiled with COBOL V5 or later releases, the COBOL option HGPR(PRESERVE) must be used. This is the default setting of this option.
- VS COBOL II user exits must be compiled with the RES and RENT compiler options.
- If a user exit contains a DISPLAY statement, the DFSORT messages normally written to SYSOUT must be directed to another data set using the MSGDDN parameter. For DISPLAY statements, COBOL also writes to SYSOUT. The messages to SYSOUT can, therefore, be lost because of interleaving of output.

  An alternative is to direct the COBOL output to another data set by using the OUTDD compiler option.
- COBOL user exits must not contain a SORT or a MERGE verb.

- When coding the MODS control statement to describe a COBOL user exit, use C for the fourth parameter. This instructs DFSORT to build the correct parameter list.
- If invoking DFSORT from a COBOL program, you can use a COBOL E15 if the FASTSRT option is in effect for input and a COBOL E35 if FASTSRT is in effect for output.
- COBOL library routines in Language Environment must be available at run time.

COBOL requirements for copy processing

For copy processing, all sort requirements apply except for the following restrictions:

- When you directly invoke DFSORT, you can use *either* a separately compiled COBOL E15 user exit *or* a separately compiled COBOL E35 user exit, but not both.
- When you invoke DFSORT from a COBOL program, the following limitations apply when FASTSRT is in effect for:
  - Input only: You can use a separately compiled E15 user exit, but not a separately compiled E35 user exit.
  - Output only: You can use a separately compiled E35 user exit, but not a separately compiled E15 user exit.
  - Input and output: You can use *either* a separately compiled E15 *or* a separately compiled E35, but not both together.

  If separately compiled E15 and E35 user exits are found together, DFSORT copy processing terminates. Message ICE161A is issued.

COBOL storage requirements

If you are running COBOL user exits compiled with the RES compiler option, make sure that you have enough storage available for the COBOL library subroutines. (This does not apply if the library has been installed resident.)

Besides the minimum DFSORT main storage requirements, you need an additional 1200KB of storage in your REGION to run Language Environment.

Under certain conditions, DFSORT can use all the storage in your REGION below 16MB virtual, thus leaving no room to load the COBOL library subroutines required during processing of your user exit.

Main storage is available above 16MB virtual unless the TMAXLIM or SIZE/MAINSIZE options specify an extremely high value (for example, your system limit for main storage above 16MB virtual). In that case, you can use the ARESALL or ARESINV option to release storage.

During processing, the actual amount of storage required for the COBOL library subroutines depends on the functions performed in the COBOL user exit. You must add a minimum of 20KB to the size of the user exit. If the user exit does I/O, additional storage must be reserved for the I/O buffers. Additional storage for buffers is specified by the **m** parameter on the MODS statement.

When SIZE/MAINSIZE=MAX is in effect, an alternative way to release storage is to use the RESALL or RESINV option.

Note: You might need to release an additional 70KB of storage when you are:

- Calling both E15 and E35 user exits
- Running with nonresident library subroutines

This can be done by adding 70KB more to one of the following:

- The m parameter of the MODS statement for the E35 user exit (m = E35 user exit size + 20KB + 70KB)
- The RESALL option when SIZE/MAINSIZE=MAX is in effect.

COBOL user exit routines (input phase user exit)

COBOL E15 user exit: passing or changing records for sort

The EXITCK option affects the way DFSORT interprets certain return codes from user exit E15. To avoid ambiguity, this section assumes that the IBM default, EXITCK=STRONG, was selected at your site. For complete information about E15 return codes in various situations with EXITCK=STRONG and EXITCK=WEAK, see “E15/E35 return codes and EXITCK” on page 512.

DFSORT enters the E15 user exit routine each time a new record is brought into the input phase. DFSORT continues to enter E15 (even when there are no input records) until the user exit tells DFSORT, with a return code of 8, not to return.

See Figure 10 on page 468 for logic flow details.

Some uses for the E15 user exit are:

- Adding records to an input data set
- Passing an entire input data set to DFSORT
- Deleting records from an input data set
- Changing records in an input data set.

Note:

1. If both E15 and E35 user exits are used, they must be in the same version of COBOL.
2. If you use the E15 user exit to pass all your records to DFSORT, you can omit the SORTIN DD statement, in which case you must include a RECORD statement in the program control statements.
3. If you omit the SORTIN DD statement, all input records are passed to DFSORT through your COBOL E15 user exit. You return to DFSORT with a return code of 12. When DFSORT returns to the E15 user exit after the last record has been passed, you return to DFSORT with a return code of 8 in register 15, which indicates "do not return."
4. DFSORT continues to reenter your E15 user exit until a return code of 8 is received. However, if STOPAFT is in effect, no additional records are inserted to DFSORT after the STOPAFT count is satisfied (even if you pass back a return code of 12).
5. You cannot use dynamic binding or link-editing with a COBOL E15 user exit.

E15 interface with COBOL

Each time the E15 user exit is called, DFSORT supplies the following fields:

- Record flags
- New record
- Length of the new record (for variable-length records)
- Length of user exit area
- User exit area.

When E15 returns to DFSORT, the E15 user exit provides to DFSORT some or all of the fields mentioned in the following. The first field is required; the others can be modified as appropriate.

- RETURN-CODE (assigned by the user exit by setting the COBOL special register RETURN-CODE)
- Return record
- Length of the return record (for VLR)
- Length of user exit area
- User Exit area.

For more information on how these fields are used in a COBOL E15 user exit, see “E15 LINKAGE SECTION fields for fixed-length and variable-length records” on page 503.

Figure 18 on page 502 details the interface to COBOL for the E15 user exit.

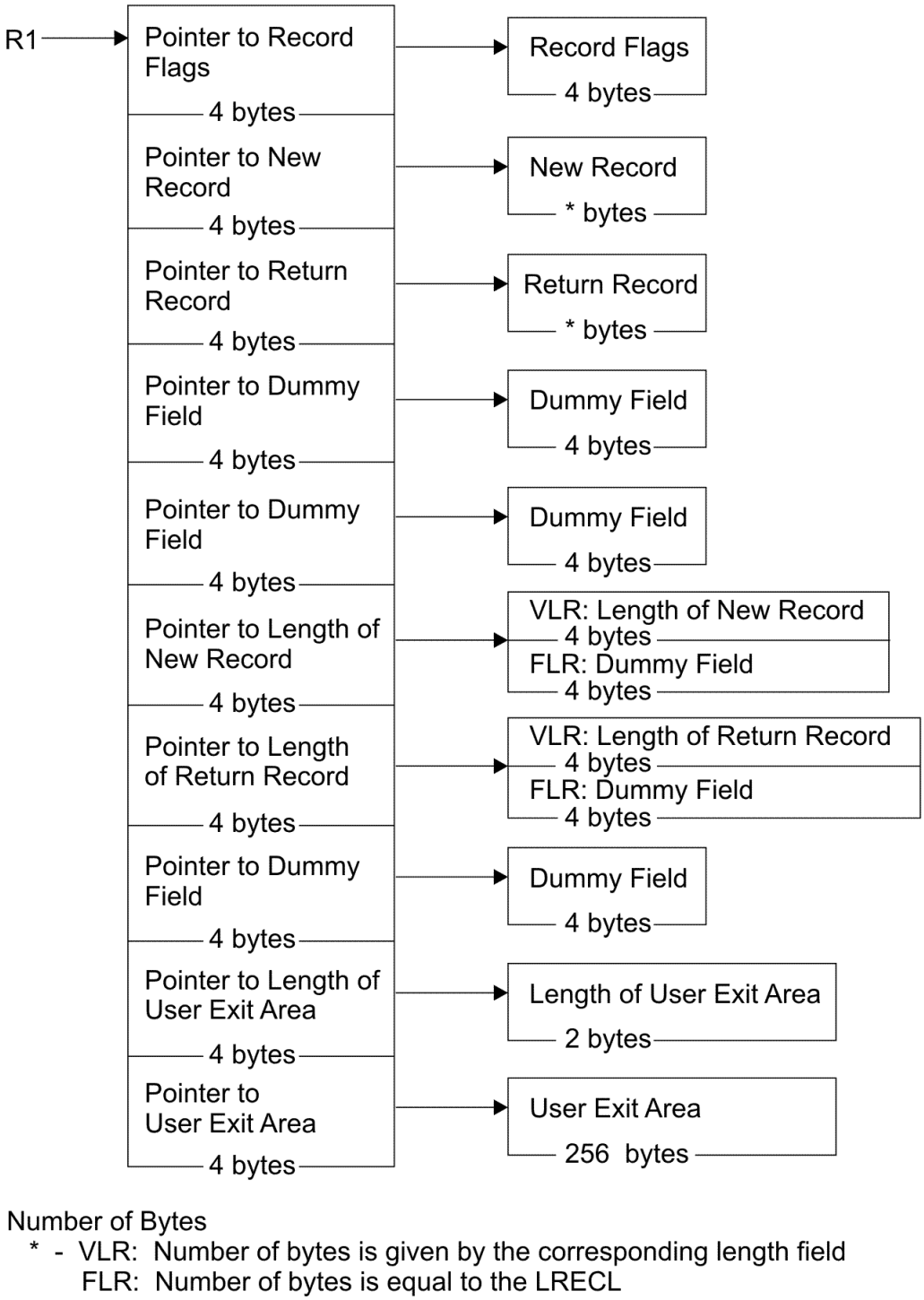


Figure 18. E15 DFSORT Interface with COBOL

E15 LINKAGE SECTION examples

Figure 19 on page 503 is an example of the LINKAGE SECTION code for a fixed-length record (FLR) data set with a logical record length (LRECL) of 100. The example shows the layout of the fields passed to your COBOL routine.

```
LINKAGE SECTION.
01  RECORD-FLAGS       PIC 9(8) BINARY.
    88  FIRST-REC            VALUE 00.
    88  MIDDLE-REC           VALUE 04.
    88  END-REC              VALUE 08.
01  NEW-REC            PIC X(100).
01  RETURN-REC         PIC X(100).
01  UNUSED1            PIC 9(8) BINARY.
01  UNUSED2            PIC 9(8) BINARY.
01  UNUSED3            PIC 9(8) BINARY.
01  UNUSED4            PIC 9(8) BINARY.
01  UNUSED5            PIC 9(8) BINARY.
01  EXITAREA-LEN       PIC 9(4) BINARY.
01  EXITAREA.
    05  EAREA   OCCURS 1 TO 256 TIMES
            DEPENDING ON EXITAREA-LEN   PIC X.
```

Figure 19. LINKAGE SECTION Code Example for E15 (Fixed-Length Records)

Figure 20 on page 503 is an example of the LINKAGE SECTION code for a variable-length record (VLR) data set with a maximum LRECL of 200. The example shows the layout of the fields passed to your COBOL routine.

```
LINKAGE SECTION.
01  RECORD-FLAGS       PIC 9(8) BINARY.
    88  FIRST-REC            VALUE 00.
    88  MIDDLE-REC           VALUE 04.
    88  END-REC              VALUE 08.
01  NEW-REC.
    05 NREC OCCURS 1 TO 200 TIMES
            DEPENDING ON NEW-REC-LEN      PIC X.
01  RETURN-REC.
    05 RREC OCCURS 1 TO 200 TIMES
            DEPENDING ON RETURN-REC-LEN   PIC X.
01  UNUSED1            PIC 9(8) BINARY.
01  UNUSED2            PIC 9(8) BINARY.
01  NEW-REC-LEN        PIC 9(8) BINARY.
01  RETURN-REC-LEN     PIC 9(8) BINARY.
01  UNUSED3            PIC 9(8) BINARY.
01  EXITAREA-LEN       PIC 9(4) BINARY.
01  EXITAREA.
    05  EAREA   OCCURS 1 TO 256 TIMES
            DEPENDING ON EXITAREA-LEN   PIC X.
```

Figure 20. LINKAGE SECTION Code Example for E15 (Variable-Length Record)

E15 LINKAGE SECTION fields for fixed-length and variable-length records

The fields in the LINKAGE SECTION are used by DFSORT and your routine as stated later in this section. For clarity, the field names from Figure 20 on page 503 have been used.

- To give your COBOL routine the status of the passed records, DFSORT uses the record flags field (RECORD-FLAGS) in the following way:

  0 (FIRST-REC)
  The new record is the first passed record.

  4 (MIDDLE-REC)
  The new record is not the first passed record.

  8 (END-REC)
  All records have been passed to your routine or there were no records to pass.
- DFSORT places the next input record in the new record field (NEW-REC). A VLR does not contain an RDW, but DFSORT places the length of this VLR in the new record length field (NEW-REC-LEN). The value in the NEW-REC-LEN field is the length of the record only and does not include the 4 bytes for the RDW.
- When your routine places an insertion/replacement record in the return record field (RETURN-REC), the VLR must not contain an RDW; your routine must place the length of this record in the return record length field (RETURN-REC-LEN). The value of the RETURN-REC-LEN field is the length of the record only and must not include the 4 bytes for the RDW.

- Each time DFSORT calls your COBOL E15 or COBOL E35 user exit, it passes the user exit a 256-byte user exit area field (EXITAREA). The first time the user exit area field is passed to your COBOL E15 user exit, it contains 256 blanks, and the user exit area length field (EXITAREA-LEN) contains 256.

  Any changes you make to the user exit area field or user exit area length fields are passed back both to your COBOL E15 user exit and your COBOL E35 user exit.

  Note:

  1. Do not set the user exit area length field to more than 256 bytes.
  2. If the data used for input was not created by a COBOL run, you need to know the LRECL defined for your data set. For a VLR, the maximum length of the record defined in your COBOL user exit is 4 bytes less than the LRECL value, because COBOL does not include the RDW as part of the record. (Each VLR begins with an RDW field of 4 bytes. The RDW is not included in the record passed to your COBOL user exit.)
  3. You need to code only up to the last field that your routine actually uses (for example, up to RETURN-REC if you do not use the user exit area).
  4. DFSORT uses the specified or defaulted value for L2 in the RECORD statement to determine the length of the records your E15 user exit passes back to DFSORT. For fixed-length records, be sure that each record your E15 user exit changes or inserts has a length that is equal to the specified or defaulted L2 value. For variable-length records, be sure that each record your E15 user exit changes or inserts has a length that is less than or equal to the specified or defaulted L2 value. Unwanted truncation or abends may occur if DFSORT uses the wrong length for the records passed to it by your E15 user exit.

     For details of the L2 value, see “RECORD control statement” on page 418

     .

E15 return codes

Your COBOL E15 routine must pass a return code to DFSORT in the RETURN-CODE field, a COBOL special register. Following are the return codes for the E15 user exit:

Return Code
Description

00 (X'00')
No Action

04 (X'04')
Delete Record

08 (X'08')
Do Not Return

12 (X'0C')
Insert Record

16 (X'10')
Terminate DFSORT

20 (X'14')
Alter or Replace Record

0: No Action
If you want DFSORT to retain the record unchanged, return with RETURN-CODE set to 0.

4: Delete Record
If you want DFSORT to delete the record, return with RETURN-CODE set to 4.

8: Do Not Return
DFSORT continues to enter your routine until you return with RETURN-CODE set to 8. After that, the user exit is not used again during the DFSORT application. *Unless you are inserting records after the end of the data set, you must set RETURN-CODE to 8 when DFSORT indicates the end of the data set*, which it does by entering your routine with the record flags field set to 8.

If your user exit routine passes a return code of 8 to DFSORT when input records still remain to be processed, the remaining records are processed by DFSORT but are *not* passed to your user exit.

12: Insert Record
If you want DFSORT to add a record before the new record in the input data set:

- Move the insert record to the return record field
- For VLR, move the length to the return record length field (Do not include the 4-byte RDW in this length.)
- Return with RETURN-CODE set to 12.

DFSORT returns to your routine with the same record as before in the new record field, allowing your routine to insert more records or handle the new record.

You can also insert records after the end of the data set. *DFSORT keeps returning to your routine as long as you pass it a RETURN-CODE of 12 and until you return with a RETURN-CODE set to 8*.

16: Terminate DFSORT
If you want to terminate DFSORT, return with RETURN-CODE set to 16. DFSORT then returns to its calling program or to the system with a return code of 16.

20: Alter Record
If you want to change the new record:

- Move the new record to the return record field.
- Change the record in the return record field.
- For VLR records, move the length to the return record length field.
- Return with RETURN-CODE set to 20.

Note: If your routine changes record size, you must indicate the new size on the RECORD statement.

20: Replace Record
If you want to replace the new record:

- Move the replacement record to the return record field.
- For VLR records, move the length to the return record length field. (Do not include the 4-byte RDW in this length.)
- Return with RETURN-CODE set to 20.

See “E15/E35 return codes and EXITCK” on page 512 for complete details of the meanings of return codes in various situations.

E15 procedure division requirements

When coding the PROCEDURE DIVISION, the following requirements must be met:

- To return control to DFSORT, you must use the GOBACK statement.
- In the USING option of the PROCEDURE DIVISION header, you must specify *each* 01-level name in the LINKAGE SECTION. You must specify each name in order up to the last one you plan to use even when you do not use all the 01-level names preceding the header.

  Examples:

  For the FLR example, Figure 19 on page 503, you would code:

```
       PROCEDURE DIVISION USING RECORD-FLAGS, NEW-REC,
                RETURN-REC, UNUSED1, UNUSED2, UNUSED3,
                UNUSED4, UNUSED5, EXITAREA-LEN, EXITAREA.
```

  For the VLR example, Figure 20 on page 503, you would code:

```
       PROCEDURE DIVISION USING RECORD-FLAGS, NEW-REC,
                RETURN-REC, UNUSED1, UNUSED2,
```

```
                NEW-REC-LEN, RETURN-REC-LEN,
                UNUSED3, EXITAREA-LEN, EXITAREA.
```

COBOL user exit routines (output phase user exit)

COBOL E35 user exit: changing records

The EXITCK option affects the way DFSORT interprets certain return codes from user exit E35. To avoid ambiguity, this section assumes that the IBM default, EXITCK=STRONG, was selected at your site. For complete information about E35 return codes in various situations with EXITCK=STRONG and EXITCK=WEAK, see “E15/E35 return codes and EXITCK” on page 512.

DFSORT enters the E35 user exit routine each time it prepares to place a record in the output area.

See Figure 10 on page 468 for logic flow details.

Some uses for the E35 user exit are:

- Adding records for output data sets
- Omitting records for output data sets
- Changing records for output data sets

When DFSORT indicates the end of the data set (record flags field set to 8), you must set RETURN-CODE to 8 (unless you are inserting records after the end of the data set); otherwise, DFSORT continues to enter E35.

Note:

1. If both E15 and E35 user exits are used, they must be in the same version of COBOL.
2. If you use the E35 user exit to dispose of all your output records, you can omit the SORTOUT DD statement.
3. If you omit the SORTOUT DD statement and you do not specify any OUTFIL data sets, your E35 user exit routine must dispose of each output record and return to DFSORT with a return code of 4. When DFSORT returns to your routine after you have disposed of the last record, return to DFSORT with a return code of 8 to indicate "do not return."
4. You cannot use dynamic binding or link-editing with a COBOL E35 user exit.

E35 interface with COBOL

Each time your E35 user exit is called, DFSORT supplies the following fields:

- Record flags
- Record leaving DFSORT
- Length of record leaving DFSORT (for variable-length records)
- Length of user exit area
- User Exit area.

When your E35 user exit returns to DFSORT, the E35 user exit provides to DFSORT some or all of the fields mentioned in the following. The first field is required; the others can be modified as appropriate.

- RETURN-CODE (assigned by the user exit by setting the COBOL special register RETURN-CODE)
- Return record
- Length of return record (for variable-length records)
- Length of user exit area
- User exit area.

For more information on how these fields are used in a COBOL E35 user exit, see "E35 LINKAGE SECTION fields for fixed-length and variable-length records" on page 508.

Figure 21 on page 507 details the interface to COBOL for the E35 user exit.

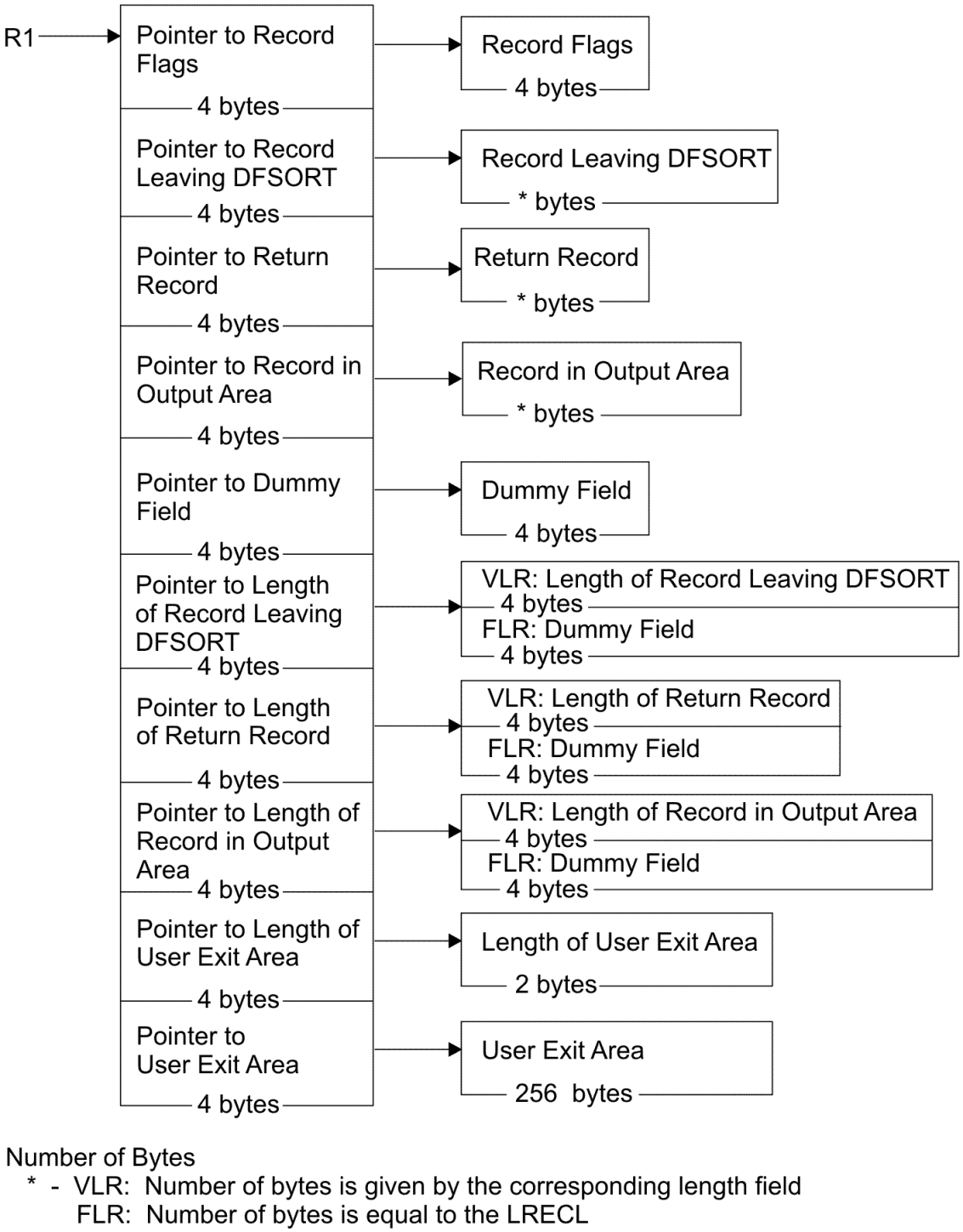


Figure 21. E35 Interface with COBOL

E35 LINKAGE SECTION examples

Figure 22 on page 508 is an example of the LINKAGE SECTION code for a fixed-length record (FLR) data set with a logical record length (LRECL) of 100. The example shows the layout of the fields passed to your COBOL routine.

```
LINKAGE SECTION.
   01  RECORD-FLAGS       PIC 9(8) BINARY.
       88  FIRST-REC            VALUE 00.
       88  MIDDLE-REC           VALUE 04.
       88  END-REC              VALUE 08.
   01  LEAVING-REC        PIC X(100).
   01  RETURN-REC         PIC X(100).
   01  OUTPUT-REC         PIC X(100).
   01  UNUSED1            PIC 9(8) BINARY.
   01  UNUSED2            PIC 9(8) BINARY.
   01  UNUSED3            PIC 9(8) BINARY.
   01  UNUSED4            PIC 9(8) BINARY.
   01  EXITAREA-LEN       PIC 9(4) BINARY.
   01  EXITAREA.
       05  EAREA   OCCURS 1 TO 256 TIMES
              DEPENDING ON EXITAREA-LEN   PIC X.
```

Figure 22. LINKAGE SECTION Code Example for E35 (Fixed-Length Records)

Figure 23 on page 508 is an example of the LINKAGE SECTION code for a variable-length record (VLR) data set with a maximum LRECL of 200. The example shows the layout of the fields passed to your COBOL routine.

```
   LINKAGE SECTION.
   01  RECORD-FLAGS       PIC 9(8) BINARY.
       88  FIRST-REC            VALUE 00.
       88  MIDDLE-REC           VALUE 04.
       88  END-REC              VALUE 08.
   01  LEAVING-REC.
       05 LREC OCCURS 1 TO 200 TIMES
              DEPENDING ON LEAVING-REC-LEN      PIC X.
   01  RETURN-REC.
       05 RREC OCCURS 1 TO 200 TIMES
              DEPENDING ON RETURN-REC-LEN   PIC X.
   01  OUTPUT-REC.
       05 OREC OCCURS 1 TO 200 TIMES
              DEPENDING ON OUTPUT-REC-LEN   PIC X.
   01  UNUSED1            PIC 9(8) BINARY.
   01  LEAVING-REC-LEN    PIC 9(8) BINARY.
   01  RETURN-REC-LEN     PIC 9(8) BINARY.
   01  OUTPUT-REC-LEN     PIC 9(8) BINARY.
   01  EXITAREA-LEN       PIC 9(4) BINARY.
   01  EXITAREA.
       05  EAREA   OCCURS 1 TO 256 TIMES
              DEPENDING ON EXITAREA-LEN   PIC X.
```

Figure 23. LINKAGE SECTION Code Example for E35 (Variable-Length Records)

E35 LINKAGE SECTION fields for fixed-length and variable-length records

The fields in the LINKAGE SECTION are used by DFSORT and your routine as stated later in this section. For clarity, the field names from Figure 23 on page 508 have been used.

- To give your COBOL routine the status of the passed records, DFSORT uses the record flags field (RECORD-FLAGS) in the following way:

  0 (FIRST-REC)
  The record leaving DFSORT is the first passed record.

  4 (MIDDLE-REC)
  The record leaving DFSORT is not the first passed record.

  8 (END-REC)
  There is no record leaving DFSORT to pass; all records have been passed to your routine or there were no records to pass.

- DFSORT places the next output record, which usually follows the record in the output area, in the record leaving field (LEAVING-REC). A VLR does not contain an RDW; DFSORT places the length of this VLR

in the record-leaving length field (LEAVING-REC-LEN). The value in the LEAVING-REC-LEN field is the length of the record only and does not include the 4 bytes for the RDW.

- When your routine places an insertion or replacement record in the return record field (RETURN-REC), the VLR must not contain an RDW; your routine must place the length of this record in the return record length field (RETURN-REC-LEN). The value in the RETURN-REC-LEN field is the length of the record only and does not include the 4 bytes for the RDW.
- DFSORT places the record already in the output area in the record in output area field (OUTPUT-REC). A VLR does not contain an RDW. DFSORT places the length, not including the 4 bytes for RDW, of this VLR in the record in output area length field (OUTPUT-REC-LEN).
- DFSORT passes your COBOL E35 routine a 256-byte user exit area field (EXITAREA) that can contain information passed by your COBOL E15 routine. If no information is passed in the EXITAREA field by your COBOL E15 routine the first time the field is passed to your COBOL E35 routine, EXITAREA contains 256 blanks, and the user exit area length field (EXITAREA-LEN) contains 256.

  Any changes you make to the user exit area field or user exit area length field are passed back to your COBOL E35 routine each time it is called by DFSORT.

  Note:

  1. Do not set the user exit area length field to more than 256 bytes.
  2. VLR records have a 4-byte RDW field at the beginning of each record. The maximum record length plus the RDW will be the length defined for the LRECL attribute of your output data set. COBOL programs do not use the RDW and, therefore, the maximum length defined in your COBOL user exit is 4 bytes less than the LRECL value.
  3. You need to code only up to the last field your routine actually uses (for example, up to OUTPUT-REC-LEN if you do not use the user exit area).
  4. DFSORT uses the specified or defaulted value for L3 in the RECORD statement to determine the length of the records your E35 user exit passes back to DFSORT. For fixed-length records, be sure that each record your E35 user exit changes or inserts has a length that is equal to the specified or defaulted L3 value. For variable-length records, be sure that each record your E35 user exit changes or inserts has a length that is less than or equal to the specified or defaulted L3 value. Unwanted truncation or abends may occur if DFSORT uses the wrong length for the records passed to it by your E35 user exit.

     For details of the L3 value, see “RECORD control statement” on page 418.

E35 return codes

Your COBOL E35 routine must pass a return code to DFSORT in the RETURN-CODE field, a COBOL special register. Following are the return codes for the E35 exit:

Return Code
Description

00 (X'00')
No Action

04 (X'04')
Delete Record

08 (X'08')
Do Not Return

12 (X'0C')
Insert Record

16 (X'10')
Terminate DFSORT

20 (X'14')
Alter or Replace Record

0: No Action
If you want DFSORT to retain the record leaving DFSORT unchanged, return with RETURN-CODE set to 0.

4: Delete Record
If you want DFSORT to delete the record leaving DFSORT, return with RETURN-CODE set to 4.

8: Do Not Return
DFSORT keeps returning to your routine until you pass a RETURN-CODE set to 8. After that, the user exit is not used again during the DFSORT application. *Unless you are inserting records after the end of the data set, you must set RETURN-CODE to 8 when DFSORT indicates the end of the data set*. This is done by entering your routine with the record flags field set to 8.

If your user exit routine passes a return code of 8 to DFSORT when input records still remain to be processed, the remaining records are processed by DFSORT but are *not* passed to your user exit.

If you do not have an output data set and would usually return with a return code of 8 before EOF, you can avoid getting the ICE025A message by specifying NOCHECK on the OPTION control statement (if installation option CHECK=NO had not already been specified).

12: Insert Record
If you want DFSORT to add an output record before the record leaving DFSORT:

- Move the insert record to the return record field
- For VLR records, move the length to the return record length field
- Return with RETURN-CODE set to 12.

DFSORT returns to your routine with the inserted record in the record output area field and with the same record as before in the record leaving DFSORT field. In this way, your routine can insert more records or handle the record leaving DFSORT.

You can also insert records after the end of the data set. *DFSORT keeps returning to your routine as long as you pass it a RETURN-CODE 12 and until you return with RETURN-CODE set to 8*.

DFSORT does not perform sequence checking for disk work data set sorts. For tape work data set sorts, DFSORT does not perform sequence checking on inserted records unless you delete the record leaving DFSORT and then replace it.

16: Terminate DFSORT
If you want to terminate DFSORT, return with RETURN-CODE set to 16. DFSORT then returns to its calling program or to the system with a return code of 16.

20: Alter Record
If you want to change the record leaving DFSORT:

- Move the record leaving DFSORT to the return record field
- Change the record in the return record field
- For VLR records, move the length to the return record length field
- Return with RETURN-CODE set to 20.

Note: If your routine changes record size, you must indicate the new size on the RECORD statement.

20: Replace Record
If you want to replace the record leaving DFSORT:

- Move the replacement record to the return record field
- For VLR records, move the length to the return record length field
- Return with RETURN-CODE set to 20.

See "E15/E35 return codes and EXITCK" on page 512 for complete details of the meanings of return codes in various situations.

E35 procedure division requirements

When coding the PROCEDURE DIVISION, the following requirements must be met:

- To return control to DFSORT, you must use the GOBACK statement.
- In the USING option of the PROCEDURE DIVISION header, you must specify *each* 01-level name in the LINKAGE SECTION. You must specify each name in order up to the last one you plan to use even when you do not use all the 01-level names preceding the header.

  Examples:

  For the FLR example, Figure 22 on page 508, you would code:

```
       PROCEDURE DIVISION USING RECORD-FLAGS, LEAVING-REC,
                RETURN-REC, OUTPUT-REC, UNUSED1, UNUSED2,
                UNUSED3, UNUSED4, EXITAREA-LEN, EXITAREA.
```

  For the VLR example, Figure 23 on page 508, you would code:

```
       PROCEDURE DIVISION USING RECORD-FLAGS, LEAVING-REC,
                RETURN-REC, OUTPUT-REC, UNUSED1,
                LEAVING-REC-LEN, RETURN-REC-LEN,
                OUTPUT-REC-LEN, EXITAREA-LEN, EXITAREA.
```

Sample routines written in COBOL

This section provides some sample program user exits written in COBOL.

COBOL E15 user exit: altering records

Figure 24 on page 511 shows an example of a COBOL E15 routine for a data set with fixed-length records of 100 bytes. It examines the department field in the passed record and takes the following action:

- If the department is D29, it changes it to J99.
- If the department is not D29, it accepts the record unchanged.

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.
      CE15.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  LINKAGE SECTION.
  01  RECORD-FLAGS       PIC 9(8) BINARY.
      88  FIRST-REC            VALUE 00.
      88  MIDDLE-REC           VALUE 04.
      88  END-REC              VALUE 08.
  01  NEW-REC.
   05 NFILL1              PIC X(10).
   05 NEW-DEPT            PIC X(3).
   05 NFILL2              PIC X(87).
  01  RETURN-REC.
   05 RFILL1              PIC X(10).
   05 RETURN-DEPT         PIC X(3).
   05 RFILL2              PIC X(87).

  PROCEDURE DIVISION USING RECORD-FLAGS, NEW-REC, RETURN-REC.

      IF END-REC
         MOVE 8 TO RETURN-CODE
      ELSE
        IF NEW-DEPT EQUAL TO "D29"
           MOVE NEW-REC TO RETURN-REC
           MOVE "J99" TO RETURN-DEPT
           MOVE 20 TO RETURN-CODE
        ELSE
           MOVE 0 TO RETURN-CODE
        END-IF
      END-IF

      GOBACK.
```

Figure 24. COBOL E15 Routine Example (FLR)

COBOL E35 user exit: inserting records

Figure 25 on page 512 shows an example of a COBOL E35 routine for a data set with variable-length records up to 200 bytes. It examines the department field in each passed record (records are assumed to be sorted by the department field) and takes the following action:

- It inserts a record for department K22 in the proper sequence.
- It accepts all passed records unchanged.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    CE35.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  INSERT-DONE PIC 9(1) VALUE 0.
01  K22-REC.
 05 K22-MANAGER PIC X(20) VALUE "J. DOE".
 05 K22-DEPT    PIC X(3)  VALUE "K22".
 05 K22-FUNC    PIC X(20) VALUE "ACCOUNTING".
 05 K22-LATER   PIC X(30) VALUE SPACES.
01  LEAVING-VAR-LEN PIC 9(8) BINARY.
LINKAGE SECTION.
01  RECORD-FLAGS       PIC 9(8) BINARY.
    88  FIRST-REC            VALUE 00.
    88  MIDDLE-REC           VALUE 04.
    88  END-REC              VALUE 08.
01  LEAVING-REC.
 05 LREC-MANAGER PIC X(20).
 05 LREC-DEPT    PIC X(3).
 05 LREC-FUNC    PIC X(20).
 05 LREC-LATER   OCCURS 1 TO 157 TIMES
           DEPENDING ON LEAVING-VAR-LEN PIC X.
01  RETURN-REC.
 05 RREC       OCCURS 1 TO 200 TIMES
           DEPENDING ON RETURN-REC-LEN   PIC X.
01  OUTPUT-REC.
 05 OREC       OCCURS 1 TO 200 TIMES
           DEPENDING ON OUTPUT-REC-LEN   PIC X.
01  UNUSED1            PIC 9(8) BINARY.
01  LEAVING-REC-LEN    PIC 9(8) BINARY.
01  RETURN-REC-LEN     PIC 9(8) BINARY.
01  OUTPUT-REC-LEN     PIC 9(8) BINARY.

PROCEDURE DIVISION USING RECORD-FLAGS, LEAVING-REC,
         RETURN-REC, OUTPUT-REC, UNUSED1, LEAVING-REC-LEN,
         RETURN-REC-LEN, OUTPUT-REC-LEN.

    IF END-REC
       MOVE 8 TO RETURN-CODE
    ELSE
      IF INSERT-DONE EQUAL TO 1
         MOVE 0 TO RETURN-CODE
      ELSE
        SUBTRACT 43 FROM LEAVING-REC-LEN
           GIVING LEAVING-VAR-LEN.
        IF LREC-DEPT GREATER THAN K22-DEPT
           MOVE 1 TO INSERT-DONE
           MOVE 43 TO RETURN-REC-LEN
           MOVE K22-REC TO RETURN-REC
           MOVE 12 TO RETURN-CODE
        ELSE
          MOVE 0 TO RETURN-CODE
        END-IF
      END-IF

    GOBACK.
```

Figure 25. COBOL E35 Routine Example (VLR)

E15/E35 return codes and EXITCK

DFSORT's interpretation of E15 and E35 return codes depends upon whether EXITCK=STRONG or EXITCK=WEAK is in effect. See the discussion of the EXITCK option in “OPTION control statement” on page 173 for more information. The following tables show the exact meaning of each E15 and E35 return code with EXITCK=STRONG and EXITCK=WEAK in all possible situations.

Note:

1. You can determine whether EXITCK=STRONG or EXITCK=WEAK is in effect from message ICE132I.

2. Use of EXITCK=WEAK can make it difficult to detect errors in the logic of your E15 and E35 user exit routines.
3. EXITCK=WEAK is treated like EXITCK=STRONG if tape work data sets are specified for a sort application or if the Blockset technique is not selected for a merge application.

Table 78. E15 Without a SORTIN Data Set

E15 Without a SORTIN Data SetE15 Return Code	Meaning with EXITCK=STRONG	Meaning with EXITCK=WEAK
0	Invalid	Do not return
4	Invalid	Do not return
8	Do not return	Do not return
12	Insert record	Insert record
16	Terminate DFSORT	Terminate DFSORT
20 (COBOL only)	Invalid	Do not return
All others	Invalid	Invalid

Table 79. E15 With a SORTIN Data Set Before End of Input

E15 With a SORTIN Data Set Before End of InputE15 Return Code	Meaning with EXITCK=STRONG or EXITCK=WEAK
0	No action/record altered
4	Delete record
8	Do not return
12	Insert record
16	Terminate DFSORT
20 (COBOL only)	Alter/replace record
All others	Invalid

Table 80. E15 With a SORTIN Data Set After End of Input

E15 With a SORTIN Data Set After End of InputE15 Return Code	Meaning with EXITCK=STRONG	Meaning with EXITCK=WEAK
0	Invalid	Do not return
4	Invalid	Do not return
8	Do not return	Do not return
12	Insert record	Insert record
16	Terminate DFSORT	Terminate DFSORT
20 (COBOL only)	Invalid	Do not return
All others	Invalid	Invalid

Table 81. E35 With a SORTOUT or OUTFIL Data Set Before End of Input

E35 With a SORTOUT or OUTFIL Data Set Before End of InputE35 Return Code	Meaning with EXITCK=STRONG or EXITCK=WEAK
0	No action/record altered
4	Delete record

Table 81. E35 With a SORTOUT or OUTFIL Data Set Before End of Input (continued)

E35 With a SORTOUT or OUTFIL Data Set Before End of InputE35 Return Code	Meaning with EXITCK=STRONG or EXITCK=WEAK
8	Do not return
12	Insert record
16	Terminate DFSORT
20 (COBOL only)	Alter/replace record
All others	Invalid

Table 82. E35 Without a SORTOUT or OUTFIL Data Set Before End of Input

E35 Without a SORTOUT or OUTFIL Data Set Before End of InputE35 Return Code	Meaning with EXITCK=STRONG	Meaning with EXITCK=WEAK
0	Invalid	Delete record
4	Delete record	Delete record
8	Do not return	Do not return
12	Invalid	Delete record
16	Terminate DFSORT	Terminate DFSORT
20 (COBOL only)	Invalid	Delete record
All others	Invalid	Invalid

Table 83. E35 With a SORTOUT or OUTFIL Data Set After End of Input

E35 With a SORTOUT or OUTFIL Data Set After End of InputE35 Return Code	Meaning with EXITCK=STRONG	Meaning with EXITCK=WEAK
0	Invalid	Do not return
4	Invalid	Do not return
8	Do not return	Do not return
12	Insert record	Insert record
16	Terminate DFSORT	Terminate DFSORT
20 (COBOL only)	Invalid	Do not return
All others	Invalid	Invalid

Table 84. E35 without a SORTOUT or OUTFIL Data Set After End of Input

E35 without a SORTOUT or OUTFIL Data Set After End of InputE35 Return Code	Meaning with EXITCK=STRONG	Meaning with EXITCK=WEAK
0	Invalid	Do not return
4	Invalid	Do not return
8	Do not return	Do not return
12	Invalid	Do not return
16	Terminate DFSORT	Terminate DFSORT
20 (COBOL only)	Invalid	Do not return

Table 84. E35 without a SORTOUT or OUTFIL Data Set After End of Input (continued)

E35 without a SORTOUT or OUTFIL Data Set After End of InputE35 Return Code	Meaning with EXITCK=STRONG	Meaning with EXITCK=WEAK
All others	Invalid	Invalid

Chapter 6. Invoking DFSORT from a program

Invoking DFSORT dynamically

DFSORT can be invoked dynamically from programs written in COBOL or PL/I. Specific information on dynamic invocation is covered in the COBOL and PL/I programming guides. JCL requirements are the same as those for assembler.

This section explains what you need to know to initiate DFSORT from within your assembler program using a system macro instruction instead of an EXEC job control statement in the input stream. Specific restrictions on invoking DFSORT from PL/I and COBOL are listed in "Restrictions for dynamic invocation " on page 536.

What are system macro instructions?

System macro instructions are macro instructions provided by IBM for communicating service requests to the control program. You can use these instructions only when programming in assembler language. They are processed by the assembler program using macro definitions supplied by IBM and placed in the macro library when your control program was installed.

You can specify one of three system macro instructions to pass control to the program: LINK, ATTACH, or XCTL.

When you issue one of these instructions, the first load module of DFSORT is brought into main storage. The linkage relationship between your program and DFSORT differs according to which of the instructions you have used. For a complete description of the macro instructions and how to use them, refer to *z/OS MVS Programming: Assembler Services Guide*, *z/OS MVS Programming: Assembler Services Reference ABE-HSP*, and *z/OS MVS Programming: Assembler Services Reference IAR-XCT*.

Using system macro instructions

To initiate DFSORT processing with a system macro instruction, you must:

- Write the required job control language (JCL) DD statements.
- Write DFSORT control statements as operands of assembler DC instructions. (Using DFSPARM or SORTCNTL data sets to specify program control statements can be more convenient. See Chapter 3, "Using DFSORT program control statements," on page 77 for details.)
- Write a parameter list containing information to be passed to DFSORT and a pointer containing the address of the parameter list. DFSORT accepts three types of parameter lists: a 24-bit parameter list, an extended (31-bit) parameter list, and a 64-bit parameter list. Although you can choose any one of the three parameter lists, the 64-bit parameter list can perform a superset of the functions of the other parameter lists; therefore, it is recommended for new DFSORT applications.

  Note: DFSORT can also be called using the system's EXEC PARM parameter list, provided that the rules for passing it are followed (for example, the parameter list must reside below 16MB virtual). DFSORT interprets a call using the EXEC PARM parameter list as a direct invocation rather than a program invocation.
- If you are using the 64-bit parameter list, prepare the macro instruction specifying one of the following as the entry point name: ICEMAN64 or SORT64.
- If you are using the 31-bit or 24-bit parameter list, prepare the macro instruction specifying one of the following as the entry point name: ICEMAN, SORT, IERRCO00, or IGHRCO00.

In addition, the following rule applies:

- If you are invoking DFSORT repeatedly (for example, from an E15 or E35 user exit), you must always wait for the last invoked sort to end before you can give control back to any of your user exits in an earlier invoked sort.

Note: The save area passed to DFSORT must begin on a fullword boundary.

Using JCL DD statements

JCL DD statements are usually required when invoking DFSORT from another program. The statements and their necessary parameters are described in detail.

Overriding DFSORT control statements from programs

You can override the control statements generated or passed by a program (for example, a COBOL SORT verb or PLISRTx routine) with DFSORT's DFSPARM data set.

For example, you could use the following to override the SORT statement generated by a COBOL SORT verb in order to use DFSORT's Year 2000 features:

```
//DFSPARM  DD    *
  OPTION Y2PAST=1956        * set fixed CW of 1956-2055
  SORT FIELDS=(11,6,Y2T,A,  * sort C'yymmdd' using CW
               31,10,CH,A)  * sort other key
/*
```

You can also use DFSPARM to provide different DFSORT control statements for multiple invocations of DFSORT from a program. However, the control statements must be located in temporary or permanent data sets and FREE=CLOSE must be used. Here's an example of using DFSPARM to override the control statements for a COBOL program with three SORT verbs:

```
//DFSPARM DD DSN=DP1,DISP=SHR,FREE=CLOSE
//DFSPARM DD DSN=DP2,DISP=SHR,FREE=CLOSE
//DFSPARM DD DSN=DP3,DISP=SHR,FREE=CLOSE
```

DP1, DP2, and DP3 can contain any DFSORT control statements you like. The statements in DP1 will be used for the first call to DFSORT, the statements in DP2 for the second and the statements in DP3 for the third.

For complete details on DFSPARM, see “DFSPARM DD statement” on page 74.

COBOL also offers additional methods for overriding DFSORT parameters and control statements such as SORT special registers and the IGZSRTCD data set. See your COBOL publications for details.

Invoking DFSORT with the 24-bit parameter list

Entry point name

When using the 24-bit parameter list, you must use ICEMAN, SORT, IERRCO00, or IGHRCO00 as the entry point name for the LINK, ATTACH or XCTL macro.

Providing program control statements

When using the 24-bit parameter list, you must supply the starting and ending address of a valid image of each control statement to be used during run-time. You must provide the image as a character string in EBCDIC format using assembler DC instructions. The rules for preparing the program control statements are as follows:

- At least one control statement must be specified—generally SORT or MERGE. If more than 15 control statements are specified, only the first 15 control statements are accepted and all others are ignored. Control statements can also be specified in SORTCNTL or DFSPARM.
- The MODS statement is required when user exits other than E15, E32, and E35 are to be used. It is also required when the E15 or E35 routine addresses are not passed by the parameter list.
- The following control statements can be passed using the 24-bit parameter list: SORT or MERGE, RECORD, ALTSEQ, DEBUG, MODS, SUM, INREC, OUTREC, INCLUDE or OMIT, JOINKEYS, JOIN, REFORMAT and OUTFIL.
- At least one blank must follow the operation definer (SORT, for example). A control statement can start and end with one or more blanks; however, no other blanks are allowed.
- The content and format of the statements are as described in Chapter 3, “Using DFSORT program control statements,” on page 77, except:
  - Labels are not allowed although a leading blank is optional.
  - Because each control statement image must be defined contiguously by one or more assembler DC instructions, explicit and implicit continuation of statements is neither necessary nor allowed.
- Neither comment statements, blank statements, nor remark fields are permitted.
- If you use ATTACH to initiate the program, you cannot use the checkpoint/restart facility and must not specify CKPT in the SORT statement image.

  For full override and applicability details, see Appendix B, “Specification/override of DFSORT options,” on page 805.

CONTROL statement images example

```
SORTBEG   DC   C' SORT FIELDS=(10,15,CH,A)'
SORTEND   DC   C' '
```

This form, with a trailing blank separately defined, allows you to refer to the last byte of the statement (SORT statement end address) by the name SORTEND.

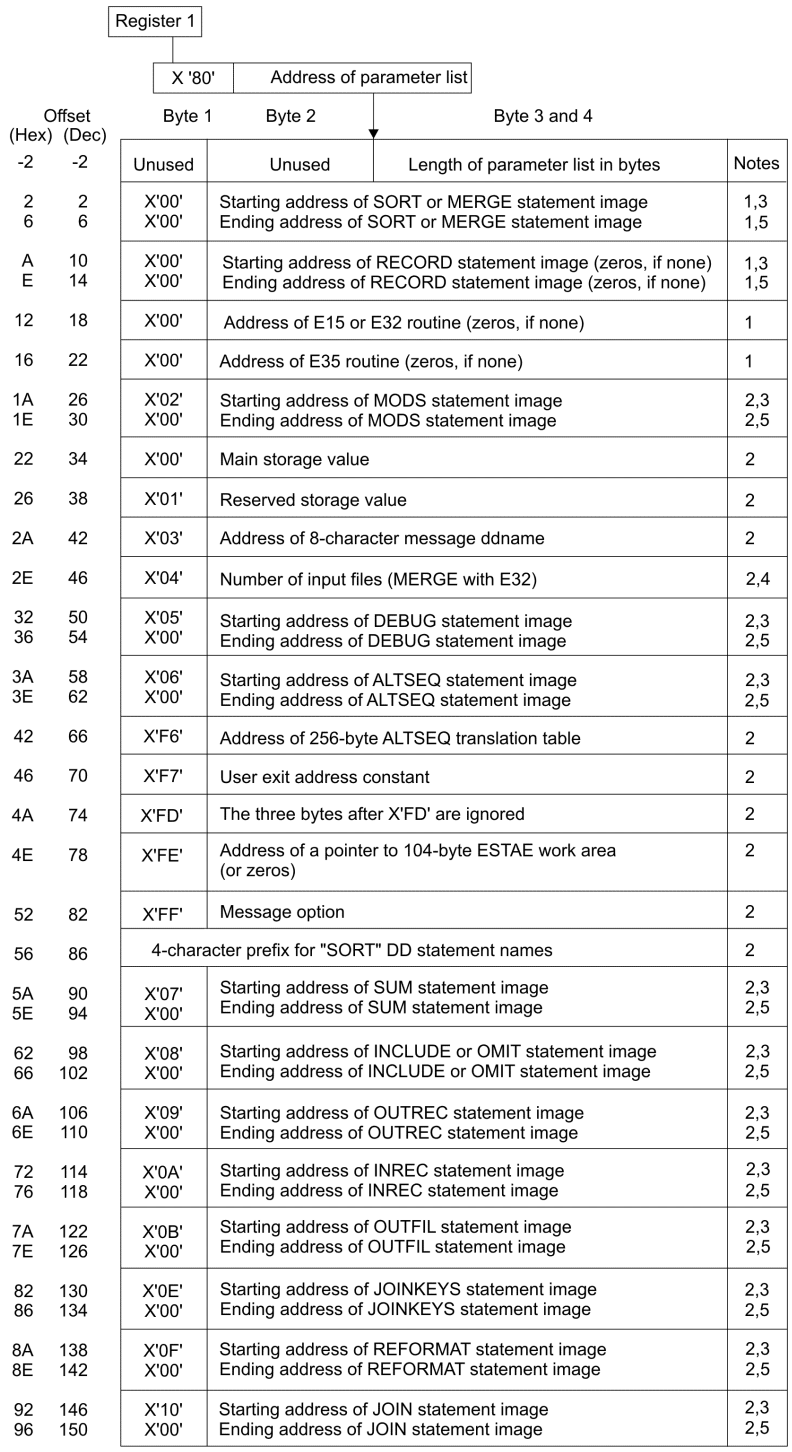
```
INCLBEG   DC   C' INCLUDE COND=(5,3,CH,NE,C'J82'')'
INCLEND   DC   C' '
```

Note: Assembler requires two single apostrophes to represent one single apostrophe.

Format of the 24-bit parameter list

Figure 26 on page 520 shows the format of the 24-bit parameter list and the pointer containing its address which you must pass to DFSORT. Detailed specifications for each of the entries in the parameter list follow.

For full override and applicability details, see Appendix B, “Specification/override of DFSORT options,” on page 805.



Offset (Hex)	Offset (Dec)	Byte 1	Byte 2	Byte 3 and 4	Notes
-2	-2	Unused	Unused	Length of parameter list in bytes	
2 6	2 6	X'00' X'00'	Starting address of SORT or MERGE statement image Ending address of SORT or MERGE statement image		1,3 1,5
A E	10 14	X'00' X'00'	Starting address of RECORD statement image (zeros, if none) Ending address of RECORD statement image (zeros, if none)		1,3 1,5
12	18	X'00'	Address of E15 or E32 routine (zeros, if none)		1
16	22	X'00'	Address of E35 routine (zeros, if none)		1
1A 1E	26 30	X'02' X'00'	Starting address of MODS statement image Ending address of MODS statement image		2,3 2,5
22	34	X'00'	Main storage value		2
26	38	X'01'	Reserved storage value		2
2A	42	X'03'	Address of 8-character message ddname		2
2E	46	X'04'	Number of input files (MERGE with E32)		2,4
32 36	50 54	X'05' X'00'	Starting address of DEBUG statement image Ending address of DEBUG statement image		2,3 2,5
3A 3E	58 62	X'06' X'00'	Starting address of ALTSEQ statement image Ending address of ALTSEQ statement image		2,3 2,5
42	66	X'F6'	Address of 256-byte ALTSEQ translation table		2
46	70	X'F7'	User exit address constant		2
4A	74	X'FD'	The three bytes after X'FD' are ignored		2
4E	78	X'FE'	Address of a pointer to 104-byte ESTAE work area (or zeros)		2
52	82	X'FF'	Message option		2
56	86	4-character prefix for "SORT" DD statement names			2
5A 5E	90 94	X'07' X'00'	Starting address of SUM statement image Ending address of SUM statement image		2,3 2,5
62 66	98 102	X'08' X'00'	Starting address of INCLUDE or OMIT statement image Ending address of INCLUDE or OMIT statement image		2,3 2,5
6A 6E	106 110	X'09' X'00'	Starting address of OUTREC statement image Ending address of OUTREC statement image		2,3 2,5
72 76	114 118	X'0A' X'00'	Starting address of INREC statement image Ending address of INREC statement image		2,3 2,5
7A 7E	122 126	X'0B' X'00'	Starting address of OUTFIL statement image Ending address of OUTFIL statement image		2,3 2,5
82 86	130 134	X'0E' X'00'	Starting address of JOINKEYS statement image Ending address of JOINKEYS statement image		2,3 2,5
8A 8E	138 142	X'0F' X'00'	Starting address of REFORMAT statement image Ending address of REFORMAT statement image		2,3 2,5
92 96	146 150	X'10' X'00'	Starting address of JOIN statement image Ending address of JOIN statement image		2,3 2,5

Figure 26. The 24-Bit Parameter List

Notes to Figure 26 on page 520:

1.
Required entry. Must appear in the relative position shown. The offset shown is the actual offset of this entry.

2.
Optional entry. Can appear anywhere after the required entries. The displayed offset is for identification purposes only—the actual offset of this entry can vary. Optional entries must be consecutive but can appear in any order.

3.
A specific control statement. Shown for illustrative purposes only. SORT or MERGE, RECORD, ALTSEQ, DEBUG, MODS, SUM, INREC, OUTREC, INCLUDE or OMIT, JOINKEYS, JOIN, REFORMAT and OUTFIL can be passed using any of the following hex entry codes: X'00' (see Note 1), X'02', X'05' through X'0B', X'0E' through X'11', X'16', X'18' and X'20' through X'29'.

4.
Required entry if the MERGE statement is present and input is supplied through an E32 user exit. This entry is not required if the FILES option of the MERGE statement is specified.

5.
Required entry. Contains the ending address for a control statement and must immediately follow the entry containing the starting address for that same control statement.

The specifications for each of the parameter list entries follow:

Byte
Explanation

-2 to -1
Unused.

0 to +1
The byte count. This 2-byte field contains the length in bytes of the parameter list. This two byte field is not included when counting the number of bytes occupied by the list.

The total length of the required entries is 24 (X'0018'). All optional entries are four bytes long except those referring to control statement images which are eight bytes each.

2-5
The starting address of the SORT or MERGE statement image. Must be in the last three bytes of this fullword. The first byte must contain X'00'.

6-9
The ending address of the SORT or MERGE statement image. Must be in the last three bytes. The first byte must contain X'00'.

10-13
The starting address of the RECORD statement image, if any; otherwise, all zeros. Must be in the last three bytes. The first byte must contain X'00'.

14-17
The ending address of the RECORD statement image, if any; otherwise, all zeros. Must be in the last three bytes. The first byte must contain X'00'.

18-21
The address of the E15 or E32 routine that your program has placed in main storage, if any; otherwise, all zeros. Must be in the last three bytes. The first byte must contain X'00'.

22-25
The address of the E35 routine that your program has placed in main storage, if any; otherwise, all zeros. Must be in the last three bytes. The first byte must contain X'00'.

26-29
The starting address of the MODS statement image. Must be in the last three bytes. The first byte must contain X'02'.

30-33
The ending address of the MODS statement. Must be in the last three bytes. The first byte must contain X'00'.

34-37
Main storage value. The first byte must contain X'00'. The next three bytes contain either the characters MAX or a hexadecimal value. You can use this option to temporarily override the SIZE installation option. For full override and applicability details, see Appendix B, “Specification/override of DFSORT options,” on page 805. For an explanation of this value, see the discussion of the MAINSIZE parameter in “OPTION control statement” on page 173.

38-41
A reserved main storage value. The first byte must contain X'01'. The next three bytes contain a hexadecimal value that specifies a number of bytes to be reserved, where the minimum is 4K. For an explanation of this value, see the explanation of the RESINV parameter in "OPTION control statement" on page 173.

You can use this option to temporarily override the RESINV installation option. For full override and applicability details, see Appendix B, "Specification/override of DFSORT options," on page 805.

42-45
Message ddname. The first byte must contain X'03'. The next three bytes contain the address of an 8-byte DD statement name for the message data set, padded with blanks on the right if necessary. The name can be any valid DD statement name, but must be unique.

You can use this option to temporarily override the MSGDDN installation option. For full override details, see Appendix B, "Specification/override of DFSORT options," on page 805. For details on the use of the message data set, see *z/OS DFSORT Messages, Codes and Diagnosis Guide*.

46-49
Number of input files to a merge. This entry is needed only if the MERGE statement is present without the FILES option and input to the merge is supplied through the E32 user exit. The first byte must contain X'04'. The next three bytes contain the number of files in hexadecimal. For full override and applicability details, see Appendix B, "Specification/override of DFSORT options," on page 805.

50-53
The starting address of the DEBUG statement image. Must be in the last three bytes. The first byte must contain X'05'.

54-57
The ending address of the DEBUG statement image. Must be in the last three bytes. The first byte must contain X'00'.

58-61
The starting address of the ALTSEQ statement image. Must be in the last three bytes. The first byte must contain X'06'.

62-65
The ending address of the ALTSEQ statement image. Must be in the last three bytes. The first byte must contain X'00'.

66-69
The address of a 256-byte translate table supplied instead of an ALTSEQ statement. The first byte must contain X'F6'. If this parameter is present, the X'06' parameter is ignored. For full override and applicability details, see Appendix B, "Specification/override of DFSORT options," on page 805.

70-73
User exit address constant. These 4 bytes are passed to E15 (at offset 4 in the E15 parameter list), to E32 (at offset 8 in the E32 parameter list) or to E35 (at offset 8 in the E35 parameter list) after DFSORT replaces the X'F7' with X'00'.

Note: The user exit address constant must not be used for a Conventional merge or tape work data set sort application.

74-77
X'FD' in the first byte (the VLSHRT option) specifies that DFSORT is to continue processing if it finds a variable-length input record too short to contain all specified control fields, compare fields, or summary fields. For full details of this option, see the discussion of the VLSHRT parameter in "OPTION control statement" on page 173. You can use this option to temporarily override the VLSHRT installation option. For full override and applicability details, see Appendix B, "Specification/override of DFSORT options," on page 805.

78-81
If the first byte contains X'FE', you can use the next three bytes to pass an address of a 104-byte field save area where ESTAE information is saved. These bytes must contain zeros if the ESTAE information is not saved.

If a system or user exit abend occurs, the DFSORT ESTAE recovery routine will copy the first 104 bytes of the SDWA into this area before returning to any higher level ESTAE recovery routines.

For more information on the DFSORT ESTAE recovery routine, see Appendix E, “DFSORT abend processing,” on page 847

82-85
The message option. The first byte must contain X'FF'. The following three bytes contain the characters NOF, (I), or (U). You can use this option to temporarily override the MSGPRT installation option.

NOF
Messages and control statements are not printed. Critical messages are written to the master console.

(I)
All messages except diagnostic messages (ICE800I to ICE999I) are printed. Critical messages are also written to the master console. Control statements are printed only if LIST is in effect.

(U)
Only critical messages are printed. They are also written to the master console. Control statements are not printed (NOLIST is forced).

All messages are written to the message data set. For details on use of the message data set, see *z/OS DFSORT Messages, Codes and Diagnosis Guide* For full override and applicability details, see Appendix B, “Specification/override of DFSORT options,” on page 805.

For compatibility reasons, the forms (NO, (AB, (AP, (AC, (CB, (CC, (CP, (PC, (SC, and (SP are also accepted.

The following list shows the equivalent specifications for these aliases:

Table 85. Aliases for Message Option

Aliases for Message OptionOption	MSGPRT	MSGCON
(NO	NONE	NONE
(AB	ALL	ALL
(AP	ALL	CRITICAL
(AC	NONE	ALL
(CB	CRITICAL	CRITICAL
(CC	NONE	CRITICAL
(CP	CRITICAL	CRITICAL
(PC	ALL	ALL
(SC	ALL	CRITICAL
(SP	CRITICAL	ALL

86-89
Four characters, which replace "SORT" in the following ddnames: SORTIN, SORTOUT, SORTINn, SORTINnn, SORTOFd, SORTOFdd, SORTWKd, SORTWKdd, SORTJNF1, SORTJNF2, and SORTCNTL. You must use this option when you dynamically invoke DFSORT more than once in a program step.

The four characters must all be alphanumeric or national ($, #, or @) characters. The first character must be alphabetic, and the reserved names DIAG, BALN, OSCL, POLY, CRCX, PEER, LIST, and SYSc (where c is any alphanumeric character) must not be used. Otherwise, the four characters are ignored.

For example, if you use ABC# as replacement characters, DFSORT uses statements ABC#IN, ABC#CNTL, ABC#WKdd, and ABC#OUT instead of SORTIN, SORTCNTL, SORTWKdd, and SORTOUT.

Note: This parameter is equivalent to the SORTDD=cccc run-time option.

90-93
The starting address of the SUM statement image. Must be in the last three bytes. The first byte must contain X'07'.

94-97
The ending address of the SUM statement image. Must be in the last three bytes. The first byte must contain X'00'.

98-101
The starting address of the INCLUDE or OMIT statement image. Must be in the last three bytes. The first byte must contain X'08'.

102-105
The ending address of the INCLUDE or OMIT statement image. Must be in the last three bytes. The first byte must contain X'00'.

106-109
The starting address of the OUTREC statement image. Must be in the last three bytes. The first byte must contain X'09'.

110-112
The ending address of the OUTREC statement image. Must be in the last three bytes. The first byte must contain X'00'.

114-116
The starting address of the INREC statement image. Must be in the last three bytes. The first byte must contain X'0A'.

118-121
The ending address of the INREC statement image. Must be in the last three bytes. The first byte must contain X'00'.

122-125
The starting address of the OUTFIL statement image. Must be in the last three bytes. The first byte must contain X'0B'.

126-129
The ending address of the OUTFIL statement image. Must be in the last three bytes. The first byte must contain X'00'.

130-133
The starting address of the JOINKEYS statement image. Must be in the last three bytes. The first byte must contain X'0E'.

134-137
The ending address of the JOINKEYS statement image. Must be in the last three bytes. The first byte must contain X'00'.

138-141
The starting address of the REFORMAT statement image. Must be in the last three bytes. The first byte must contain X'0F'.

142-145
The ending address of the REFORMAT statement image. Must be in the last three bytes. The first byte must contain X'00'.

146-149
The starting address of the JOIN statement image. Must be in the last three bytes. The first byte must contain X'10'.

150-153
The ending address of the JOIN statement image. Must be in the last three bytes. The first byte must contain X'00'.

Invoking DFSORT with the extended (31-bit) parameter list

Entry point name

When using the extended parameter list, you must use ICEMAN, SORT, IERRCO00, or IGHRCO00 as the entry point name for the LINK, ATTACH or XCTL macro.

Providing program control statements

When using the extended parameter list, the control statements are written in a single area to which the parameter list points. The control statement area consists of:

- A 2-byte field containing the length (in binary) of the character string to follow.
- A character string in EBCDIC format using assembler DC instructions and containing valid images of the control statements to be used during run-time.

The rules for preparing the program control statements are:

- The control statements must be separated by one or more blanks. A blank preceding the first statement is optional; however, a trailing blank is required. No labels, comment statements, or remark fields are allowed. Because each control statement image must be defined contiguously by one or more assembler DC instructions, explicit and implicit continuation of statements is not necessary or allowed.
- The MODS statement is required when user exits other than E15, E18, E32, E35, and E39 are to be used or when the E15, E18, E35, or E39 routine addresses are not passed by the parameter list.
- All of the control statements described in Chapter 3, “Using DFSORT program control statements,” on page 77 can be specified. None is required, but SORT, MERGE, or OPTION COPY must be specified in the parameter list, SORTCNTL, or DFSPARM.
- If you use ATTACH to initiate the program, you cannot use the checkpoint/restart facility. Do not specify CKPT on the SORT or OPTION statement.

For full override and applicability details, see Appendix B, “Specification/override of DFSORT options,” on page 805.

Format of the extended parameter list

Figure 27 on page 526 shows the format of the extended parameter list and the pointer, which you must pass to DFSORT, containing its address.

The first parameter must be specified. A 4-byte field containing X'FFFFFFFF' *must* be used to indicate the end of the parameter list. It can be coded anywhere after the first parameter.

If a parameter is specified, it must appear in the indicated position and must contain a 31-bit address or a clean (the first 8 bits containing zeros) 24-bit address. If a parameter is not specified, it is treated as if it were specified as zeros. For full override and applicability details, see Appendix B, “Specification/override of DFSORT options,” on page 805.

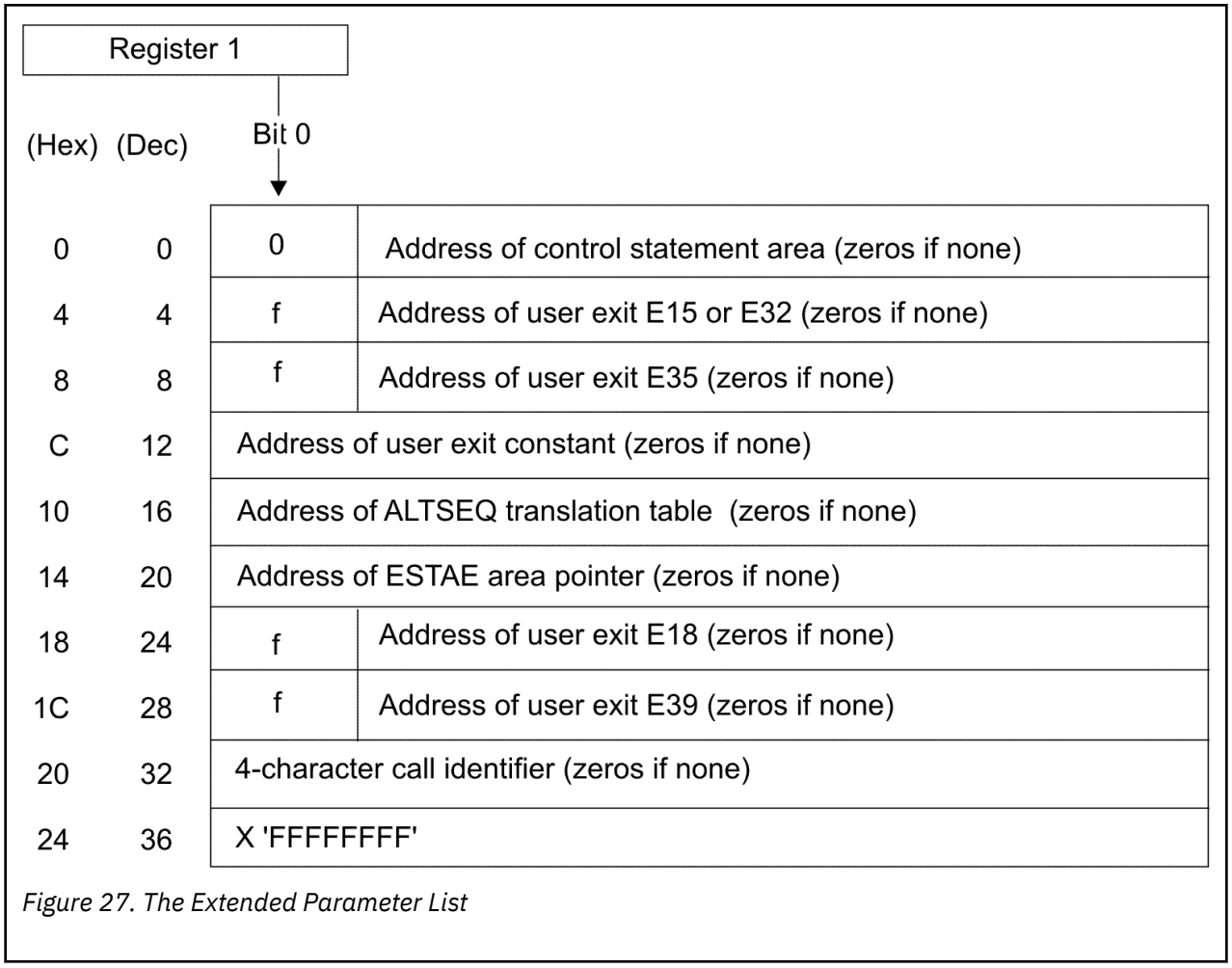


Figure 27. The Extended Parameter List

Detailed specifications for each of the entries in the parameter list follow:

Byte
Explanation

0-3
Required. The address of the area containing the DFSORT control statements, if any; otherwise, all zeros. The high order bit must be 0 to identify this as an extended parameter list.

Refer to the previous section for the format of the control statement area.

Attention: The area must start with a two-byte length field.

If you specify this parameter as zeros, you must supply all the required control statements in DFSPARM or SORTCNTL.

4-7
Optional. The address of the E15 or E32 user exit routine that your program has placed in main storage (for example, via LOAD), if any; otherwise, all zeros.

f (bit 0) has the following meaning:

- 0 = Enter the user exit with 24-bit addressing in effect (AMODE 24).
- 1 = Enter the user exit with 31-bit addressing in effect (AMODE 31).

Note: If the Blockset or Peerage/Vale technique is not selected, the user exit is always entered with 24-bit addressing in effect (AMODE 24).

8-11
Optional. The address of the E35 user exit routine that your program has placed in main storage (for example, via LOAD), if any; otherwise, all zeros.

f (bit 0) has the following meaning:

- 0 = Enter the user exit with 24-bit addressing in effect (AMODE 24).
- 1 = Enter the user exit with 31-bit addressing in effect (AMODE 31).

Note: If the Blockset or Peerage/Vale technique is not selected, the user exit is always entered with 24-bit addressing in effect (AMODE 24).

12-15
Optional. This field will be passed to the E15, E32 or E35 user exit routines.

Note: The user exit address constant must not be used for a Conventional merge or tape work data set sort application.

16-19
Optional. The address of a 256-byte ALTSEQ translation table supplied instead of an ALTSEQ statement, if any; otherwise, all zeros. You can use this option to override an ALTSEQ installation option. For full override and applicability details, see Appendix B, "Specification/override of DFSORT options," on page 805.

20-23
Optional. The address of a 4-byte field containing the address of a 112-byte work area where ESTAE information is saved, or all zeros if the ESTAE information is not saved.

If a system or user exit abend occurs, the DFSORT recovery routine will copy the first 112 bytes of the software diagnostic work area (SDWA) into this area before returning to your ESTAE recovery routine.

24-27
Optional. The address of the E18 user exit routine that your program has placed in main storage (for example, via LOAD), if any; otherwise, all zeros.

Note: This parameter is ignored for a merge application and for a tape work data set sort application.

f (bit 0) has the following meaning:

- 0 = Enter the user exit with 24-bit addressing in effect (AMODE 24).
- 1 = Enter the user exit with 31-bit addressing in effect (AMODE 31).

Note: If the Blockset or Peerage/Vale technique is not selected, the user exit is always entered with 24-bit addressing in effect (AMODE 24).

28-31
Optional. The address of the E39 user exit routine that your program has placed in main storage (for example, via LOAD), if any; otherwise, all zeros.

Note: This parameter is ignored for a conventional merge application and for a tape work data set sort application.

f (bit 0) has the following meaning:

- 0 = Enter the user exit with 24-bit addressing in effect (AMODE 24).
- 1 = Enter the user exit with 31-bit addressing in effect (AMODE 31).

Note: If the Blockset or Peerage/Vale technique is not selected, the user exit is always entered with 24-bit addressing if effect (AMODE 24).

32-35
Optional. 4 characters to be used as an identifier for this call to DFSORT. This field can be used to uniquely identify each call to DFSORT from a program that calls DFSORT more than once. DFSORT prints message ICE200I to display the field identifier exactly as you specify it; the field is not checked for valid characters.

If the field identifier is specified, it must appear in the indicated position. If the identifier field contains zeros (X'00000000'), or X'FFFFFFFF' is used to end the parameter list before or at the field identifier, DFSORT does not print message ICE200I.

Note: The list can be ended after any parameter. The last parameter in the list *must* be followed by X'FFFFFFFF'.

Invoking DFSORT with the 64-bit parameter list

Entry point name

When using the 64-bit parameter list, you must use ICEMAN64 or SORT64 as the entry point name for the LINK, ATTACH or XCTL macro.

64-bit address terminology

When referring to 64-bit addresses:

- A **clean 24-bit address** refers to binary zeros in the high-order 5 bytes followed by a 24-bit address in the low-order 3 bytes, that is, X'0000000000aaaaaa'
- A **clean 31-bit address** refers to binary zeros in the high-order 4 bytes followed by a 31-bit address in the low-order 4 bytes, that is, X'00000000aaaaaaaa' where Bit 32 of the 31-bit address must be 0.
- A 64 bit address uses all 64-bits for the address.

The following summarizes the previous information:

Table 86. Summary of 64-bit Terminology for Programs

Address	Bits 0-31	Bit 32	Bits 33-39	Bits 40-63
24-bit	0	0	0	Address
31-bit	0	0	Address	
64-bit	Address			

Providing program control statements

When using the 64-bit parameter list, the control statements are written in a single area to which the parameter list points. The control statement area can be above or below the bar. The control statement area consists of:

- A 2-byte field containing the length (in binary) of the character string to follow.
- A character string in EBCDIC format using assembler DC instructions and containing valid images of the control statements to be used during run-time.

The rules for preparing the program control statements are:

- The control statements must be separated by one or more blanks. A blank preceding the first statement is optional; however, a trailing blank is required. No labels, comment statements, or remark fields are allowed. Because each control statement image must be defined contiguously by one or more assembler DC instructions, explicit and implicit continuation of statements is not necessary or allowed.
- The MODS statement is required when user exits other than E15, E18, E32, E35, and E39 are to be used or when the E15, E18, E35, or E39 routine addresses are not passed by the parameter list.
- All of the control statements described in Chapter 3, "Using DFSORT program control statements," on page 77 can be specified. None is required, but SORT, MERGE, or OPTION COPY must be specified in the parameter list, SORTCNTL, or DFSPARM.
- If you use ATTACH to initiate the program, you cannot use the checkpoint/restart facility. Do not specify CKPT on the SORT or OPTION statement.

For full override and applicability details, see Appendix B, “Specification/override of DFSORT options,” on page 805.

Format of the 64-bit parameter list

Figure 28 on page 530 shows the format of the 64-bit parameter list. As indicated, some 64-bit fields must contain zeros in the first 32 bits as shown in Figure 28 on page 530. You must pass the pointer to the parameter list in 64-bit register 1.

The DFSORT target library, SICEUSER, contains a mapping macro called ICEPL64, which provides a separate Assembler DSECT for the 64-bit invocation parameter list.

The 8-character identifier 'PL64SORT' must be present at the start of the parameter list. The list must be 136 bytes long with parameters or zeros specified in the indicated positions. Each parameter is 64-bits wide (8 bytes) and must contain a 64-bit address or a clean 31-bit address as indicated in the description.

64-bit Register 1 = address to invocation parameter list

Hex	Dec	Length	Bit	Contents
0	0	8		C'PL64SORT'
8	8	1	0	AMODE 24 E15 or E32
			1	AMODE 31 E15 or E32
			2	AMODE 64 E15 or E32
			3	AMODE 24 E35
			4	AMODE 31 E35
			5	AMODE 64 E35
			6	AMODE 24 E18
			7	AMODE 31 E18
9	9	1	0	Reserved. Must be set to 0.
			1	AMODE 24 E39
			2	AMODE 31 E39
			3	Reserved. Must be set to 0.
			4	E15 or E32 type of exit parameter list used, and how Register 1 should be interpreted
			5	E35 type of exit parameter list used, and how Register 1 should be interpreted
			6	E18 type of exit parameter list used, and how Register 1 should be interpreted
			7	E39 type of exit parameter list used, and how Register 1 should be interpreted
A	10	13		Reserved. Must be set to zeros.
17	23	1	0	64-bit block list used for E15 exit
			1	64-bit block list used for E35 exit
			2	Reserved. Must be set to zeros.
18	24	8		Address of control statement area (zeros if none)
20	32	8		X'00000000' | Address of user exit E15 or E32 (zeros if none)
28	40	8		X'00000000' | Address of user exit E35 (zeros if none)
30	48	8		X'00000000' | Address of user exit constant
38	56	8		Address of ALTSEQ Translation table (zeros if none)
40	64	8		X'00000000' | Address of ESTAE Area Pointer (zeros if none)
48	72	8		X'00000000' | Address of user exit E18 (zeros if none)
50	80	8		X'00000000' | Address of user exit E39 (zeros if none)
50	88	8		X'00000000' | 4-Character call identifier (zeros if none)
60	96	8		Address of user block list parameters area
68	104	32		Reserved. Must be set to zeros.

Figure 28. The 64-Bit Parameter List

Detailed specifications for each of the entries in the parameter list follow:

Byte (dec)
Explanation

0-7
C'PL64SORT' identifier used to indicate the 64-bit invocation parameter list has been specified.

8

Bit 0
Indicates 24-bit addressing mode (AMODE 24) will be used when the E15 or E32 exit is entered.

- 0 = Do not enter the E15 or E32 with 24-bit addressing in effect.
- 1 = Enter the E15 or E32 with 24-bit addressing in effect.

Bit 1
Indicates 31-bit addressing mode (AMODE 31) will be used when the E15 or E32 exit is entered.

- 0 = Do not enter the E15 or E32 with 31-bit addressing in effect.
- 1 = Enter the E15 or E32 with 31-bit addressing in effect.

Bit 2
Indicates 64-bit addressing mode (AMODE 64) will be used when the E15 or E32 exit is entered.

- 0 = Do not enter the E15 or E32 with 64-bit addressing in effect.
- 1 = Enter the E15 or E32 with 64-bit addressing in effect.

Note: If an E15 or E32 is used, only one of the E15 or E32 AMODE bits described previously can be set. If more than one E15 AMODE bit is set, DFSORT will terminate.

Bit 3
Indicates 24-bit addressing mode (AMODE 24) will be used when the E35 exit is entered.

- 0 = Do not enter the E35 with 24-bit addressing in effect.
- 1 = Enter the E35 with 24-bit addressing in effect.

Bit 4
Indicates 31-bit addressing mode (AMODE 31) will be used when the E35 exit is entered.

- 0 = Do not enter the E35 with 31-bit addressing in effect.
- 1 = Enter the E35 with 31-bit addressing in effect.

Bit 5
Indicates 64-bit addressing mode (AMODE 64) will be used when the E35 exit is entered.

- 0 = Do not enter the E35 with 64-bit addressing in effect.
- 1 = Enter the E35 with 64-bit addressing in effect.

Note: If an E35 is used, only one of the previous E35 AMODE bits can be set. If more than one E35 AMODE bit is set, DFSORT will terminate.

Bit 6
Indicates 24-bit addressing mode (AMODE 24) will be used when the E18 exit is entered.

- 0 = Do not enter the E18 with 24-bit addressing in effect.
- 1 = Enter the E18 with 24-bit addressing in effect.

Bit 7
Indicates 31-bit addressing mode (AMODE 31) will be used when the E18 exit is entered.

- 0 = Do not enter the E18 with 31-bit addressing in effect.
- 1 = Enter the E18 with 31-bit addressing in effect.

Note: If an E18 is used, only one of the previous E18 AMODE bits can be set. If more than one E18 AMODE bit is set, DFSORT will terminate.

9

Bit 0
Reserved. Must be set to zero.

Bit 1
Indicates 24-bit addressing mode (AMODE 24) will be used when the E39 exit is entered.

- 0 = Do not enter the E39 with 24-bit addressing in effect
- 1 = Enter the E39 with 24-bit addressing in effect.

Bit 2
Indicates 31-bit addressing mode (AMODE 31) will be used when the E39 exit is entered.

- 0 = Do not enter the E39 with 31-bit addressing in effect
- 1 = Enter the E39 with 31-bit addressing in effect.

Note: If an E39 is used, only one of the previous E39 AMODE bits can be set. If more than one E39 AMODE bit is set, DFSORT will terminate.

Bit 3
Reserved. Must be set to zero.

Bit 4
Indicates type of E15 or E32 exit parameter list used

- 0 = 32 bit exit parameter list used, 32-bit Register 1 returned from exit, and exit saves 32-bit registers in save area (Format-0 72 byte save area) pointed at by 32-bit Register 13.
- 1 = 64 bit exit parameter list used, 64-bit Register 1 returned from exit, and exit saves 64-bit registers in save area (Format-4 144 byte save area) pointed at by 64-bit Register 13.

Bit 5
Indicates type of E35 exit parameter list used

- 0 = 32 bit exit parameter list used, 32-bit Register 1 returned from exit, and exit saves 32-bit registers in save area (Format-0 72 byte save area) pointed at by 32-bit Register 13.
- 1 = 64 bit exit parameter list used, 64-bit Register 1 returned from exit, and exit saves 64-bit registers in save area (Format-4 144 byte save area) pointed at by 64-bit Register 13.

Bit 6
Indicates type of E18 exit parameter list used

- 0 = 32 bit exit parameter list used, 32-bit Register 1 returned from exit, and exit saves 32-bit registers in save area (Format-0 72 byte save area) pointed at by 32-bit Register 13.
- 1 = Reserved.

Bit 7
Indicates type of E39 exit parameter list used

- 0 = 32 bit exit parameter list used, 32-bit Register 1 returned from exit, and exit saves 32-bit registers in save area (Format-0 72 byte save area) pointed at by 32-bit Register 13.
- 1 = Reserved.

Note: In the previous situations when a 32-bit Register 1 is returned, the high half of Register 1 will be ignored by DFSORT when the exit returns to DFSORT.

10-22
Reserved. Must be set to zeros.

23

Bit 0
Indicates 64-bit block list is used for E15 Exit.

Bit 1
Indicates 64-bit block list is used for E35 Exit.

Bit 2-7
Reserved. Must be set to zeros.

24-31
The address of the area containing the DFSORT control statements, if any; otherwise, all zeros. Can be a 64-bit address, a clean 31-bit address or a clean 24-bit address.

Refer to the previous section for the format of the control statement area.

Attention: The area must start with a two-byte length field. If zeros are specified, all the required control statements must be provided in DFSORT DD DFSPARM or SORTCNTL.

32-39
The address of the E15 or E32 user exit routine that the program has placed in main storage (for example, via LOAD), if any; otherwise, all zeros. Must be a clean 31-bit address or a clean 24-bit address.

40-47
The address of the E35 user exit routine that the program has placed in main storage (for example, via LOAD), if any; otherwise, all zeros. Must be a clean 31-bit address or a clean 24-bit address.

48-55
The 4-byte user exit address constant will be passed to the E15, E32 or E35 user exit routines.

Note: The user exit address constant must not be used for a Conventional merge or tape work data set sort application.

56-63
The address of a 256-byte ALTSEQ translation table supplied instead of an ALTSEQ statement, if any; otherwise, all zeros. It can be a 64-bit address, a clean 31-bit address or a clean 24-bit address. You can use this option to override an ALTSEQ installation option. For full override and applicability details, see Appendix B, “Specification/override of DFSORT options,” on page 805.

64-71
The address of a 112-byte work area where ESTAE information is saved, or all zeros if the ESTAE information is not saved. Must be a clean 31-bit address or a clean 24-bit address. If a system or user exit abend occurs, the DFSORT recovery routine will copy the first 112 bytes of the software diagnostic work area (SDWA) into this area before returning to the ESTAE recovery routine.

72-79
The address of the E18 user exit routine that the program has placed in main storage (for example, via LOAD), if any; otherwise, all zeros. Must be a clean 31-bit address or a clean 24-bit address.

Note: This parameter is ignored for a merge application and for a tape work data set sort application.

80-87
The address of the E39 user exit routine that the program has placed in main storage (for example, via LOAD), if any; otherwise, all zeros. Must be a clean 31-bit address or a clean 24-bit address.

Note: This parameter is ignored for a conventional merge application and for a tape work data set sort application.

88-95
Four bytes of zeros followed by 4 characters to be used as an identifier for this call to DFSORT. This field can be used to uniquely identify each call to DFSORT from a program that calls DFSORT more than once. DFSORT prints message ICE200I to display the 4-character identifier exactly as you specify it; the field is not checked for valid characters. If the identifier field contains zeros (X'00000000'), DFSORT does not print message ICE200I.

96-103
8 byte field for the 64-bit address of the user block list parameters area containing the information related to block of records passed between DFSORT and E15/E35 exits, if any; otherwise, all zeros.

104-135
Reserved. Must be set to zeros.

Writing the macro instruction

When writing the LINK, ATTACH, or XCTL macro instruction using the 64-bit parameter list, you must:

- Specify SORT64 (the entry point) in the EP parameter of the instruction. (This applies to sort, merge, and copy applications.)

- Load the address of the pointer to the parameter list into 64-bit register 1 or pass it in the MF parameter of the instruction. The parameter list can be above or below the bar.

When writing the LINK, ATTACH, or XCTL macro instruction using the extended or 24-bit parameter list, you must:

- Specify SORT (the entry point) in the EP parameter of the instruction. (This applies to sort, merge, and copy applications.)
- Load the address of the pointer to the parameter list into register 1 or pass it in the MF parameter of the instruction. The parameter list must be below the bar.

Note: If you are using ATTACH, you might also need the ECB parameter.

If you provide an E15 user exit routine address in the parameter list, DFSORT ignores the SORTIN data set; your E15 routine must pass all input records to DFSORT. The same applies to a merge if you specify an E32 routine address. This means that your routine must issue a return code of 12 (insert record) until the input data set is complete, and then a return code of 8 ("do not return").

DFSORT ignores the SORTOUT data set if you provide an E35 routine address in the parameter list. Unless you use OUTFIL processing, your routine is then responsible for disposing of all output records. It must issue a return code of 4 (delete record) for each record in the output data set. When the program has deleted all the records, your routine issues a return code of 8 ("do not return").

When DFSORT is done, it passes control to the routine that invoked it.

When a single task attaches two or more program applications, you must modify the standard ddnames so that they are unique. For ways of doing this, and for the rules of override, see Appendix B, “Specification/override of DFSORT options,” on page 805.

If you ATTACH more than one DFSORT application from the same program, you must wait for each to complete before attaching the next unless DFSORT and your user exits are installed re-entrant.

When you initiate DFSORT via XCTL, you must give special consideration to the area where the parameter list, address list, optional parameters, and modification routines (if any) are stored. This information must not reside in the module that issues the XCTL because the module is overlaid by DFSORT.

There are two ways to overcome this problem. First, the control information can reside in a task that attaches the module that issues the XCTL. Second, the module issuing the XCTL can first issue a GETMAIN macro instruction and place the control information in the main storage area it obtains. This area is not overlaid when the XCTL is issued. The address of the control information in the area must be passed to DFSORT in general register 1.

Parameter list examples

24-Bit Parameter List Example

The example in Figure 29 on page 535 shows, in assembler language, how to use a 24-bit parameter list to code parameters and statement images and how to pass control to DFSORT.

```
        LA   1,PARLST              LOAD ADDR OF PARAM POINTER IN R1
        ATTACH EP=SORT             INVOKE SORT
        .
        .
        .
PARLST  DC   X'80',AL3(ADLST)      POINTER FLAG/ADDRESS OF PARAM LIST
        .
        .
        .
        CNOP 2,4                   ALIGN TO CORRECT BOUNDARY
ADLST   DC   AL2(LISTEND-LISTBEG)  PARAM LIST LENGTH
LISTBEG DC   A(SORTA)              BEGINNING ADDRESS OF SORT STMT
        DC   A(SORTZ)              END ADDRESS OF SORT STMT
        DC   A(RECA)               BEGINNING ADDR OF RECORD STMT
        DC   A(RECZ)               END ADDR OF RECORD STMT
        DC   A(MOD1)               ADDR OF E15 RTN
        DC   A(MOD2)               ADDR OF E35 RTN
        DC   C'ABC#'               DDNAME CHARACTERS
        DC   F'720000'             OPTIONAL MAIN STORAGE VALUE
        DC   X'FF'                 MESSAGE OPTION FLAG BYTE
        DC   C'(U)'                MESSAGE OPTION
LISTEND EQU  *
SORTA   DC   C' SORT FIELDS=(10,15,CH,A),'   SORT CONTROL STMT
        DC   C'FILSZ=E4780'          (CONTINUED)
SORTZ   DC   C' '                  DELIMITER
RECA    DC   C' RECORD LENGTH=100,TYPE=F'    RECORD CONTROL STMT
RECZ    DC   C' '                  DELIMITER
        DS   0H
        USING *,15
MOD1    (routine for E15 user exit)
        .
        .
        USING *,15
MOD2    (routine for E35 user exit)
```

Figure 29. Coding a 24-Bit Parameter List

Extended Parameter List Example

The example in Figure 30 on page 536 shows, in assembler language, how to use an extended parameter list to code parameters and statement images and how to pass control to DFSORT.

```
         .
         .
         .
         LA   R1,PL1            SET ADDRESS OF PARAMETER LIST
*                                TO BE PASSED TO SORT/MERGE
         ST   R2,PL4            SET ADDRESS OF GETMAINED AREA
*                                TO BE PASSED TO E15
         LINK EP=SORT           INVOKE SORT/MERGE
         .
         .
         .
PL1      DC   A(CTLST)          ADDRESS OF CONTROL STATEMENTS
PL2      DC   A(E15)            ADDRESS OF E15 ROUTINE
PL3      DC   A(0)              NO E35 ROUTINE
PL4      DS   A                 USER EXIT ADDRESS CONSTANT
PL5      DC   F'-1'             INDICATE END OF LIST
CTLST    DS   0H                CONTROL STATEMENTS AREA
         DC   AL2(CTL2-CTL1)    LENGTH OF CHARACTER STRING
CTL1     DC   C' SORT FIELDS=(4,5,CH,A)'
         DC   C' OPTION '
         DC   C'RESINV=2048,FILSZ=E25000,MSGDDN=MSGOUT '
         DC   C' OMIT COND=(5,8,EQ,13,8),FORMAT=FI '
         DC   C' RECORD TYPE=F,LENGTH=80 '
CTL2     EQU  *
OUT      DCB  DDNAME=SYSOUT,... MYSORT USES SYSOUT
E15      DS   0H                E15 ROUTINE
         .
         .
         .
         BR   R14               RETURN TO SORT/MERGE
* MAPPING OF PARAMETER LIST PASSED TO E15 FROM SORT/MERGE
SRTLST DS   A                   ADDRESS OF RECORD
GMA      DS   A                 ADDRESS OF AREA GETMAINED BY
*                                MYSORT
         .
         .
         .
```

Figure 30. Coding an Extended Parameter List

64-Bit Parameter List Example

"Example 15. Sort with 64-bit parameter lists, E15, E35 and OUTFIL" on page 785 in Chapter 11, "Examples of DFSORT job streams," on page 765 shows, in assembler language, how to use a 64-bit parameter list to code parameters and statement images and how to pass control to DFSORT.

Restrictions for dynamic invocation

Merge restriction

Merge applications cannot be done when DFSORT is invoked from a PL/I program.

Copy restrictions

Copy applications cannot be done when DFSORT is invoked from a PL/I program.

If you invoke DFSORT from a COBOL program, the following restrictions apply:

- The OPTION COPY statement can be placed in either the COBOL IGZSRTCD data set or the DFSORT SORTCNTL or DFSPARM data set.
- If using the FASTSRT compiler option for any part or all of the COBOL SORT statement, a copy application can be done.
- If using the COBOL MERGE statement, a copy application cannot be done.

See "COBOL requirements for copy processing" on page 500 for user exit requirements.

Chapter 7. Using ICETOOL

Overview

This chapter describes ICETOOL, a multi-purpose DFSORT utility. ICETOOL uses the capabilities of DFSORT to perform multiple operations on one or more data sets in a single job step. These operations include the following:

- Creating multiple copies of sorted, merged, edited, or unedited input data sets
- Creating output data sets containing subsets of input data sets based on various criteria for character and numeric field values, the number of times unique values occur, header records, trailer records or relative record numbers
- Sorting records between headers and trailers
- Creating output data sets containing different field arrangements of input data sets
- Resizing fixed length records by creating a larger record from multiple shorter records, or multiple shorter records from a larger record
- Creating list data sets showing character and numeric fields in a variety of simple, tailored, and sectioned report formats, allowing control of title, date, time, page numbers, carriage control characters, headings, lines per page, field formats, and total, maximum, minimum, average and count values for the columns of numeric data
- Printing messages that give statistical information for selected numeric fields such as minimum, maximum, average, total, count of values, and count of unique values
- Printing messages that identify invalid decimal values
- Printing messages that give record counts
- Setting RC=12, RC=8, RC=4, or RC=0 based on record counts
- Creating an output data set containing an output record with text and the record count
- Creating a list data set showing the DFSORT installation defaults in use
- Creating list data sets showing unique values for selected character and numeric fields and the number of times each occurs, in a variety of simple and tailored report formats
- Creating list and output data sets for records with: duplicate values, non-duplicate values, or values that occur n times, less than n times or more than n times
- Creating output data sets with information spliced together from two or more input records with duplicate values. The information in the input records can originate from different data sets, helping you to perform various file "join" and "match" operations.
- Using three different modes (stop, continue, and scan) to control error checking and actions after error detection for groups of operators.

You can use ICETOOL for SORT, MERGE and COPY operations.

ICETOOL/DFSORT relationship

ICETOOL is a batch front-end utility that uses the capabilities of DFSORT to perform the operations you request.

ICETOOL includes 17 operators that perform sort, merge, copy, statistical, report, selection, and splice operations. Most of the operations performed by ICETOOL require only simple JCL and operator statements. Some ICETOOL operations require or allow you to specify complete DFSORT control statements (such as SORT, MERGE, INCLUDE or OMIT, INREC, and OUTFIL) to take full advantage of DFSORT's capabilities.

ICETOOL automatically calls DFSORT with the particular DFSORT control statements and options required for each operation (such as DYNALLOC for sorting).

ICETOOL also produces messages and return codes describing the results of each operation and any errors detected. Although you generally do not need to look at the DFSORT messages produced as a result of an ICETOOL run, they are available in a separate data set if you need them.

ICETOOL can be called directly or from a program. ICETOOL allows operator statements (and comments) to be supplied in a data set or in a parameter list passed by a calling program. For each operator supplied in the parameter list, ICETOOL puts information in the parameter list pertaining to that operation, thus allowing the calling program to use the information derived by ICETOOL.

ICETOOL JCL summary

The JCL statements used with ICETOOL are summarized in the following. See "Job control language for ICETOOL" on page 544 for more detailed information. See also "JCL restrictions" on page 546 and "ICETOOL notes and restrictions" on page 683.

//JOBLIB DD

Defines your program link library if it is not already known to the system.

//STEPLIB DD

Same as //JOBLIB DD

//TOOLMSG DD

Defines the ICETOOL message data set for all operations.

//DFSMSG DD

Defines the DFSORT message data set for all operations.

//SYMNAMES DD

Defines the SYMNAMES data set containing statements to be used for symbol processing.

//SYMNOUT DD

Defines the data set in which SYMNAMES statements and the symbol table are to be listed.

//TOOLIN DD

Contains ICETOOL control statements.

//indd DD
Defines an input data set for a COPY, COUNT, DATASORT, DISPLAY, MERGE, OCCUR, RANGE, RESIZE, SELECT, SORT, SPLICE, STATS, SUBSET, UNIQUE, or VERIFY operation.

//outdd DD
Defines an output data set for a COPY, DATASORT, MERGE, RESIZE, SELECT, SORT, SPLICE, or SUBSET operation.

//savedd DD
Defines an output data set for a SELECT or SUBSET operation.

//listdd DD
Defines a list data set for a DEFAULTS, DISPLAY, or OCCUR operation.

//countdd DD
Defines an output data set for a COUNT operation

//xxxxCNTL DD
Contains DFSORT control statements for a COPY, COUNT, DATASORT, MERGE, RESIZE, SELECT, SORT, SPLICE or SUBSET operation.

ICETOOL operator summary

ICETOOL has 17 operators that are used to perform a variety of functions. The functions of these operators are summarized later in this section. See "ICETOOL statements" on page 546 for more detailed information. Additionally, information pertaining to each operator is provided to calling programs that supply statements to ICETOOL using a parameter list. See "Parameter list interface" on page 679 for details.

COPY
Copies a data set to one or more output data sets.

COUNT
Prints a message containing the count of records in a data set. COUNT can also be used to create an output data set containing text and the count, or to set RC=12, RC=8, RC=4, or RC=0 based on meeting criteria for the number of records in a data set.

DATASORT
Sorts data records between header and trailer records in a data set to an output data set.

DEFAULTS
Prints the DFSORT installation defaults in a separate list data set.

DISPLAY
Prints the values or characters of specified numeric or character fields in a separate list data set. Simple, tailored, or sectioned reports can be produced. Maximums, minimums, totals, averages and counts can be produced.

MERGE
Merges one or more data sets to one or more output data sets.

MODE
Three modes are available that can be set or reset for groups of operators:

- STOP mode (the default) stops subsequent operations if an error is detected
- CONTINUE mode continues with subsequent operations if an error is detected
- SCAN mode allows ICETOOL statement checking without actually performing any operations.

OCCUR
Prints each unique value for specified numeric or character fields and how many times it occurs in a separate list data set. Simple or tailored reports can be produced. The values printed can be limited to those for which the value count meets specified criteria (for example, only duplicate values or only non-duplicate values).

RESIZE
Creates a larger record from multiple shorter records, or creates multiple shorter records from a larger record, that is, resizes fixed length records.

RANGE
Prints a message containing the count of values in a specified range for a specified numeric field in a data set.

SELECT
Selects records from a data set for inclusion in an output data set based on meeting criteria for the number of times specified numeric or character field values occur (for example, only duplicate values or only non-duplicate values). Records that are not selected can be saved in a separate output data set.

SORT
Sorts a data set to one or more output data sets.

SPLICE
Splices together specified fields from records that have the same specified numeric or character field values (that is, duplicate values), but different information. Specified fields from two or more records can be combined to create an output record. The fields to be spliced can originate from records in different data sets, so you can use SPLICE to do various "join" and "match" operations.

STATS
Prints messages containing the minimum, maximum, average, and total for specified numeric fields in a data set.

SUBSET
Selects records from a data set based on keeping or removing header records (the first n records), relative records, or trailer records (the last n records). Records that are not selected can be saved in a separate output data set.

UNIQUE
Prints a message containing the count of unique values for a specified numeric or character field.

VERIFY
Examines specified decimal fields in a data set and prints a message identifying each invalid value found for each field.

Restriction: You can perform a JOINKEYS application with the COPY and SORT operators, but not with the other operators.

Complete ICETOOL examples

"ICETOOL example" on page 794 contains a complete ICETOOL sample job with all required JCL and control statements. The following example shows the JCL and control statements for a simple ICETOOL job.

```
//EXAMP JOB A402,PROGRAMMER
//RUNIT EXEC PGM=ICETOOL,REGION=1024K
//TOOLMSG DD SYSOUT=A
//DFSMSG DD SYSOUT=A
//TOOLIN DD *
* Show installation defaults
  DEFAULTS LIST(SHOWDEF)
* Create three copies of a data set
  COPY FROM(IN1) TO(OUT1,OUT2,OUT3)
* Print a report
  DISPLAY FROM(IN2) LIST(REPORT) DATE TITLE('Monthly Report') PAGE -
    HEADER('Location') ON(1,25,CH) -
    HEADER('Revenue') ON(23,10,FS) -
    HEADER('Profit') ON(45,10,FS) -
    TOTAL('Totals') AVERAGE('Averages') BLANK
* Select all records with duplicate (non-unique) keys
  SELECT FROM(IN2) TO(DUPKEYS) ON(1,25,CH) ALLDUPS -
* Save all records with non-duplicate (unique) keys
  DISCARD (UNQKEYS)
/*
//SHOWDEF DD SYSOUT=A
//IN1 DD DSN=FLY.INPUT1,DISP=SHR
//IN2 DD DSN=FLY.INPUT2,DISP=SHR
//OUT1 DD DSN=FLY.NEW,DISP=OLD
//OUT2 DD DSN=FLY.BU1,DISP=OLD
//OUT3 DD DSN=FLY.BU2,DISP=OLD
//DUPKEYS DD DSN=FLY.DUPS,DISP=OLD
//UNQKEYS DD DSN=FLY.UNQS,DISP=OLD
//REPORT DD SYSOUT=A
```

Figure 31. Simple ICETOOL Job

Using symbols

You can define and use a symbol for any field or constant in the following ICETOOL operators: COUNT, DATASORT, DISPLAY, OCCUR, RANGE, SELECT, SPLICE, STATS, SUBSET, UNIQUE, and VERIFY. You can

also use symbols in the DFSORT control statements you specify for an ICETOOL run. This makes it easy to create and reuse collections of symbols (that is, mappings) representing information associated with various record layouts. You can use system symbols (for example, &JOBNAME.) in your symbol constants. See Chapter 8, “Using symbols for fields and constants,” on page 685 for complete details.

Using SET and PROC symbols in control statements

For jobs that directly invoke ICETOOL (PGM=ICETOOL), you can construct DFSORT symbols that incorporate JCL PROC or SET symbols as well as text and system symbols. You can then use those constructed DFSORT symbols in ICETOOL and DFSORT control statements in the same way you use regular DFSORT symbols. For complete information, see Chapter 8, “Using symbols for fields and constants,” on page 685.

Invoking ICETOOL

ICETOOL can be invoked in the following three ways:

- Directly (that is, not from a program) using the TOOLIN Interface
- From a program using the TOOLIN Interface
- From a program using the Parameter List Interface.

With the TOOLIN Interface, you supply ICETOOL statements in a data set defined by the TOOLIN DD statement. ICETOOL prints messages in the data set defined by the TOOLMSG DD statement.

With the Parameter List Interface, your program supplies ICETOOL statements in a parameter list. ICETOOL prints messages in the data set defined by the TOOLMSG DD statement and also puts information in the parameter list for use by your program.

Putting ICETOOL to use

By using various combinations of the 17 ICETOOL operators, you can easily create applications that perform many complex tasks. The two small samples that follow show some things you can do with ICETOOL.

Obtaining various statistics

Table 87. Obtaining Various Statistics

Obtaining Various StatisticsSample code

```
MODE STOP
VERIFY FROM(DATA1) ON(22,7,PD)
DISPLAY FROM(DATA1) LIST(SALARIES) -
  TITLE('Employee Salaries') DATE TIME -
  HEADER('Employee Name') HEADER('Salary') -
  ON(1,20,CH)             ON(22,7,PD)      BLANK -
  AVERAGE('Average Salary')
STATS FROM(DATA1) ON(22,7,PD)
RANGE FROM(DATA1) ON(22,7,PD) LOWER(20000)
RANGE FROM(DATA1) ON(22,7,PD) HIGHER(19999) LOWER(40000)
RANGE FROM(DATA1) ON(22,7,PD) HIGHER(40000)
OCCUR FROM(DATA1) LIST(SALARIES) -
  TITLE('Employees Receiving Each Salary') DATE TIME -
  HEADER('Salary') HEADER('Employee Count') -
  ON(22,7,PD)      ON(VALCNT)               BLANK
```

Assume that you specify DD statements with the following ddnames for the indicated data sets:

DATA1
A data set containing the name, salary, department, location and so on, of each of your employees. The name field is in positions 1 through 20 in character format and the salary field is in positions 22 through 28 in packed decimal format.

SALARIES
A SYSOUT data set.

You can use the ICETOOL operators in Table 87 on page 541 to do the following:

MODE STOP

If an error is found while processing one of the operators, subsequent operators are not processed (that is, each operator is dependent on the success of the previous operator).

VERIFY

Prints error messages in the TOOLMSG data set identifying any invalid values in the packed decimal salary field.

DISPLAY

Prints a report with each employee's name and salary and the average for all employee salaries in the SALARIES list data set.

STATS

Prints messages in the TOOLMSG data set showing the minimum, maximum, average, and total of the individual salaries.

RANGE

The three RANGE operators print messages in the TOOLMSG data set showing the number of salaries below $20,000, from $20,000 to $39,999, and above $40,000.

OCCUR

Prints a report with each unique salary and the number of employees who receive it in the SALARIES list data set.

Creating multiple versions/combinations of data sets

Table 88. Creating Multiple Versions/Combinations of Data Sets

Creating Multiple Versions/Combinations of Data SetsSample code

```
  * GROUP 1
    MODE CONTINUE
    COPY FROM(DATA1) TO(DATA2)
    COPY FROM(MSTR1) TO(MSTR2)
    SELECT FROM(DATA1) TO(SMALLDPT) ON(30,4,CH) LOWER(10)
    UNIQUE FROM(MSTR1) ON(30,4,CH)
  * GROUP 2
    MODE STOP
    COPY FROM(DATA1) TO(TEMP1) USING(NEW1)
    COPY FROM(DATA1) TO(TEMP2) USING(NEW2)
    COPY FROM(DATA1) TO(TEMP3) USING(NEW3)
    SORT FROM(CONCAT) TO(FINALD,FINALP) USING(FINL)
```

Assume that you specify DD statements with the following ddnames for the indicated data sets:

DATA1
A data set containing the name, salary, department, location, and so on, of each of your employees. The department field is in positions 30 through 33 in character format.

MSTR1
Master data set containing only the name and department of each of your employees. The department field is in positions 30 through 33 in character format.

DATA2, MSTR2, and SMALLDPT
Permanent data sets.

NEW1CNTL
A data set containing DFSORT control statements to INCLUDE employees in department X100 and change the records to match the format of MSTR1.

NEW2CNTL
Same as NEW1CNTL but for department X200.

NEW3CNTL
Same as NEW1CNTL but for department X300.

TEMP1, TEMP2, and TEMP3
Temporary data sets.

FINLCNTL
A data set containing a DFSORT control statement to sort by department and employee name.

CONCAT
A concatenation of the TEMP1, TEMP2, TEMP3, and MSTR1 data sets.

FINALD
A permanent data set.

FINALP
A SYSOUT data set.

You can use the ICETOOL operators in Table 88 on page 542 to do the following:

MODE CONTINUE

If an error is found while processing any of the group 1 operators, subsequent group 1 operators are still processed; that is, group 1 operators are not dependent on the success of the previous group 1 operators.

COPY

The two copy operators create backup copies of DATA1 and MSTR1.

SELECT

Creates a permanent output data set containing the name, salary, department, location, and so on, of each employee in departments with less than 10 people.

UNIQUE

Prints a message in the TOOLMSG data set showing the number of unique departments.

MODE STOP

If an error is found while processing one of the group 2 operators, subsequent group 2 operators are not processed; that is, each group 2 operator is dependent on the success of previous group 2 operators.

COPY

The three COPY operators create an output data set for the employees in each department containing only name and department. Note that the ddname requested by the USING(xxxx) operand is xxxxCNTL. For example, USING(NEW1) requests ddname NEW1CNTL.

SORT

Sorts the three output data sets created by the COPY operators along with the master name/department data set and creates permanent and SYSOUT data sets containing the resulting sorted records.

You can combine both of these examples into a single ICETOOL job step.

Job control language for ICETOOL

An overview of the job control language (JCL) statements for ICETOOL is in Table 89 on page 544 followed by discussions of each ICETOOL DD statement and the use of reserved DD statements and ddnames.

Table 89. JCL Statements for ICETOOL

JCL Statements for ICETOOLJCL statements

//EXAMPL JOB ...
//* ICETOOL CAN BE CALLED DIRECTLY OR FROM A PROGRAM
//STEP EXEC PGM=ICETOOL (or PGM=program_name)
//* THE FOLLOWING DD STATEMENTS ARE ALWAYS REQUIRED
//TOOLMSG DD SYSOUT=A (or DSN=...)
//DFSMSG DD SYSOUT=A
//* THE FOLLOWING DD STATEMENTS ARE USED FOR SYMBOL PROCESSING
//* SYMNAMES DD ...
//* SYMNOUT DD SYSOUT=A (OR DSN=...)
//* THE TOOLIN DD STATEMENT IS ONLY REQUIRED IF THE TOOLIN
//* INTERFACE IS USED.
//TOOLIN DD *
ICETOOL statements
/*
//* THE FOLLOWING DD STATEMENTS ARE ONLY REQUIRED IF SPECIFIED
//* IN ICETOOL STATEMENTS.
//indd DD ...
.
.
.
//outdd DD ...
.
.
.
//listdd DD SYSOUT=A (or DSN=...)
.
.
.
//countdd DD ...
.
.
.
//xxxxCNTL DD *
DFSORT control statements
/*
.
.
.

TOOLMSG DD Statement

Defines the ICETOOL message data set for all operations. ICETOOL messages and statements appear in this data set. ICETOOL uses RECFM=FBA, LRECL=121 and the specified BLKSIZE for the TOOLMSG data set. If the BLKSIZE you specify is not a multiple of 121, ICETOOL uses BLKSIZE=121. If you do not specify the BLKSIZE, ICETOOL selects the block size as directed by the SDBMSG installation option (see *z/OS DFSORT Installation and Customization*).

The TOOLMSG DD statement *must* be present.

DFSMSG DD Statement
Defines the DFSORT message data set for all operations. The DFSORT messages and control statements from all ICETOOL calls to DFSORT appear in this data set. Refer to the discussion of SYSOUT in "System DD statements" on page 63 for details.

Either a DFSMSG DD statement or an SSMSG DD statement *must* be present. If both are present, ICETOOL uses DFSMSG as the DFSORT message data set. If a DFSMSG DD is not present, but an SSMSG DD is present, ICETOOL uses SSMSG as the DFSORT message data set.

Note: A SYSOUT data set should be used for the DFSORT message data set (for example, `//DFSMSG DD SYSOUT=*`). If you define the DFSORT message data set as a temporary or permanent data set, you will only see the DFSORT messages from the last call to DFSORT, unless you allocate a new data set using a disposition of MOD.

//SYMNAMES DD
Defines the SYMNAMES data set containing statements to be used for symbol processing. See Chapter 8, "Using symbols for fields and constants," on page 685 for complete details.

//SYMNOUT DD
Defines the data set in which SYMNAMES statements and the symbol table are to be listed. See Chapter 8, "Using symbols for fields and constants," on page 685 for complete details.

TOOLIN DD statement
Defines the ICETOOL statement data set that must have the following attributes: RECFM=F or RECFM=FB and LRECL=80.

If the TOOLIN Interface is used, the TOOLIN DD statement must be present. If the Parameter List Interface is used, the TOOLIN DD statement is not required and is ignored if present.

indd DD Statement
Defines an input data set for an operation. Refer to "SORTIN DD statement" on page 66 (copy or sort) or "SORTINnn DD statement" on page 68 (merge) for details. ICETOOL imposes the additional restriction that the LRECL of this data must be at least 4.

An indd DD statement must be present for each unique indd name specified in each FROM operand.

outdd DD Statement
Defines an output data set for a COPY, DATASORT, MERGE, RESIZE, SELECT, SORT, SPLICE or SUBSET operation. Refer to "SORTOUT and OUTFIL DD statements" on page 71 for details.

An outdd DD statement must be present for each unique outdd name specified in each TO operand.

savedd DD Statement
Defines an output data set for a SELECT or SUBSET operation. Refer to "SORTOUT and OUTFIL DD statements" on page 71 for details.

A savedd DD statement must be present for each unique savedd name specified in each DISCARD operand.

listdd DD Statement
Defines the list data set for a DEFAULTS, DISPLAY, or OCCUR operation. For each listdd data set, ICETOOL uses RECFM=FBA (or RECFM=FB for DISPLAY or OCCUR with the NOCC operand specified), LRECL=121 (for DEFAULTS) or the LRECL specified in the WIDTH operand or calculated as needed if WIDTH is not specified (DISPLAY and OCCUR), and the specified block size. If the BLKSIZE you specify is not a multiple of the LRECL, ICETOOL uses BLKSIZE=LRECL. If you do not specify BLKSIZE, ICETOOL selects the block size as directed by the LISTSDB or LISTNOSDB option if specified, or otherwise as directed by the SDBMSG installation option (see *z/OS DFSORT Installation and Customization*).

A listdd DD statement must be present for each unique listdd name specified in each LIST operand.

countdd DD Statement
Defines the output data set for a COUNT operation. For each countdd data set, ICETOOL uses RECFM=FB, the LRECL specified in the WIDTH operand or calculated if WIDTH is not specified, and the specified block size. If the BLKSIZE you specify is not a multiple of the LRECL, ICETOOL uses

BLKSIZE=LRECL. If you do not specify BLKSIZE, ICETOOL uses the system-determined optimum block size.

A countdd DD statement must be present for each unique countdd name specified in each WRITE operand.

xxxxCNTL DD Statement
Defines the DFSORT control statement data set for a COPY, COUNT, DATASORT, MERGE, RESIZE, SELECT, SORT, SPLICE or SUBSET operation. Refer to "SORTCNTL DD statement" on page 73 for more details.

An xxxxCNTL DD statement must be present for each unique xxxx specified in each USING operand.

JCL restrictions

You should avoid using ddnames reserved for ICETOOL and DFSORT in ICETOOL operands (FROM, TO, LIST, DISCARD, WRITE). In general, you should also avoid supplying DD statements with ddnames reserved for DFSORT when using ICETOOL because doing so can cause unpredictable results. Specifically:

- SORTCNTL should not be used as a ddname in ICETOOL operators nor should it be supplied as a DD statement. A xxxxCNTL DD statement should only be supplied when you specify a USING(xxxx) operand. xxxx must be four characters that are valid in a ddname of the form xxxxCNTL. xxxx must not be SYSx.
- SYSIN, SORTCNTL, SORTIN, SORTOUT, SORTINnn, and xxxxINnn (where xxxx is specified in a USING operand) should not be used as ddnames in ICETOOL operators nor supplied as DD statements.
- TOOLMSG, DFSMSG, SSMSG, SYMNAMES, SYMNOUT, TOOLIN, SYSUDUMP, and SYSABEND should not be used as ddnames in ICETOOL operators.
- In general, xxxxWKdd ddnames should not be used as ddnames in ICETOOL operators nor supplied as DD statements. However, if you want to override dynamic allocation of work data sets for ICETOOL operators OCCUR and UNIQUE, you can use SORTWKdd DD statements for that purpose. If you want to override dynamic allocation of work data sets for ICETOOL operators DATASORT, RESIZE, SELECT, SORT, SPLICE, and SUBSET, you can use xxxxWKdd DD statements for that purpose in conjunction with the USING operand.
- DFSPARM (or the ddname specified for the PARMDDN installation option) should not be used as a ddname in ICETOOL operators. It should only be used as a DD statement to override DFSORT options for all operators, if appropriate. Refer to "DFSPARM DD statement" on page 74 for details.
- xxxxOFdd (where xxxx is specified in a USING operand) is required as the ddname when an OUTFIL statement in the xxxxCNTL data set specifies FILES=dd. To avoid this requirement, use the FNAMES=ddname operand rather than the FILES=dd operand in OUTFIL statements, and include a DD statement for the specified ddname. See "OUTFIL control statements" on page 221 for details of the FNAMES operand.

You should not use different DDs for a data set to be used for output and then input in the same step, if that data set can be extended to a second or subsequent volume, because that will result in incorrect output. See "Data set notes and limitations" on page 13 for more information.

ICETOOL statements

Each operation must be described to ICETOOL using an operator statement. Additionally, ICETOOL allows comment statements and blank statements. An explanation of the general rules for coding ICETOOL statements is given later in this section followed by a detailed discussion of each operator.

General coding rules

The general format for all ICETOOL operator statements is:

- OPERATOR operand ... operand

where each operand consists of KEYWORD(parameter, parameter...) or just KEYWORD. Any number of operators can be specified and in any order.

The following rules apply for operator statements:

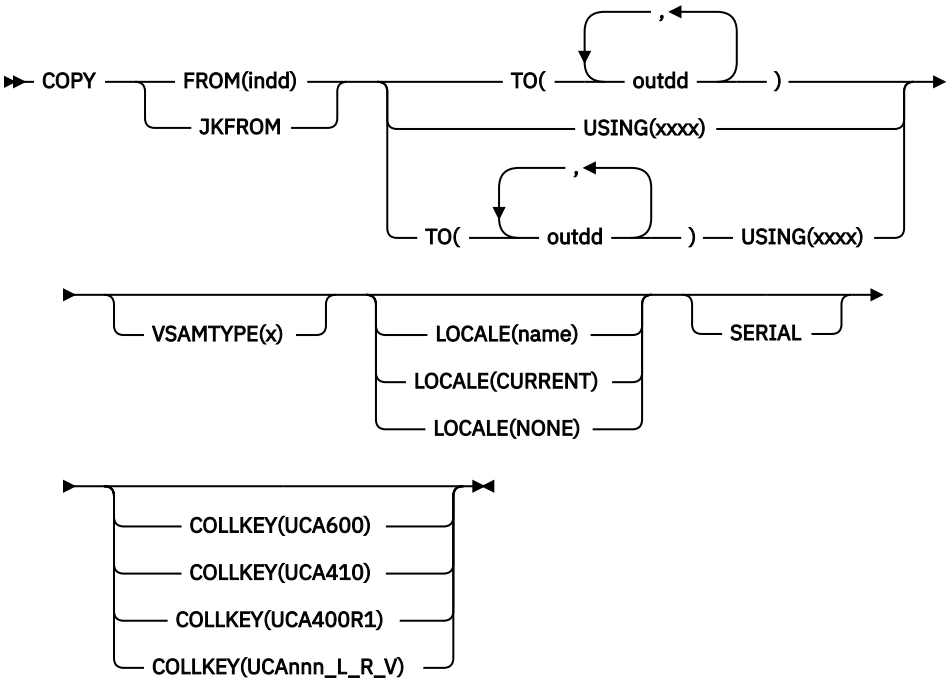
- The operator and operands must be in uppercase EBCDIC.
- The operator must be specified first.
- One blank is required between the operator and the first operand.
- One blank is required between operands.
- Any number of blanks can be specified before or after the operator or any operand, but blanks cannot be specified anywhere else except within quoted character strings.
- Parentheses must be used where shown. Commas or semicolons must be used where commas are shown.
- Operands can be in any order.
- Columns 1-72 are scanned; columns 73-80 are ignored.
- Continuation can be indicated by a hyphen (-) **after** the operator or after any operand. The next operand must then be specified on the next line. For example:

```
SORT FROM(INDD) -
   USING(ABCD) -
   TO(OUTPUT1,OUTPUT2,OUTPUT3)
```

  Any characters specified after the hyphen are ignored. Each operand **must** be completely specified on one line.

A statement with an asterisk (*) in column 1 is treated as a comment statement. It is printed with the other ICETOOL statements, but otherwise not processed. A statement with blanks in columns 1 through 72 is treated as a blank statement. It is ignored, because ICETOOL prints blank lines where appropriate.

COPY operator



Copies an input data set to one or more output data sets.

DFSORT is called to copy the indd data set to the outdd data sets; the DFSORT control statements in xxxxCNTL are used if USING(xxxx) is specified. You can use DFSORT control statements and options in

the xxxxCNTL data set to copy a subset of the input records (INCLUDE or OMIT statement; SKIPREC and STOPAFT options; OUTFIL INCLUDE, OMIT, SAVE, STARTREC, ENDREC, SAMPLE, SPLIT, SPLITBY, and SPLIT1R operands; user exit routines), reformat records for output (INREC, OUTREC and OUTFIL statements, user exit routines), and so on.

If an INCLUDE or OMIT statement or an OUTFIL INCLUDE or OMIT operand is specified in the xxxxCNTL data set, the active locale's collating rules affect INCLUDE and OMIT processing, as explained in the "Cultural Environment Considerations" discussion in “INCLUDE control statement” on page 91.

Operand descriptions

The operands described in this section can be specified in any order.

FROM(indd)
Specifies the ddname of the input data set to be read by DFSORT for this operation. An indd DD statement must be present and must define an input data set that conforms to the rules for DFSORT's SORTIN data set.

Refer to “JCL restrictions” on page 546 for more information regarding the selection of ddnames.

JKFROM
Specifies you are using a JOINKEYS application with this COPY operator to copy the joined records. You must provide a USING(xxxx) operand. In xxxxCNTL, you must provide a JOINKEYS statement with F1=ddname1 for the F1 file and a JOINKEYS statement with F2=ddname2 for the F2 file, as well as JOIN and REFORMAT statements as needed.

TO(outdd,...)
Specifies the ddnames of the output data sets to be written by DFSORT for this operation. From 1 to 10 outdd names can be specified. An outdd DD statement must be present for each outdd name specified. If a single outdd data set is specified, DFSORT is called once to copy the indd data set to the outdd data set, using SORTOUT processing; the outdd data set must conform to the rules for DFSORT's SORTOUT data set. If multiple outdd data sets are specified and SERIAL is not specified, DFSORT is called once to copy the indd data set to the outdd data sets, using OUTFIL processing; the outdd data sets must conform to the rules for DFSORT's OUTFIL data sets.

TO and USING can both be specified. If USING is not specified, TO must be specified. If TO is not specified, USING must be specified.

A ddname specified in the FROM operand must not also be specified in the TO operand.

Refer to “JCL restrictions” on page 546 for more information regarding the selection of ddnames.

USING(xxxx)
Specifies the first 4 characters of the ddname for the control statement data set to be used by DFSORT for this operation. xxxx must be four characters that are valid in a ddname of the form xxxxCNTL. xxxx must not be SYSx.

If USING is specified, an xxxxCNTL DD statement must be present and the control statements in it must conform to the rules for DFSORT's SORTCNTL data set.

TO and USING can both be specified. If USING is not specified, TO must be specified. If TO is not specified, USING must be specified and the xxxxCNTL data set must contain either one or more OUTFIL statements or a MODS statement for an E35 routine that disposes of all records. Other statements are optional.

Refer to “JCL restrictions” on page 546 for more information regarding the selection of ddnames.

VSAMTYPE(x)
Specifies the record type for a VSAM input data set. x must be either F for fixed-length record processing or V for variable-length record processing.

If VSAMTYPE(x) is specified, ICETOOL will pass a RECORD TYPE=x control statement to DFSORT. (If you specify a RECORD TYPE=x statement in the xxxxCNTL data set, it will override the one passed by ICETOOL.)

For complete information on record type processing for VSAM input, see "RECORD control statement" on page 418.

LOCALE(name)
Specifies that locale processing is to be used and designates the name of the locale to be made active during DFSORT processing. LOCALE(name) can be used to override the LOCALE installation option. For complete details on LOCALE(name), see the discussion of the LOCALE operand in "OPTION control statement" on page 173.

LOCALE(CURRENT)
Specifies that locale processing is to be used, and the current locale active when DFSORT is entered will remain the active locale during DFSORT processing. LOCALE(CURRENT) can be used to override the LOCALE installation option. For complete details on LOCALE(CURRENT), see the discussion of the LOCALE operand in "OPTION control statement" on page 173.

LOCALE(NONE)
Specifies that locale processing is not to be used. DFSORT will use the binary encoding of the code page defined for your data for collating and comparing. LOCALE(NONE) can be used to override the LOCALE installation option. For complete details on LOCALE(NONE), see the discussion of the LOCALE operand in "OPTION control statement" on page 173.

SERIAL
Specifies that OUTFIL processing is not to be used when multiple outdd data sets are specified. DFSORT is called multiple times and uses SORTOUT processing; the outdd data sets must conform to the rules for DFSORT's SORTOUT data set. SERIAL is not recommended because the use of serial processing (that is, multiple calls to DFSORT) instead of OUTFIL processing can degrade performance and imposes certain restrictions as detailed later in this section. SERIAL is ignored if a single outdd data set is specified.

DFSORT is called to copy the indd data set to the first outdd data set using the DFSORT control statements in the xxxxCNTL data set if USING(xxxx) is specified. If the first copy is successful, DFSORT is called as many times as necessary to copy the first outdd data set to the second and subsequent outdd data sets. Therefore, for maximum efficiency, use a disk data set as the first in a list of outdd data sets on both disk and tape. If more than one outdd data set is specified, DFSORT must be able to read the *first* outdd data set after it is written in order to copy it to the other outdd data sets. Do not use a SYSOUT or DUMMY data set as the first in a list of outdd data sets because:

- if the first data set is SYSOUT, DFSORT abends when it tries to copy the SYSOUT data set to the second outdd data set.
- if the first data set is DUMMY, DFSORT copies the empty DUMMY data set to the other outdd data sets, with the result that all outdd data sets are then empty.

COLLKEY(UCA600)
This collation version supports the Unicode Standard character suite 6.0.0 and uses Normalization Service under 6.0.0 Unicode character suite.

COLLKEY(UCA410)
This collation version supports the Unicode Standard character suite 4.1.0 and uses Normalization Service under 4.1.0 Unicode character suite.

COLLKEY(UCA400R1)
This collation version supports the Unicode Standard character suite 4.0.0 and uses Normalization Service under 4.0.1 Unicode character suite.

COLLKEY(UCAnnn_L_R_V)
Specifies a collation version, where specific collation rules will modify any of the default Unicode Collation tables specified (UCA400R1, UCA410, or UCA600). Collation versions are set when you specify the following fields:

L
- Language (Specify a language for desired locale.)

R
- Region (Specify a region for desired locale.)

V
- Variant (Specify a variant for desired locale.)

Note:

- For supported collation version settings (Language/Region/Variant), see Appendix E. Locales for collation in *z/OS Unicode Services User's Guide and Reference*
- If there is no collation version information, installation default collation version will be set as default without any change.

Unicode Locales repository data set name SYS1.SCUNLOCL contains a set of locales documented in Locales for collation support. All of those locales contain a section for Collation rules.

Copy examples

This section contains 2 copy examples.

Example 1

```
* Method 1
COPY FROM(MASTER) TO(PRINT,TAPE,DISK)
```

```
* Method 2
COPY FROM(MASTER) TO(DISK,TAPE,PRINT) SERIAL
```

This example shows two different methods for creating multiple output data sets.

Method 1 requires one call to DFSORT, one pass over the input data set, and allows the output data sets to be specified in any order. The COPY operator copies all records from the MASTER data set to the PRINT (SYSOUT), TAPE, and DISK data sets, using OUTFIL processing.

Method 2 requires three calls to DFSORT, three passes over the input data set, and imposes the restriction that the SYSOUT data set must not be the first TO data set. The COPY operator copies all records from the MASTER data set to the DISK data set and then copies the resulting DISK data set to the TAPE and PRINT (SYSOUT) data sets. Because the first TO data set is processed three times (written, read, read), placing the DISK data set first is more efficient than placing the TAPE data set first. PRINT must not be the first in the TO list because a SYSOUT data set cannot be read.

Example 2

```
* Method 1
COPY FROM(IN) TO(DEPT1) USING(DPT1)
COPY FROM(IN) TO(DEPT2) USING(DPT2)
COPY FROM(IN) TO(DEPT3) USING(DPT3)
```

```
* Method 2
COPY FROM(IN) USING(ALL3)
```

This example shows two different methods for creating subsets of an input data set. Assume that:

- The DPT1CNTL data set contains:

  ```
  INCLUDE COND=(5,3,CH,EQ,C'D01')
  ```

- The DPT2CNTL data set contains:

  ```
  INCLUDE COND=(5,3,CH,EQ,C'D02')
  ```

- The DPT3CNTL data set contains:

  ```
  INCLUDE COND=(5,3,CH,EQ,C'D03')
  ```

- The ALL3CNTL data set contains:

```
 OUTFIL FNAMES=DEPT1,INCLUDE=(5,3,CH,EQ,C'D01')
 OUTFIL FNAMES=DEPT2,INCLUDE=(5,3,CH,EQ,C'D02')
 OUTFIL FNAMES=DEPT3,INCLUDE=(5,3,CH,EQ,C'D03')
```

Method 1 requires three calls to DFSORT and three passes over the input data set:

- The first COPY operator copies the records from the IN data set that contain D01 in positions 5-7 to the DEPT1 data set.
- The second COPY operator copies the records from the IN data set that contain D02 in positions 5-7 to the DEPT2 data set.
- The third COPY operator copies the records from the IN data set that contain D03 in positions 5-7 to the DEPT3 data set.

Method 2 accomplishes the same result as method 1, but because it uses OUTFIL statements instead of TO operands, requires only one call to DFSORT and one pass over the input data set.

Example 3

```
 COPY FROM(VSAMIN) TO(VSAMOUT) VSAMTYPE(V)
```

The COPY operator copies all records from the VSAMIN data set to the VSAMOUT data set. The VSAM records are treated as variable-length.

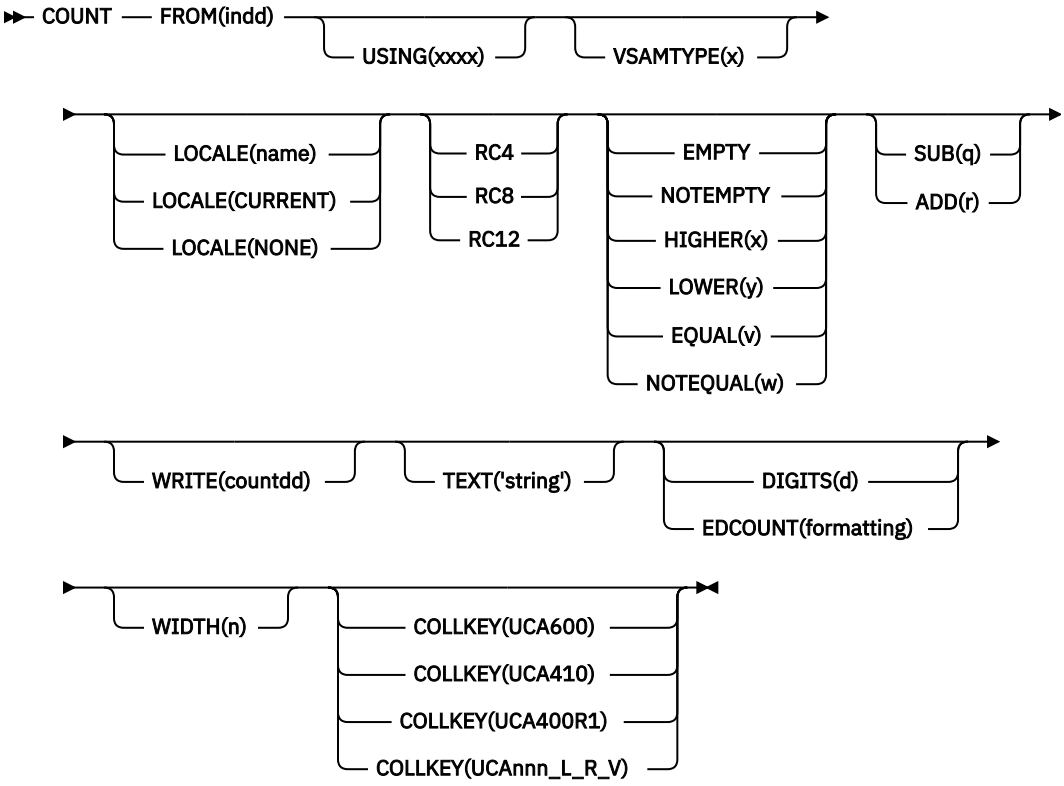
COPY operator with JOINKEYS example

Here is an example of using multiple COPY operators for JOINKEYS applications that preprocess different input files in different ways:

```
//CPYJK EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//IN1 DD DSN=MY.IN1,DISP=SHR
//IN2 DD DSN=MY.IN2,DISP=SHR
//IN3 DD DSN=MY.IN3,DISP=SHR
//OUT1 DD SYSOUT=*
//OUT2 DD SYSOUT=*
//TOOLIN DD *
* First COPY operator with JOINKEYS application.
COPY JKFROM TO(OUT1) USING(CTL1)
* Second COPY operator with JOINKEYS application.
COPY JKFROM USING(CTL2)
/*
//CTL1CNTL DD *
* JOINKEYS application control statements for first COPY operator.
  JOINKEYS F1=IN1,FIELDS=(5,12,A),TASKID=T1
  JOINKEYS F2=IN2,FIELDS=(11,12,A),TASKID=T1
  JOIN UNPAIRED
  REFORMAT FIELDS=(F1:4,40,F2:15,20),FILL=C'$'
* Main task control statements for first COPY operator
* (operates on joined records).
  INCLUDE COND=(8,1,CH,EQ,C'Y')
/*
//CTL2CNTL DD *
* JOINKEYS application control statements for second COPY operator.
  JOINKEYS F1=IN1,FIELDS=(5,12,A),TASKID=T1
  JOINKEYS F2=IN3,FIELDS=(9,12,A),TASKID=T2,SORTED
  REFORMAT FIELDS=(F1:4,40,F2:7,20)
* Main task control statements for second COPY operator
* (operates on joined records).
  OUTFIL FNAMES=OUT2,HEADER1=('Analysis Report'),REMOVECC
/*
//T1F1CNTL DD *
* Control statements for subtask1 (F1=IN1) of both COPY operators.
* Subtask1 sorts/joins on 5,12,A automatically
* per JOINKEYS statement for TASKID=T1/F1=IN1.
  INCLUDE COND=(21,3,CH,EQ,C'J82')
  SUM FIELDS=NONE
/*
//T1F2CNTL DD *
* Control statements for subtask2 (F2=IN2) of first COPY operator.
* Subtask1 sorts/joins on 11,12,A automatically
```

```
* per JOINKEYS statement for TASKID=T1/F2=IN2.
  OPTION SKIPREC=1
  INCLUDE COND=(25,3,CH,EQ,C'J82')
/*
//T2F2CNTL DD *
* Control statements for subtask2 (F2=IN3) of second COPY operator.
* Subtask1 copies/joins on 9,12,A automatically
* per JOINKEYS statement for TASKID=T2/F2=IN3/SORTED.
  INCLUDE COND=(5,3,CH,EQ,C'J82')
/*
```

COUNT operator



Prints a message containing the count of records in a data set. Can also be used to subtract a value from the count or add a value to the count, to create an output data set containing text and the count, or to set RC=12, RC=8, RC=4, or RC=0 based on meeting criteria for the number of records in a data set.

DFSORT is called to copy the indd data set to ICETOOL's E35 user exit. The DFSORT control statements in xxxxCNTL are used if USING(xxxx) is specified. You can use a DFSORT INCLUDE or OMIT statement in the xxxxCNTL data set to count a subset of the input records.

If an INCLUDE or OMIT statement is specified in the xxxxCNTL data set, the active locale's collating rules affect INCLUDE and OMIT processing as explained in the "Cultural Environment Considerations" discussion in “INCLUDE control statement” on page 91.

If EMPTY, NOTEMPTY, HIGHER(x), LOWER(y), EQUAL(v) or NOTEQUAL(w) is not specified, ICETOOL prints a message containing the record count as determined by its E35 user exit.

If EMPTY, NOTEMPTY, HIGHER(x), LOWER(y), EQUAL(v) or NOTEQUAL(w) is specified, ICETOOL checks the record count as determined by its E35 user exit against the specified criteria. If the criteria is met (for example, HIGHER(20) is specified and the record count is 21 or more), ICETOOL sets the following return code for the COUNT operator:

- RC=12 if RC12 is specified, or by default if RC8 and RC4 are not specified
- RC=8 if RC8 is specified
- RC=4 if RC4 is specified

If the criteria is not met (for example, HIGHER(20) is specified and the record count is 20 or less), ICETOOL sets RC=0 for the COUNT operator. ICETOOL uses DFSORT's STOPAFT option to process the minimum number of records required to determine whether or not the criteria is met.

Note: Be sure to check the messages in TOOLMSG when you initially set up any COUNT operators with criteria to make sure that RC=12 is not issued because of syntax errors.

If SUB(q) is specified, ICETOOL subtracts q from the count, but does not reduce the count below 0. If ADD(r) is specified, ICETOOL adds r to the count. SUB(q) and ADD(r) are especially useful for dealing with data sets that contain header and trailer records.

If WRITE(countdd) is specified, ICETOOL writes a record containing the count in the countdd data set. TEXT('string') can be used to insert a string before the count in the output record. DIGITS(n) or EDCOUNT(formatting) can be used to change the appearance of the count in the output record. WIDTH(n) can be used to change the length of the output record.

You must not supply your own DFSORT MODS statement because it would override the MODS statement passed by ICETOOL for this operator.

Note: The record count is also printed for the DISPLAY, OCCUR, RANGE, SELECT, STATS, UNIQUE, and VERIFY operators.

Operand descriptions

The operands described in this section can be specified in any order.

FROM(indd)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

USING(xxxx)
Specifies the first 4 characters of the ddname for the control statement data set to be used by DFSORT for this operation. xxxx must be four characters that are valid in a ddname of the form xxxxCNTL. xxxx must not be SYSx.

If USING is specified, an xxxxCNTL DD statement must be present and the control statements in it:

1. Must conform to the rules for DFSORT's SORTCNTL data set.
2. Should generally be used only for an INCLUDE or OMIT statement or comments statements.

Refer to "JCL restrictions" on page 546 for more information regarding the selection of ddnames.

VSAMTYPE(x)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

LOCALE(name)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

LOCALE(CURRENT)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

LOCALE(NONE)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

RC4
Can be used to set RC=4 for this COUNT operator if the record count meets the specified criteria. RC4 can be specified only if EMPTY, NOTEMPTY, HIGHER(x), LOWER(y), EQUAL(v), or NOTEQUAL(w) is specified. RC4 overrides the default of setting RC=12 for this COUNT operator if the record count meets the specified criteria

RC8
Can be used to set RC=8 for this COUNT operator if the record count meets the specified criteria. RC8 can be specified only if EMPTY, NOTEMPTY, HIGHER(x), LOWER(y), EQUAL(v), or NOTEQUAL(w) is specified. RC8 overrides the default of setting RC=12 for this COUNT operator if the record count meets the specified criteria.

RC12
Can be used to set RC=12 for this COUNT operator if the record count meets the specified criteria. RC12 can be specified only if EMPTY, NOTEMPTY, HIGHER(x), LOWER(y), EQUAL(v), or NOTEQUAL(w) is specified. RC12 is equivalent to the default used when RC4 or RC8 is not specified.

EMPTY
Sets RC=12 (or RC=8 if RC8 is specified, or RC=4 if RC4 is specified) for this COUNT operator if the input data set or subset is empty, or sets RC=0 for this COUNT operator if the input data set or subset is not empty.

EMPTY is equivalent to EQUAL(0).

NOTEMPTY
Sets RC=12 (or RC=8 if RC8 is specified, or RC=4 if RC4 is specified) for this COUNT operator if the input data set or subset is not empty, or sets RC=0 for this COUNT operator if the input data set or subset is empty.

EMPTY is equivalent to NOTEQUAL(0).

HIGHER(x)
Sets RC=12 (or RC=8 if RC8 is specified, or RC=4 if RC4 is specified) for this COUNT operator if the record count is higher than x, or sets RC=0 for this COUNT operator if the record count is equal to or lower than x.

x must be specified as n or +n where n can be 0 to 562949953421310.

LOWER(y)
Sets RC=12 (or RC=8 if RC8 is specified, or RC=4 if RC4 is specified) for this COUNT operator if the record count is lower than y, or sets RC=0 for this COUNT operator if the record count is equal to or higher than y.

y must be specified as n or +n where n can be 0 to 562949953421310.

EQUAL(v)
Sets RC=12 (or RC=8 if RC8 is specified, or RC=4 if RC4 is specified) for this COUNT operator if the record count is equal to v, or sets RC=0 for this COUNT operator if the record count is not equal to v.

v must be specified as n or +n where n can be 0 to 562949953421310.

NOTEQUAL(w)
Sets RC=12 (or RC=8 if RC8 is specified, or RC=4 if RC4 is specified) for this COUNT operator if the record count is not equal to w, or sets RC=0 for this COUNT operator if the record count is equal to w.

w must be specified as n or +n where n can be 0 to 562949953421310.

SUB(q)
Subtracts q from the record count, but does not reduce the count below 0. The resulting modified record count is displayed in the count message in TOOLMSG. If WRITE(countdd) is specified, the modified record count is used in the count record. If EMPTY, NOTEMPTY, HIGHER(x), LOWER(y), EQUAL(v) or NOTEQUAL(w) is specified, the modified record count is used to determine if the criteria is satisfied.

SUB(q) is especially useful for getting the count of just the data records for a data set that contains header, data and trailer records.

q must be specified as n or +n where n can be 1 to 999.

ADD(r)
Adds r to the record count. The resulting modified record count is displayed in the count message in TOOLMSG. If WRITE(countdd) is specified, the modified record count is used in the count record. If EMPTY, NOTEMPTY, HIGHER(x), LOWER(y), EQUAL(v) or NOTEQUAL(w) is specified, the modified record count is used to determine if the criteria is satisfied.

ADD(r) is especially useful for getting the count of header, data and trailer records for a data set that just contains data records to which you want to add header and trailer records.

r must be specified as n or +n where n can be 1 to 999.

WRITE(countdd)
Specifies the ddname of the count data set to be produced by ICETOOL for this operation. A countdd DD statement must be present. ICETOOL sets the attributes of the count data set as follows:

- RECFM is set to FB.
- LRECL is set to one of the following:
  - If WIDTH(n) is specified, LRECL is set to n. Use WIDTH(n) if your count record length and LRECL must be set to a particular value (for example, 80), or if you want to ensure that the count record length does not exceed a specific maximum (for example, 20 bytes).
  - If WIDTH(n) is not specified, LRECL is set to the calculated required record length. If your LRECL does not need to be set to a particular value, you can let ICETOOL determine and set the appropriate LRECL value by not specifying WIDTH(n).
- BLKSIZE is set to one of the following:
  - The BLKSIZE from the DD statement, DSCB, or label, if it is a multiple of the LRECL used.
  - The LRECL if the BLKSIZE from the DD statement, DSCB, or label is not a multiple of the LRECL used.
  - The system determined blocksize if the BLKSIZE is not available from the DD statement, DSCB, or label.

TEXT('string')
Specifies a string to be printed starting in the first byte of the count record. The count follows the string. TEXT can only be specified if WRITE(countdd) is specified.

The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes (''). To suppress printing of a string, do not specify TEXT('string') or specify TEXT('') using two single apostrophes.

DIGITS(d)
Specifies d digits for the count in the output record, overriding the default of 15 digits. d can be 1 to 15. The count is written as d decimal digits with leading zeros. DIGITS can only be specified if WRITE(countdd) is specified.

If you know that your count requires less than 15 digits, you can use a lower number of digits (d) instead by specifying DIGITS(d). For example, if DIGITS(10) is specified, 10 digits are used instead of 15.

If you use DIGITS(d) and the count overflows the number of digits used, ICETOOL terminates the operation. You can prevent the overflow by specifying an appropriately higher d value for DIGITS(d). For example, if DIGITS(5) results in overflow, you can use DIGITS(6) instead.

EDCOUNT(formatting)
Specifies formatting items that indicate how the count in the output record is to be formatted, overriding the default of 15 digits with leading zeros. Formatting items can be specified in any order, but each item can only be specified once. EDCOUNT can only be specified if WRITE(countdd) is specified.

mask
See the discussion of mask under ON(p,m,f,formatting) in "DISPLAY Operator" on page “mask” on page 572.

E'pattern'
specifies an edit pattern to be applied to the count. The pattern (1 to 24 characters) must be enclosed in single apostrophes. Each 9 in the pattern (up to 15) is replaced by a corresponding digit from the count. Characters other than 9 in the pattern appear as specified. To include a single apostrophe (') in the pattern, specify two single apostrophes (''). F'string' or a mask cannot be specified with E'pattern'.

When E'pattern' is specified for the count:

- If the number of significant digits in the count is less than the number of 9's in the pattern, 0's are filled in on the left. For example, 1234 is shown as 001234 with EDCOUNT(E'999999').

- If the number of significant digits in the count is greater than the number of 9's in the pattern, digits are truncated from the left. For example, 1234567 is shown as *4567* with EDCOUNT(E'*9999*').

L'string
See the discussion of L'string' under ON(p,m,f,formatting) in "DISPLAY Operator" on page “L'string'” on page 574

F'string'
See the discussion of F'string' under ON(p,m,f,formatting) in "DISPLAY Operator" on page “F'string'” on page 574

T'string'
See the discussion of T'string' under ON(p,m,f,formatting) in "DISPLAY Operator" on page “T'string'” on page 575

LZ
See the discussion of LZ under ON(p,m,f,formatting) in "DISPLAY Operator" on page “LZ” on page 575

Udd
specifies the number of digits to be used for the count. dd specifies the number of digits and must be a two-digit number between 01 and 15. The default number of digits (d) for the count is 15.

If you know that your count requires less than 15 digits, you can use a lower number of digits (dd) instead by specifying Udd. For example, if EDCOUNT(U09) is specified, 9 digits (from U09) is used instead of 15 (default for the count).

If you use Udd and the count overflows the number of digits used, ICETOOL terminates the operation. You can prevent the overflow by specifying an appropriately higher dd value for Udd. For example, if EDCOUNT(U05) results in overflow, you can use EDCOUNT(U06) instead.

If E'pattern' is specified, Udd is ignored, because d is determined from the pattern.

WIDTH(n)
Specifies the record length and LRECL you want ICETOOL to use for the count data set. n can be from 1 to 32760. WIDTH can only be specified if WRITE(countdd) is specified. ICETOOL always calculates the record length required to write the count record and uses it as follows:

- If WIDTH(n) is specified and the calculated record length is less than or equal to n, ICETOOL sets the record length and LRECL to n. ICETOOL pads the count record on the right with blanks to the record length.
- If WIDTH(n) is specified and the calculated record length is greater than n, ICETOOL issues an error message and terminates the operation.
- If WIDTH(n) is not specified, ICETOOL sets the record length and LRECL to the calculated record length.

Use WIDTH(n) if your count record length and LRECL must be set to a particular value (for example, 80), or if you want to ensure that the count record length does not exceed a specific maximum (for example, 20 bytes). Otherwise, you can let ICETOOL calculate and set the appropriate record length and LRECL by not specifying WIDTH(n).

COLLKEY(UCA600)
See the discussion of this operand on the COPY statement in “COPY operator” on page 547

COLLKEY(UCA410)
See the discussion of this operand on the COPY statement in “COPY operator” on page 547

COLLKEY(UCA400R1)
See the discussion of this operand on the COPY statement in “COPY operator” on page 547

COLLKEY(UCAnnn_L_R_V)
See the discussion of this operand on the COPY statement in “COPY operator” on page 547

COUNT examples

Example 1

For this example, assume that the CTL1CNTL data set contains a DFSORT INCLUDE statement.

```
COUNT FROM(IN1)
COUNT FROM(IN2) USING(CTL1)
```

The first COUNT operator prints a message containing the count of records in the IN1 data set.

The second COUNT operator prints a message containing the count of records included from the IN2 data set.

Example 2

```
COUNT FROM(INPUT1) EMPTY
```

Sets RC=12 if INPUT1 is empty (that is, INPUT1 has no records), or sets RC=0 if INPUT1 is not empty (that is, INPUT1 has at least one record).

Example 3

For this example, assume that the CTL2CNTL data set contains a DFSORT INCLUDE statement.

```
COUNT FROM(INPUT2) HIGHER(50000) RC4 USING(CTL2)
```

Sets RC=4 if more than 50000 records are included from INPUT2, or sets RC=0 if 50000 or less records are included from INPUT2.

Example 4

```
COUNT FROM(IN2) WRITE(CT2) TEXT('Count is ') -
  EDCOUNT(A1,U10) WIDTH(80)
```

Prints a message containing the count of records in the IN2 data set. Writes an 80-byte record with the specified string and an edited count to the CT2 data set. If IN2 contains 3286721 records, the 80-byte output record in CT2 would look like this:

```
Count is      3,286,721
```

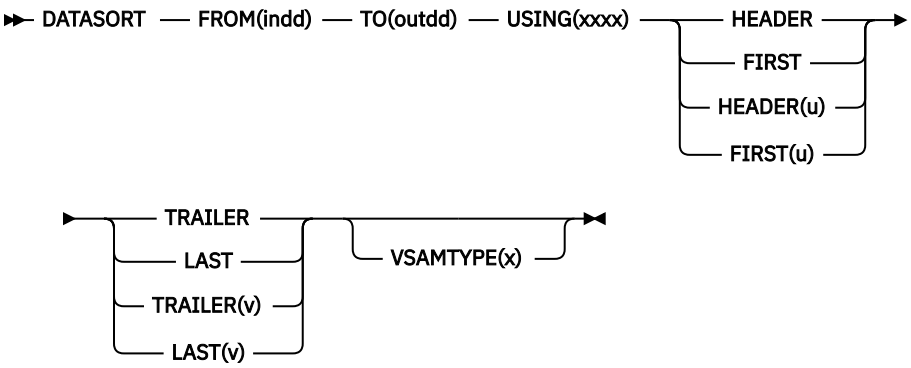
Example 5

```
COUNT FROM(IN3) WRITE(CT3) DIGITS(6) SUB(2)
```

Subtracts 2 from the count of records in the IN3 data set. Prints a message containing the modified count. Writes a 6-byte record with the modified count to the CT3 data set. If IN3 contains 8125 records, the 6-byte output record in CT3 would look like this:

```
008123
```

DATASORT operator



Copies one or more header records and one or more trailer records to the output data set in their original input record order, while sorting the data records between the header and trailer records, using the DFSORT control statements in xxxxCNTL. By definition, the header records are the first n records in the input data set, the trailer records are the last n records in the input data set, and the data records (also called detail records) are the records between the header and trailer records. Thus, the first n records (header records) and last n records (trailer records) are kept in place and the data records between them are sorted.

You must specify one header operand (HEADER, FIRST, HEADER(u) or FIRST(u)), one trailer operand (TRAILER, LAST, TRAILER(v) or LAST(v)), or one header operand and one trailer operand. If you specify a header operand without a trailer operand, only the header records will be kept in place. If you specify a trailer operand without a header operand, only the trailer records will be kept in place. If you specify a header operand and a trailer operand, both the header records and trailer records will be kept in place.

DFSORT is called to copy the header and trailer records and to sort the data records. DFSORT uses its E15 and E35 exits to process the records as needed. ICETOOL passes the EQUALS option to DFSORT to ensure that duplicates are kept in their original input order.

You must supply a DFSORT SORT statement in the xxxxCNTL data set to indicate the control fields for sorting the data records.

You can use additional DFSORT control statements in xxxxCNTL providing you observe these rules:

- A SORT statement must be present
- MODS and OUTREC statements should not be present.
- A STOPAFT operand should not be present.
- Comment statements can be present.
- Header and trailer records will only be affected by the SKIPREC option and OUTFIL statements. SKIPREC=n will remove the first n indd records, so the first header record will be the n+1 indd record. OUTFIL statements will process the header and trailer records in the normal way.
- Data records will be processed by INCLUDE, OMIT, INREC, SUM, OPTION and OUTFIL statements in the normal way.
- If you use INREC to change the length of the data records, DFSORT will preserve the header and trailer records by setting the TO data set LRECL to the maximum of the input or reformatted record length. For fixed-length records, DFSORT will pad the header and trailer records, or data records, on the right with blanks as appropriate.
- You can further process the outdd records **after** DATASORT processing using an OUTFIL statement like this:

  ```
  OUTFIL FNAMES=outdd,...
  ```

  or multiple OUTFIL statements like this:

```
OUTFIL FNAMES=outdd,...
OUTFIL FNAMES=outdd1,...
```

For example, with TO(OUT1) you could further modify the OUT1 records after DATASORT processing, with a statement like this:

```
OUTFIL FNAMES=OUT1,REMOVECC,
  TRAILER1=('Record count ',COUNT=(M11,LENGTH=5))
```

ICETOOL requires extra storage for DATASORT processing, over and above what is normally needed by ICETOOL and DFSORT, in order to save your header and trailer records. The amount of storage needed depends on the number of header and trailer records you specify, and the LRECL of the FROM data set. In most cases, the needed storage can be obtained (above 16MB virtual). However, if ICETOOL cannot get the storage it needs, it issues a message and terminates the DATASORT operation. Increasing the REGION by the amount indicated in the message may allow ICETOOL to run successfully.

The DYNALLOC option is passed to DFSORT to ensure that work space is available for the sort. If your installation defaults for dynamic allocation are inappropriate for a DATASORT operator, you can take one of the following actions:

1. Override the DYNALLOC option using an OPTION control statement such as:

   ```
   OPTION DYNALLOC=(,8)
   ```

   in the xxxxCNTL data set.
2. Use xxxxWKdd DD statements to override the use of dynamic allocation. Refer to “SORTWKdd DD statement” on page 69 for details.

Tape work data sets **cannot** be used with ICETOOL.

Operand descriptions

The operands described in this section can be specified in any order.

FROM(indd)
See the discussion of this operand on the COPY statement in "COPY Operator" on page “FROM(indd)” on page 548.

TO(outdd)
Specifies the ddname of the output data set to be written by DFSORT for this operation.

An outdd DD statement must be present and must define an output data set that conforms to the rules for DFSORT's SORTOUT data set.

The ddname specified in the TO operand must not be the same as the ddname specified in the FROM operand.

Refer to “JCL restrictions” on page 546 for more information.

USING(xxxx)
Specifies the first 4 characters of the ddname for the control statement data set to be used by DFSORT for this operation. xxxx must be four characters that are valid in a ddname of the form xxxxCNTL. xxxx must not be SYSx

An xxxxCNTL DD statement must be present. The xxxxCNTL data set must contain a SORT statement. Other control statements are optional, but if specified must conform to the rules for DFSORT's SORTCNTL data set, and should be used as described for “DATASORT operator” on page 558.

HEADER or FIRST
Specifies one header record (the first record in the indd data set) is to be kept in place.

HEADER and FIRST are equivalent to HEADER(1) and FIRST(1).

HEADER(u) or FIRST(u)
Specifies u header records (the first u records in the indd data set) are to be kept in place.

u must be specified as n or +n where n can be 1 to 1000000.

TRAILER or LAST
Specifies one trailer record (the last record in the indd data set) is to be kept in place.

TRAILER and LAST are equivalent to TRAILER(1) and LAST(1).

TRAILER(v) or LAST(v)
Specifies v trailer records (the last v records in the indd data set) are to be kept in place.

v must be specified as n or +n where n can be 1 to 1000000.

VSAMTYPE(x)
See the discussion of this operand on the COPY statement in "COPY Operator" on page “VSAMTYPE(x)” on page 548.

DATASORT examples

Although the DATASORT operators in the examples in this section could all be contained in a single ICETOOL job step, they are shown and discussed separately for clarity.

Example 1

```
DATASORT FROM(INPUT) TO(OUTPUT) HEADER TRAILER USING(CTL1)
```

This example shows how you can sort the data records between a header record (first record) and a trailer record (last record).

The CTL1CNTL data set contains:

```
  SORT FIELDS=(16,13,CH,A)
```

The input records look like this:

```
MM9999900510100823DDDDD FFFFF 004200806128
AAR   FIRST C  1134341444441     XXXXXXXXX
ATX   SECOND   7777777770111     XXXXXXXXX
ATX   THIRD L  6297132201111     XXXXXXXXX
ATX   FOURTH   2830012906356     XXXXXXXXX
MM9999900510100823DDDDD FFFFF 004
```

We want to keep the MM records in place and sort the other records by the CH field in positions 16-28. We use HEADER and TRAILER to indicate the first and last records should not be sorted. We use the SORT statement to SORT ascending on positions 16-28.

The output records would look like this:

```
MM9999900510100823DDDDD FFFFF 004200806128
AAR   FIRST C  1134341444441     XXXXXXXXX
ATX   FOURTH   2830012906356     XXXXXXXXX
ATX   THIRD L  6297132201111     XXXXXXXXX
ATX   SECOND   7777777770111     XXXXXXXXX
MM9999900510100823DDDDD FFFFF 004
```

Note that we could use FIRST and LAST instead of HEADER and TRAILER.

Example 2

```
DATASORT FROM(IN) TO(OUT) HEADER(2) TRAILER(3) USING(CTL2)
```

This example illustrates how you can sort the data records between header records (first records) and trailer records (last records), and modify just the data records or the header, data and trailer records.

The CTL2CNTL data set contains:

```
  INREC IFTHEN=(WHEN=(24,2,CH,EQ,C'23'),
    OVERLAY=(30:C'Old'))
```

```
  SORT FIELDS=(1,14,CH,A)
  OUTFIL FNAMES=OUT,
    IFTHEN=(WHEN=(24,2,CH,EQ,C'23'),
      OVERLAY=(35:C'First'))
```

The input records look like this:

```
Header 1       2008/04/23
Header 2       2008/04/23
Geometry       2008/04/24
Algebra        2008/04/23
Trigonometry   2008/04/24
Calculus       2008/04/25
Geography      2008/04/25
History        2008/04/23
Trailer 1      2008/04/23
Trailer 2      2008/04/23
Trailer 3      2008/04/23
```

We want to keep the two Header records and the three Trailer records in place and sort the other records by the CH field in positions 1-14. We want to put 'Old' in positions 30-32 of **each data record** (but not the Header or Trailer records) that has '23' in positions 24-25. We want to put 'First' in positions 35-39 of **each record** (Header, data and Trailer) that has '23' in positions 24-25.

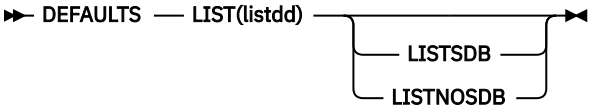
We use HEADER(2) and TRAILER(3) to indicate the first two records and last three records should not be sorted. We use the INREC statement to add 'Old' to data records that have '23' in positions 24-25. INREC applies to the data records, but not to the Header and Trailer records. We use the SORT statement to sort ascending on positions 1-14. We use the OUTFIL statement to add 'First' to Header, data and Trailer records that have '23' in positions 24-25. OUTFIL applies to all of the records.

The output records would look like this:

```
Header 1       2008/04/23         First
Header 2       2008/04/23         First
Algebra        2008/04/23    Old  First
Calculus       2008/04/25
Geography      2008/04/25
Geometry       2008/04/24
History        2008/04/23    Old  First
Trigonometry   2008/04/24
Trailer 1      2008/04/23         First
Trailer 2      2008/04/23         First
Trailer 3      2008/04/23         First
```

Note that we could use FIRST(2) and LAST(3) instead of HEADER(2) and TRAILER(3).

DEFAULTS operator



Prints the DFSORT installation defaults report in a separate list data set.

DFSORT enables you to maintain separate sets of installation defaults for eight different installation environments as follows:

- JCL (ICEAM1) - batch JCL directly invoked installation environment
- INV (ICEAM2) - batch program invoked installation environment
- TSO (ICEAM3) - TSO directly invoked installation environment
- TSOINV (ICEAM4) - TSO program invoked installation environment
- TD1 (ICETD1) - first time-of-day installation environment
- TD2 (ICETD2) - second time-of-day installation environment
- TD3 (ICETD3) - third time-of-day installation environment

- TD4 (ICETD4) - fourth time-of-day installation environment

Each installation default has two or more possible values; DFSORT is shipped with a set of IBM-supplied defaults that can be modified using ICEPRMxx members of PARMLIB or the ICEMAC macro. The DEFAULTS operator provides an easy way to determine the installation default values to be used at run-time. See *z/OS DFSORT Installation and Customization* for a complete discussion of ICEPRMxx members in PARMLIB, the ICEMAC macro, the eight installation environments, and the installation default values.

DEFAULTS produces a three-part report showing:

1. The merged PARMLIB/ICEMAC installation default values for ICEAM1-4 and ICETD1-4 that will be used at run-time.
2. The specified PARMLIB ICEPRMxx member option values for ICEAM1-4 and ICETD1-4 (for reference).
3. The ICEMAC installation default values for ICEAM1-4 and ICETD1-4 (for reference).

The format of the report produced by DEFAULTS varies depending on the defaults selected, but the first part of the report might look like this conceptually:

```
Z/OS DFSORT V2R3  MERGED PARMLIB/ICEMAC DEFAULTS     - p -

   * IBM-SUPPLIED DEFAULT (ONLY SHOWN IF DIFFERENT FROM THE SPECIFIED DEFAULT)

ITEM           JCL (ICEAM1) VALUE       INV (ICEAM2) VALUE       TSO (ICEAM3) VALUE       TSOINV (ICEAM4) VALUE
----------     --------------------     --------------------     --------------------     --------------------
item           value                    value                    value                    value
 .
 .
 .
item           value                    value                    value                    value
item           value                    value                    value                    value
                                        * IBM_value
 .
 .
 .


Z/OS DFSORT V2R3  MERGED PARMLIB/ICEMAC DEFAULTS     - p -

   * IBM-SUPPLIED DEFAULT (ONLY SHOWN IF DIFFERENT FROM THE SPECIFIED DEFAULT)

ITEM           TD1 (ICETD1) VALUE       TD2 (ICETD2) VALUE       TD3 (ICETD3) VALUE       TD4 (ICETD4) VALUE
----------     --------------------     --------------------     --------------------     --------------------
item           value                    value                    value                    value
 .
 .
 .
item           value                    value                    value                    value
               * IBM_value              * IBM_value
item           value                    value                    value                    value
 .
 .
 .
```

The merged PARMLIB/ICEMAC default value for each item is shown as it is set for each of the eight installation environments. For any value that is different from the IBM-supplied value, the IBM-supplied value is shown below it.

The control character occupies the first byte of each record. The title and headings are always printed; p is the page number. The item name column occupies 10 bytes, each of the item value columns occupies 20 bytes, and 5 blanks appear between columns.

Operand descriptions

The operands described in this section can be specified in any order.

LIST(listdd)
Specifies the ddname of the list data set to be produced by ICETOOL for this operation. A listdd DD statement must be present. ICETOOL uses RECFM=FBA, LRECL=121 and the specified BLKSIZE for the list data set. If the BLKSIZE you specify is not a multiple of 121, ICETOOL uses BLKSIZE=121. If you do not specify the BLKSIZE, ICETOOL selects the block size as directed by LISTSDB or LISTNOSDB, if specified, or otherwise as directed by installation option SDBMSG from ICEAM2 or ICEAM4 (see *z/OS DFSORT Installation and Customization*).

Refer to “JCL restrictions” on page 546 for more information regarding the selection of ddnames.

LISTSDB or LISTNOSDB

Can be used to override the SDBMSG value for this LIST data set. LISTSDB directs ICETOOL to select the system-determined optimum block size for the LIST data set in the same way as for installation option SDBMSG=YES. LISTNOSDB directs ICETOOL to select the block size for the LIST data set in the same way as for installation option SDBMSG=NO. See the discussion of the LIST(listdd) operand previously in this section for more information on how LISTSDB or LISTNOSDB affects the LIST data set block size.

Attention: LISTSDB has no effect for SYSOUT data sets (for example, //RPT1 DD SYSOUT=*), because the system-determined optimum block size is not used for spool or dummy data sets.

DEFAULTS example

```
 DEFAULTS LIST(OPTIONS)
```

Prints, in the OPTIONS data set, the DFSORT installation defaults report. The OPTIONS output starts on a new page and might look as follows. The first few items are shown with illustrative values for ICEAM1-4 and ICETD1-4 for each of the three parts of the report.

```
 Z/OS DFSORT V2R3  MERGED PARMLIB/ICEMAC DEFAULTS     - 1 -

    * IBM-SUPPLIED DEFAULT (ONLY SHOWN IF DIFFERENT FROM THE SPECIFIED DEFAULT)

 ITEM           JCL (ICEAM1) VALUE       INV (ICEAM2) VALUE       TSO (ICEAM3) VALUE
TSOINV (ICEAM4) VALUE
 ----------     --------------------     --------------------     --------------------
--------------------
 ENABLE         NONE                     TD1                      NONE                     NONE

 ABCODE         MSG                      99                       MSG                      99
                                         * MSG                                             * MSG
 ALTSEQ         SEE BELOW                SEE BELOW                SEE BELOW                SEE
BELOW
 ARESALL        0                        0                        0                        0
 ARESINV        NOT APPLICABLE           0                        NOT APPLICABLE           0
 CFW            YES                      YES                      YES                      YES
 CHALT          YES                      YES                      NO                       NO
                * NO                     * NO
 CHECK          YES                      YES                      YES                      YES
 CINV           YES                      YES                      YES                      YES
  .
  .
  .
```

```
 Z/OS DFSORT V2R3  MERGED PARMLIB/ICEMAC DEFAULTS     - 4 -

    * IBM-SUPPLIED DEFAULT (ONLY SHOWN IF DIFFERENT FROM THE SPECIFIED DEFAULT)

 ITEM           TD1 (ICETD1) VALUE       TD2 (ICETD2) VALUE       TD3 (ICETD3) VALUE       TD4
(ICETD4) VALUE
 ----------     --------------------     --------------------     --------------------
------------------
 SUN            0600-2000                NONE                     NONE                     NONE
                * NONE
 MON            NONE                     NONE                     NONE                     NONE
 TUE            NONE                     NONE                     NONE                     NONE
 WED            NONE                     NONE                     NONE                     NONE
 THU            NONE                     NONE                     NONE                     NONE
 FRI            NONE                     NONE                     NONE                     NONE
 SAT            0600-2000                NONE                     NONE                     NONE
                * NONE

 ABCODE         99                       MSG                      MSG                      MSG
                * MSG
 ALTSEQ         SEE BELOW                SEE BELOW                SEE BELOW                SEE
BELOW
 ARESALL        0                        0                        0                        0
 ARESINV        0                        0                        0                        0
 CFW            YES                      YES                      YES                      YES
 CHALT          YES                      NO                       NO                       NO
                * NO
 CHECK          YES                      YES                      YES                      YES
 CINV           YES                      YES                      YES                      YES
```

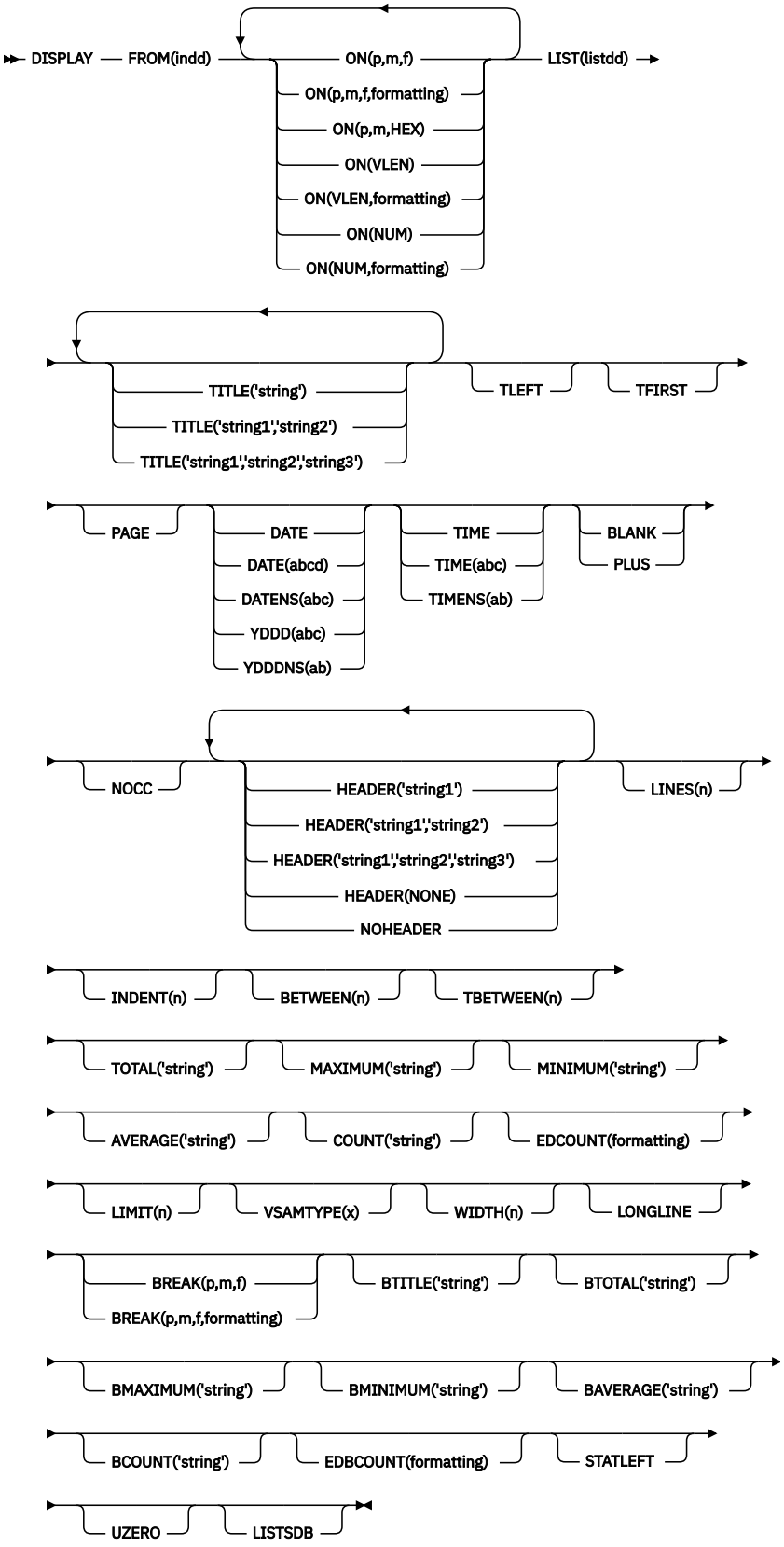
```
  .
  .
  .
 Z/OS DFSORT V2R3  PARMLIB ICEPRMXX MEMBER OPTIONS    - 7 -

 ICEOPT COMMAND IN EFFECT: START ICEOPT,ICEPRM=(03)

 RELEASE:        2.03
 MODULE:         ICEPRML
 APAR LEVEL:     BASE

 COMPILED:       03/21/17
 AREA:           1

 JCL (ICEAM1) OPTIONS

 ITEM           VALUE                                    MEMBER
 ----------     ---------------------------------------  --------
 RESALL         40000                                    ICEPRM03
 CHALT          YES                                      ICEPRM03

 INV (ICEAM2) OPTIONS

 ITEM           VALUE                                    MEMBER
 ----------     ---------------------------------------  --------
 RESINV         40000                                    ICEPRM03
 CHALT          YES                                      ICEPRM03

 TSO (ICEAM3) OPTIONS - NONE

 TSOINV (ICEAM4) OPTIONS - NONE

 TD1 (ICETD1) OPTIONS

 ITEM           VALUE                                    MEMBER
 ----------     ---------------------------------------  --------
 CHALT          YES                                      ICEPRM03

 TD2 (ICETD2) OPTIONS - NONE

 TD3 (ICETD3) OPTIONS - NONE

 TD4 (ICETD4) OPTIONS - NONE
```

```
 Z/OS DFSORT V2R3  ICEMAC DEFAULTS                  - 8 -

 ITEM           JCL (ICEAM1) VALUE       INV (ICEAM2) VALUE       TSO (ICEAM3) VALUE
TSOINV (ICEAM4) VALUE
 ----------     --------------------     --------------------     --------------------
--------------------
 RELEASE        2.01                     2.01                     2.01                     2.01
 MODULE         ICEAM1                   ICEAM2                   ICEAM3
ICEAM4
 APAR LEVEL     BASE                     BASE                     BASE                     BASE
 COMPILED       05/15/11                 05/15/11                 05/08/11
05/15/11

 ENABLE         NONE                     TD1                      NONE                     NONE

 ABCODE         MSG                      99                       MSG                      99
 ALTSEQ         SEE BELOW                SEE BELOW                SEE BELOW                SEE
BELOW
 ARESALL        0                        0                        0                        0
 ARESINV        NOT APPLICABLE           0                        NOT APPLICABLE           0
 CFW            YES                      YES                      YES                      YES
 CHALT          NO                       NO                       NO                       NO
 CHECK          YES                      YES                      YES                      YES
 CINV           YES                      YES                      YES                      YES
  .
  .
  .

 Z/OS DFSORT V2R1  ICEMAC DEFAULTS                  - 11 -

 ITEM           TD1 (ICETD1) VALUE       TD2 (ICETD2) VALUE       TD3 (ICETD3) VALUE       TD4
(ICETD4) VALUE
 ----------     --------------------     --------------------     --------------------
--------------------
 RELEASE        2.01                     2.01                     2.01                     2.01
```

```
 MODULE         ICETD1                  ICETD2                  ICETD3
ICETD4
 APAR LEVEL     BASE                    BASE                    BASE                    BASE
 COMPILED       05/15/11                05/08/11                05/08/11
05/08/11

 SUN            0600-2000               NONE                    NONE                    NONE
 MON            NONE                    NONE                    NONE                    NONE
 TUE            NONE                    NONE                    NONE                    NONE
 WED            NONE                    NONE                    NONE                    NONE
 THU            NONE                    NONE                    NONE                    NONE
 FRI            NONE                    NONE                    NONE                    NONE
 SAT            0600-2000               NONE                    NONE                    NONE

 ABCODE         99                      MSG                     MSG                     MSG
 ALTSEQ         SEE BELOW               SEE BELOW               SEE BELOW               SEE
BELOW
 ARESALL        0                       0                       0                       0
 ARESINV        0                       0                       0                       0
 CFW            YES                     YES                     YES                     YES
 CHALT          NO                      NO                      NO                      NO
 CHECK          YES                     YES                     YES                     YES
 CINV           YES                     YES                     YES                     YES
   .
   .
   .
```

The title and appropriate heading lines appear at the top of each page. The specified and IBM-supplied ALTSEQ tables are printed separately after the other items.

DISPLAY operator



Prints the values or characters of specified numeric fields (including SMF, TOD, and ETOD date and time) or character fields in a separate list data set. Simple, tailored, and sectioned reports can be produced.

From 1 to 50 fields can be specified, but the resulting list data set line length must not exceed the limit specified by the WIDTH operand or 8192 bytes if LONGLINE is specified and WIDTH is not specified, or 2048 bytes if LONGLINE and WIDTH are not specified. The record number can be printed as a special field.

DFSORT is called to copy the indd data set to ICETOOL's E35 user exit. ICETOOL uses its E35 user exit to print appropriate titles, headings and data in the list data set.

You must not supply your own DFSORT MODS, INREC, or OUTREC statement, because they would override the DFSORT statements passed by ICETOOL for this operator.

Specifying formatting items or the PLUS or BLANK operand, which can "**compress**" **the columns of output data, can enable you to include more fields in your report, up to a maximum of 50**, if your line length is limited by the character width your printer or display supports.

Simple report

You can produce a simple report by specifying just the required operands. For example, if you specify FROM and LIST operands, and ON operands for 10-byte character and 7-byte zoned decimal fields, the output in the list data set can be represented as follows:

```
(p,m,f)                (p,m,f)
characters             sddddddddddddddd
 .                                  .
 .                                  .
 .                                  .
```

A control character occupies the first byte of each list data set record. Left-justified standard headings are printed at the top of each page to indicate the contents of each column, followed by a line for each record showing the characters and numbers in the fields of that record.

The fields are printed in columns in the same order in which they are specified in the DISPLAY statement. All fields are left-justified. For numeric fields, leading zeros are printed, a - is used for the minus sign, and a + is used for the plus sign.

By default, the first column of data starts immediately after the control character, and three blanks appear between columns. The NOCC operand can be used to suppress the control character. The INDENT operand can be used to change the number of blanks before the first column of data. The BETWEEN operand can be used to change the number of blanks between columns.

The standard column widths are as follows:

- Character data: the length of the character field, or 20 bytes if the field length is less than 21 bytes
- Numeric data: 16 bytes, or 32 bytes if the numeric field is BI or FI with a length greater than 4, PD with a length greater than 8, ZD, CSF, FS, UFF or SFF with a length greater than 15, or FL.
- Record number: 15 bytes

HEADER operands can be used to change or suppress the headings. Formatting items or the PLUS or BLANK operand can be used to change the appearance of numeric fields in the report. PLUS, BLANK, and HEADER operands can be used to change the width of the columns for numeric and character fields and the justification of headings and fields.

The NOHEADER operand can be used to create list data sets containing only data records. Data sets created in this way can be processed further by other operators (for example, STATS or UNIQUE) using CH format for character values or FS format for numeric values.

The TOTAL, MAXIMUM, MINIMUM, and AVERAGE operands can be used to print statistics for numeric fields after the columns of data. Formatting items can be used to suppress the statistics for selected numeric fields. The COUNT operand can be used to print the record count after the columns of data.

Tailored report

You can tailor the output in the list data set using various operands that control title strings, date, time, page number, headings, lines per page, field formats, total, maximum, minimum, and average values for the columns of numeric data, and the record count. The optional operands can be used in many different combinations to produce a wide variety of report formats. For example, if you specify FROM, LIST, BLANK, TITLE, PAGE, DATE, TIME, HEADER and AVERAGE operands, and ON operands for 10-byte character and 7-byte zoned decimal fields, the output in the list data set can be represented as follows:

```
title        - p -        mm/dd/yy        hh:mm:ss

header         header
----------   --------
characters         sd
.                   .
.                   .
.                   .

average            sd
```

By default, a control character occupies the first byte of each list data set record. The NOCC operand can be used to suppress the control characters. The title lines (up to 3) are printed at the top of each page of the list data set. The first title line contains the elements you specify (title strings, page number, date and time) in the order in which you specify them. The second and third title lines contain the title strings you specify. By default, eight blanks appear between title elements, the title strings are centered with respect to each other, and the title lines appear on every page. The TBETWEEN(n) operand can be used to change the number of blanks between title elements. The TLEFT operand can be used to left-justify the title strings with respect to each other. The TFIRST operand can be used to only print the title lines on the first page. A blank line is printed after each title line.

Your specified headings (underlined) are printed after the title line on each page to indicate the contents of each column, followed by a line for each record showing the characters and numbers in the fields of that record. Your specified headings can be one, two or three lines. Headings for character fields are left-justified and headings for numeric fields are right-justified.

Your specified statistical lines (total, maximum, minimum, average, and count, and their associated strings) are printed for selected numeric fields and the record count, after the columns of data. The associated strings can be printed in the first column or to the left of it.

The fields are printed in columns in the same order in which they are specified in the DISPLAY statement. Character fields are left-justified and numeric fields are right-justified. For numeric fields, leading zeros are suppressed, a - is used for the minus sign, and a blank is used for the plus sign (you can specify PLUS rather than BLANK if you want a + to be used for the plus sign).

Formatting items can be used to change the appearance of individual numeric fields in the report with respect to separators, number of digits, decimal point, decimal places, signs, leading zeros, division by 10, 100, 1000, 10000, 100000, 1000000, 1000000000, 1024, 1048576 (1024*1024), or 1073741824 (1024*1024*1024), leading strings, floating strings, and trailing strings. Formatting items can also be used to insert leading or trailing strings for character fields.

The column widths are dynamically adjusted according to the length of the headings and the maximum number of bytes needed for the character or numeric data.

Sectioned report

You can produce a sectioned report (simple or tailored) by including a BREAK operand to indicate the break field used to divide the report into sections. Each set of sequential input records (which you have previously sorted on the break field and other fields, as appropriate), with the same value for the specified break field, results in a corresponding set of data lines that is treated as a section in the report.

The break field is printed at the beginning of each section. Formatting items can be used to change the appearance of numeric break fields, and to insert a string before or after character or numeric break fields.

Optional break operands can be used to modify the break title for each section (the break value is always printed as part of the break title) and to print statistics for selected numeric fields and the count, in each section. For example, if you add BTITLE, BREAK, BCOUNT, BMAXIMUM, and BMINIMUM to the operands for the tailored report discussed previously, each section of the output in the list data set starts on a new page and can be represented as follows:

```
title        - p -        mm/dd/yy        hh:mm:ss

btitle  bvalue

header         header
----------   --------
characters         sd
.                   .
.                   .
.                   .

bcount      d

bmaximum           sd

bminimum           sd
```

The final page showing the overall statistics starts on a new page and can be represented as follows:

```
title        - p -        mm/dd/yy        hh:mm:ss

header         header
----------   --------

average            sd
```

Operand descriptions

The operands described in this section can be specified in any order.

FROM(indd)
Specifies the ddname of the input data set to be read by DFSORT for this operation. An indd DD statement must be present and must define an input data set that conforms to the rules for DFSORT's SORTIN data set. In addition, the LRECL of the data set must be at least 4.

ON(p,m,f)
Specifies the position, length, and format of a numeric or character field to be used for this operation. '(p,m,f)' is used for the standard column heading (see HEADER('string1'), HEADER('string1','string2'), HEADER('string1','string2','string3'), HEADER(NONE) and NOHEADER for alternative heading options).

By default, three blanks appear between columns. You can change the space between columns with BETWEEN(n).

p specifies the first byte of the field relative to the beginning of the input record. p is 1 for the first **data** byte of a fixed-length record and 5 for the first **data** byte of a variable-length record as illustrated in the following (RRRR represents the 4-byte record descriptor word):

```
    Fixed-length record    |    Variable-length record
    | D | A | T | A | ...  |    | R | R | R | R | D | A | T | A | ...
 p=   1   2   3   4        | p=   1   2   3   4   5   6   7   8
```

m specifies the length of the field in bytes. A field must not extend beyond position 32752 or beyond the end of a record. The maximum length for a field depends on its format.

f specifies the format of the field as shown in the following.

Field formats of ICETOOL DISPLAY operator Format Code	Length	Description
BI	1 to 8 bytes	Unsigned binary

Field formats of ICETOOL DISPLAY operator Format Code	Length	Description
FI	1 to 8 bytes	Signed fixed-point
PD	1 to 16 bytes	Signed packed decimal
ZD	1 to 31 bytes	Signed zoned decimal
FL	4 or 8 bytes	Signed hexadecimal floating-point converted to signed integer
CH	1 to 4000 bytes	Character
CSF or FS	1 to 32 bytes (31 digit limit)	Signed numeric with optional leading floating sign
UFF	1 to 44 bytes (31 digit limit)	Unsigned free form numeric
SFF	1 to 44 bytes (31 digit limit)	Signed free form numeric
DT1	4 bytes	SMF date interpreted as Z'yyyymmdd'
DT2	4 bytes	SMF date interpreted as Z'yyyymm'
DT3	4 bytes	SMF date interpreted as Z'yyyyddd'
DC1	8 bytes	TOD date interpreted as Z'yyyymmdd'
DC2	8 bytes	TOD date interpreted as Z'yyyymm'
DC3	8 bytes	TOD date interpreted as Z'yyyyddd'
DE1	8 bytes	ETOD date interpreted as Z'yyyymmdd'
DE2	8 bytes	ETOD date interpreted as Z'yyyymm'
DE3	8 bytes	ETOD date interpreted as Z'yyyyddd'
TM1	4 bytes	SMF time interpreted as Z'hhmmss'
TM2	4 bytes	SMF time interpreted as Z'hhmm'
TM3	4 bytes	SMF time interpreted as Z'hh'
TM4	4 bytes	SMF time interpreted as Z'hhmmssxx'
TC1	8 bytes	TOD time interpreted as Z'hhmmss'
TC2	8 bytes	TOD time interpreted as Z'hhmm'
TC3	8 bytes	TOD time interpreted as Z'hh'
TC4	8 bytes	TOD time interpreted as Z'hhmmssxx'
TE1	8 bytes	ETOD time interpreted as Z'hhmmss'
TE2	8 bytes	ETOD time interpreted as Z'hhmm'
TE3	8 bytes	ETOD time interpreted as Z'hh'
TE4	8 bytes	ETOD time interpreted as Z'hhmmssxx'
Note: See Appendix C, “Data format descriptions,” on page 823 for detailed format descriptions.		

For a CSF, FS, UFF, or SFF format field:

- A maximum of 31 digits is allowed. If a value with more than 31 digits is found, ICETOOL issues an error message and terminates the operation.

For a ZD or PD format field:

- If a decimal value contains an invalid digit (A-F), ICETOOL identifies the bad value in a message and prints asterisks for that value, and for the total, maximum, minimum and average (if specified) for that field, in the list data set. If the number of bad values reaches the LIMIT for invalid decimal values, ICETOOL terminates the operation. If the LIMIT operand is not specified, a default of 200 is used for the invalid decimal value limit.
- A value is treated as positive if its sign is F, E, C, A, 8, 6, 4, 2, or 0.
- A value is treated as negative if its sign is D, B, 9, 7, 5, 3, or 1.

For an FL format field:

- The normalized or unnormalized FL (hexadecimal floating-point) value is converted to a signed integer in the range -9223372036854775808 to 9223372036854775807. The fractional part of the FL value is lost, and in some cases the signed integer may be one of a number of possible signed integers for the FL value depending on its precision. Converted values less than -9223372036854775808 are set to -9223372036854775808. Converted values greater than 9223372036854775807 are set to 9223372036854775807.
- If you are not running in z/Architecture mode, specifying an FL format field results in an error message and termination.

For a DT1, DT2 or DT3 format field:

- An invalid SMF date can result in a data exception (0C7 ABEND) or an incorrect ZD date.
- SMF date values are always treated as positive.

For a DC1, DC2, DC3, DE1, DE2, or DE3 format field:

- TOD and ETOD date values are always treated as positive.

For a TM1, TM2, TM3 or TM4 format field:

- An invalid SMF time can result in an incorrect ZD time.
- SMF time values are always treated as positive.

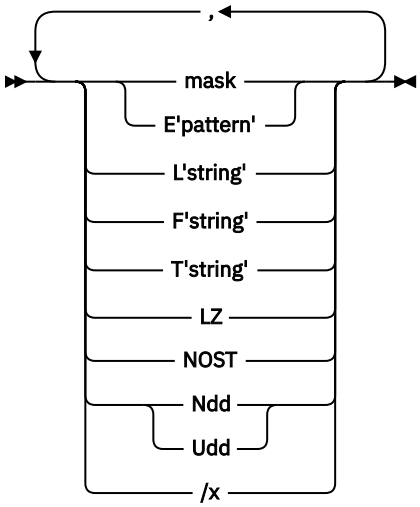
For a TC1, TC2, TC3, TC4, TE1, TE2, TE3, or TE4 format field:

- TOD and ETOD time values are always treated as positive.

ON(p,m,f,formatting)
Specifies the position, length, and format of a numeric or character field to be used for this operation and how the data for this field is to be formatted for printing. The BLANK operand is automatically in effect.

See ON(p,m,f) for further details.

formatting



specifies formatting items that indicate how the data for this field is to be formatted for printing. Formatting items can be specified in any order, but each item can only be specified once. Any formatting item can be specified for a numeric field, but only L'string' and T'string' can be specified for a character field.

The column width is dynamically adjusted to accommodate the maximum bytes to be inserted as a result of all formatting items specified.

mask

specifies an edit mask to be applied to the numeric data for this field. Thirty-nine pre-defined edit masks are available, encompassing many of the numeric notations throughout the world with respect to separators, decimal point, decimal places, signs, and so forth. ICETOOL edits the data according to the selected mask. If other formatting items are specified but mask is not, the default mask of A0 is applied to the data.

E'pattern' cannot be specified with a mask.

The attributes of each group of masks is shown in Table 90 on page 572.

Table 90. Attributes of Edit Masks

Attributes of Edit Masks Masks	Separators	Decimal Places	Positive Sign	Negative Sign
A0	No	0	blank	-
A1-A5	Yes	0	blank	-
B1-B6	Yes	1	blank	-
C1-C6	Yes	2	blank	-
D1-D6	Yes	3	blank	-
E1-E4	Yes	0	blank	( )
F1-F5	Yes	2	blank	( )
G1-G6	Yes	4	blank	-

Table 91 on page 573 describes the available masks and shows how the values 12345678 and -1234567 would be printed for each mask. In the pattern:

- **d** is used to represent a decimal digit (0-9)

- **w** is used to represent a leading sign that will be blank for a positive value or - for a negative value
- **x** is used to represent a trailing sign that will be blank for a positive value or - for a negative value
- **y** is used to represent a leading sign that will be blank for a positive value or (for a negative value
- **z** is used to represent a trailing sign that will be blank for a positive value or) for a negative value

Table 91. Edit Mask Patterns

Edit Mask PatternsMask	Pattern	12345678	-1234567
A0	wddddddddddddddddddddddddddddddd	12345678	-1234567
A1	wd,ddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd	12,345,678	-1,234,567
A2	wd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd	12.345.678	-1.234.567
A3	wd ddd ddd ddd ddd ddd ddd ddd ddd ddd ddd	12 345 678	-1 234 567
A4	wd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd	12'345'678	-1'234'567
A5	d ddd ddd ddd ddd ddd ddd ddd ddd ddd dddx	12 345 678	1 234 567-
B1	wddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd.d	1,234,567.8	-123,456.7
B2	wddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd,d	1.234.567,8	-123.456,7
B3	wddd ddd ddd ddd ddd ddd ddd ddd ddd ddd,d	1 234 567,8	-123 456,7
B4	wddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd.d	1'234'567.8	-123'456.7
B5	wddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd,d	1'234'567,8	-123'456,7
B6	ddd ddd ddd ddd ddd ddd ddd ddd ddd ddd,dx	1 234 567,8	123 456,7-
C1	wdd,ddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd.dd	123,456.78	-12,345.67
C2	wdd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd,dd	123.456,78	-12.345,67
C3	wdd ddd ddd ddd ddd ddd ddd ddd ddd ddd,dd	123 456,78	-12 345,67
C4	wdd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd.dd	123'456.78	-12'345.67
C5	wdd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd,dd	123'456,78	-12'345,67
C6	dd ddd ddd ddd ddd d ddd ddd ddd ddd,ddx	123 456,78	12 345,67-
D1	wd,ddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd.ddd	12,345.678	-1,234.567
D2	wd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd,ddd	12.345,678	-1.234,567
D3	wd ddd ddd ddd ddd ddd ddd ddd ddd ddd,ddd	12 345,678	-1 234,567
D4	wd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd.ddd	12'345.678	-1'234.567
D5	wd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd,ddd	12'345,678	-1'234,567
D6	d ddd ddd ddd ddd ddd ddd ddd ddd ddd,dddx	12 345,678	1 234,567-
E1	yd,ddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd,dddz	12,345,678	(1,234,567)
E2	yd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.dddz	12.345.678	(1.234.567)
E3	yd ddd ddd ddd ddd ddd ddd ddd ddd ddd dddz	12 345 678	(1 234 567)
E4	yd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'dddz	12'345'678	(1'234'567)
F1	ydd,ddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd.ddz	123,456.78	(12,345.67)
F2	ydd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd,ddz	123.456,78	(12.345,67)
F3	ydd ddd ddd ddd ddd ddd ddd ddd ddd ddd,ddz	123 456,78	(12 345,67)
F4	ydd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd.ddz	123'456.78	(12'345.67)

Table 91. Edit Mask Patterns (continued)

Edit Mask PatternsMask	Pattern	12345678	-1234567
F5	ydd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd,ddz	123'456,78	(12'345,67)
G1	wddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd,ddd.dddd	1,234.5678	-123.4567
G2	wddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd,dddd	1.234,5678	-123,4567
G3	wddd ddd ddd ddd ddd ddd ddd ddd ddd,dddd	1 234,5678	-123,4567
G4	wddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd.dddd	1'234.5678	-123.4567
G5	wddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd,dddd	1'234,5678	-123,4567
G6	ddd ddd ddd ddd ddd ddd ddd ddd ddd,ddddx	1 234,5678	123,4567-

If LZ is specified, leading zeros are printed. for example, +1 is shown as 0,000.01 with ON(21,6,FS,C1,LZ).

If LZ is not specified, leading zeros are suppressed except when inappropriate. For example, +1 is shown as 1 with ON(21,6,FS,A1) and as 0.01 with ON(21,6,FS,C1).

The leading sign (blank for a positive value or - for a negative value) appears to the left of the first non-suppressed digit of the formatted value. For example, -1 is shown as -1 with ON(21,6,FS,A2), as -000.001 with ON(21,6,FS,A2,LZ) and as -0.01 with ON(21,6,FS,C2).

E'pattern'
specifies an edit pattern to be applied to the numeric data for this field. E'pattern' is useful for formatting **unsigned** numeric data such as telephone numbers, dates, time-of-day, social security numbers, and so on. For example, 0123456789 is shown as (012)-345-6789 with ON(21,10,ZD,E'(999)-999-9999').

The pattern (1 to 44 characters) must be enclosed in single apostrophes. Each **9** in the pattern (up to 31) is replaced by a corresponding digit from the numeric value. Characters other than 9 in the pattern appear as specified. To include a single apostrophe (') in the pattern, specify two single apostrophes ('').

F'string' or a mask cannot be specified with E'pattern'.

When E'pattern' is specified for a field:

- Values are shown unsigned. For example, +120622 and -120622 are both shown as 12:06:22 with ON(12,7,FS,E'99:99:99').
- If the number of significant digits in a value is less than the number of 9's in the pattern, 0's are filled in on the left. For example, 1234 is shown as 0012-34 with ON(12,6,FS,E'9999-99').
- If the number of significant digits in a value is greater than the number of 9's in the pattern, digits are truncated from the left. For example, 1234567 is shown as *45:67* with ON(9,4,PD,E'*99:99*').

L'string'
specifies a leading string to appear at the beginning of the character or numeric data column for this field. For example, 'DFSORT ' is shown as '**DFSORT ' with ON(1,8,CH,L'**').

The string (1 to 10 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes ('').

F'string'
specifies a floating string to appear to the left of the first non-blank character of the formatted numeric data for this field. For example, 0001234 is shown as $12.34 with ON(9,7,ZD,C1,F'$').

The string (1 to 10 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes ('').

E'pattern' cannot be specified with F'string'.

T'string'
specifies a trailing string to appear at the end of the character or numeric data column for this field. For example, 'DFSORT ' is shown as '**DFSORT ***' with ON(1,8,CH,L'**',T'***').

The string (1 to 10 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes ('').

LZ
specifies that leading zeros are to be printed when the specified edit mask is applied to the numeric data for this field, overriding the default of suppressing leading zeros. For example, +123 is shown as 123 with ON(21,6,FS,A0), but as 000123 with ON(21,6,FS,A0,LZ).

LZ is useful for formatting numeric data, such as account numbers, for which leading zeros must be printed.

Leading zeros are always printed for E'pattern' regardless of whether or not LZ is specified.

NOST
specifies that requested statistics (TOTAL, MAXIMUM, MINIMUM, AVERAGE, BTOTAL, BMAXIMUM, BMINIMUM, BAVERAGE) are **not** to be printed for this numeric field.

Ndd or Udd
specifies the number of digits to be used for the numeric field. Ndd or Udd can be used to change the column width for numeric fields, and to prevent overflow for totals. dd specifies the number of digits and must be a **two-digit** number between 01 and 31.

The default number of digits (d) for a numeric field is the maximum number of digits for that field. For example, d is 5 for ON(1,5,ZD). If you know that your numeric field requires less than d digits, you can use a lower number of digits (dd) instead by specifying Udd, thus reducing the column width if it is determined by d. For example, ON(1,5,ZD,U03) reduces d from 5 to 3. If you want your numeric field to be displayed with more than d digits, you can use a higher number of digits (dd) instead by specifying Ndd or Udd, thus increasing the column width if it is determined by d. For example, ON(1,5,ZD,U10) increases d from 5 to 10.

The default number of digits (d) for a total is 15 if the numeric field is BI or FI with a length up to 4, PD with a length up to 8, or ZD, CSF, FS, UFF or SFF with a length up to 15. The default number of digits (d) for a total is 31 if the numeric field is BI or FI with a length greater than 4, PD with a length greater than 8, or ZD, CSF, FS, UFF or SFF with a length greater than 15.

If you know that your total requires less than d digits, you can use a lower number of digits (dd) instead by specifying Ndd or Udd for the ON field, thus reducing the column width if it is determined by d. For example, ON(1,18,ZD,U18) with TOTAL reduces d from 31 to 18. If you know that your total can overflow d digits, you can use a higher number of digits (dd) instead by specifying Ndd or Udd, thus preventing overflow and increasing the column width if it is determined by d. For example, ON(1,15,ZD,U17) with TOTAL increases d from 15 to 17.

Either Ndd or Udd can be used to set d greater than the maximum for a numeric field, but only Udd can be used to set d less than the maximum for a numeric field.

For Udd:

dd is used for d. For example:

If TOTAL and BTOTAL are not used:

- If ON(1,5,ZD) is specified, 5 digits (default for 5,ZD) are used.
- If ON(1,5,ZD,U10) is specified, 10 digits (from U10) are used..
- If ON(1,5,ZD,U03) is specified, 3 digits (from U03) are used.
- If ON(1,16,FS) is specified, 16 digits (default for 16,FS) are used.
- If ON(1,16,FS,U16) is specified, 16 digits (from U16) are used..
- If ON(1,16,FS,U15) is specified, 15 digits (from U15) are used.

If TOTAL or BTOTAL is used:

- If ON(1,5,ZD) is specified, 15 digits (default for TOTAL or BTOTAL and 5,ZD) are used.
- If ON(1,5,ZD,U10) is specified, 10 digits (from U10) are used.
- If ON(1,5,ZD,U03) is specified, 3 digits (from U03) are used.
- If ON(1,16,FS) is specified, 31 digits (default for TOTAL or BTOTAL and 16,FS) are used.
- If ON(1,16,FS,U16) is specified, 16 digits (from U16) are used.
- If ON(1,16,FS,U15) is specified, 15 digits (from U15) are used.

If you use Udd and a numeric value or total overflows dd digits, ICETOOL prints asterisks for that numeric value or total and terminates the operation. You can prevent the overflow by specifying an appropriately higher dd value for Udd. For example, if ON(1,12,ZD,U09) results in overflow, you can use ON(1,12,ZD,U10) instead.

If E'pattern' is specified, Udd is ignored, because d is determined from the pattern.

For Ndd:

If dd is greater than or equal d, dd is used. If dd is less than d, d is used. For example:

If TOTAL and BTOTAL are not used:

- If ON(1,5,ZD) is specified, 5 digits (default for 5,ZD) are used.
- If ON(1,5,ZD,N10) is specified, 10 digits (from N10) are used.
- If ON(1,5,ZD,N03) is specified, 5 digits (from 5,ZD) are used

If TOTAL or BTOTAL is used:

- If ON(1,5,ZD) is specified, 15 digits (default for TOTAL or BTOTAL and 5,ZD) are used.
- If ON(1,5,ZD,N10) is specified, 10 digits (from N10) are used.
- If ON(1,5,ZD,N03) is specified, 5 digits (from 5,ZD) are used.

If you use Ndd and a total overflows dd digits, ICETOOL prints asterisks for the total and terminates the operation. You can prevent the overflow by specifying an appropriately higher dd value for Ndd. For example, if ON(1,17,ZD,N17) with TOTAL results in overflow, you can use ON(1,17,ZD,N18) instead

If E'pattern' is specified, Ndd is ignored, because d is determined from the pattern.

/x

specifies division of the numeric data for this field before formatting. x indicates the division factor to be used as described later in this section. The resulting values are rounded down to the nearest integer. Statistics (TOTAL, MAXIMUM, MINIMUM, AVERAGE, BTOTAL, BMAXIMUM, BMINIMUM, BAVERAGE) and column widths reflect the divided numbers.

/D

specifies division by 10 before formatting. For example, -1234 is shown as -123 with ON(11,2,FI,/D).

/C

specifies division by 100 before formatting. For example, 12345 is shown as 12.3 with ON(11,2,BI,/C,B1).

/K

specifies division by 1000 before formatting. For example, -1234567890 is shown as (1 234 567) with ON(1,11,FS,/K,E3).

/DK

specifies division by 10000 (10*1000) before formatting. For example, 6213849653 is shown as 0-6213-84 with ON(31,10,FS,/DK,E'9-9999-99').

/CK
specifies division by 100000 (100*1000) before formatting. For example, 123456789012345678901234 5 is shown as 1,234,567,890,123,456.7890 with ON(21,25,ZD,G1,/CK).

/M
specifies division by 1000000 (1000*1000) before formatting. For example, -123456789 is shown as -1.23 with ON(31,10,FS,/M,C4).

/G
specifies division by 1000000000 (1000*1000*1000) before formatting. For example, 1234567898765 is shown as 1'234 with ON(15,13,ZD,A4,/G).

/KB
specifies division by 1024 before formatting. For example, 1234567890 is shown as 1 205 632 with ON(45,10,ZD,/KB,A3).

/MB
specifies division by 1048576 (1024*1024) before formatting. For example, 123456789 is shown as 117 with ON(60,9,FS,/MB).

/GB
specifies division by 1073741824 (1024*1024*1024) before formatting. For example, 1234567898765 is shown as 1,149 with ON(15,13,ZD,/GB,A1).

ON(p,m,HEX)
Specifies the position and length of a character field to be used for this operation and printed in hexadecimal format (00-FF for each byte). '(p,m,HEX)' is used for the standard column heading. See HEADER('string1'), HEADER('string1','string2'), HEADER('string1','string2','string3'), HEADER(NONE), and NOHEADER for alternative heading options.

See ON(p,m,f) for a discussion of **p**.

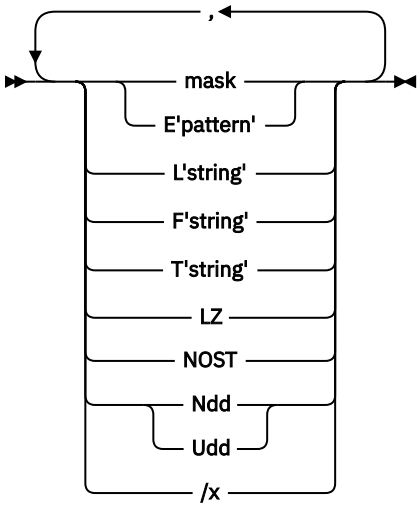
m specifies the length of the field in bytes. A field must not extend beyond position 32752 or beyond the end of a record. A field can be 1 to 2000 bytes.

ON(VLEN)
Equivalent to specifying ON(1,2,BI); a two-byte binary field starting at position 1. For variable-length records, ON(VLEN) represents the record-length for each record. ' RECORD LENGTH' is used for the standard column heading. See HEADER('string1'), HEADER('string1','string2'), HEADER('string1','string2','string3'), HEADER(NONE), and NOHEADER for alternative heading options.

ON(VLEN,formatting)
Equivalent to specifying ON(1,2,BI,formatting); a two-byte binary field starting at position 1, and how the data for this field is to be formatted for printing. The BLANK operand is automatically in effect.

See ON(VLEN) for further details.

formatting



specifies formatting items that indicate how the data for this field is to be formatted for printing. Formatting items can be specified in any order, but each item can only be specified once.

The column width is dynamically adjusted to accommodate the maximum bytes to be inserted as a result of all formatting items specified.

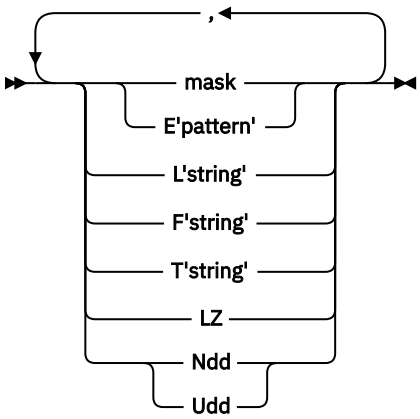
See ON(p,m,f,formatting) for a discussion of **formatting**.

ON(NUM)
Specifies that the record number is to be printed. The record number starts at 1 and is incremented by 1 for each record printed in the list data set. 'RECORD NUMBER' is used for the standard column heading. See HEADER('string1'), HEADER('string1','string2'), HEADER('string1','string2','string3'), HEADER(NONE), and NOHEADER for alternative heading options.

ON(NUM,formatting)
Specifies that the record number is to be printed, and how the record number is to be formatted for printing. The BLANK operand is automatically in effect.

See ON(NUM) for further details.

formatting



specifies formatting items that indicate how the record number is to be formatted for printing. Formatting items can be specified in any order, but each item can only be specified once.

The column width is dynamically adjusted to accommodate the maximum bytes to be inserted as a result of all formatting items specified.

mask
See ON(p,m,f,formatting) for a discussion of **mask**.

E'pattern'
specifies an edit pattern to be applied to the record number. The pattern (1 to 24 characters) must be enclosed in single apostrophes. Each 9 in the pattern (up to 15) is replaced by a corresponding digit from the numeric value. Characters other than 9 in the pattern appear as specified. To include a single apostrophe (') in the pattern, specify two single apostrophes ('').

F'string' or a mask cannot be specified with E'pattern'.

When E'pattern' is specified for the record number:

- If the number of significant digits in a record number is less than the number of 9's in the pattern, 0's are filled in on the left. For example, 1234 is shown as 001234 with ON(NUM,E'999999').
- If the number of significant digits in a record number is greater than the number of 9's in the pattern, digits are truncated from the left. For example, 1234567 is shown as *4567* with ON(NUM,E'*9999*').

L'string'
See ON(p,m,f,formatting) for a discussion of **L'string'**.

F'string'
See ON(p,m,f,formatting) for a discussion of **F'string'**.

T'string'
See ON(p,m,f,formatting) for a discussion of **T'string'**.

LZ
See ON(p,m,f,formatting) for a discussion of **LZ**.

Ndd or Udd
Specifies the number of digits to be used for the record number when determining the column width. dd specifies the number of digits and must be a **two-digit** number between 01 and 15.

The default number of digits (d) for the record number is 15. If you know that your record numbers require less than 15 digits, you can use a lower number of digits (dd) instead by specifying Ndd or Udd thus reducing the column width if it is determined by d. For example, if ON(NUM,N09) or ON(NUM,U09) is specified, 9 digits (from N09 or U09) is used instead of 15 (default for record number).

If you use Ndd or Udd and the number of records overflows the number of digits used, ICETOOL terminates the operation. You can prevent the overflow by specifying an ppropriately higher dd value for Ndd or Udd. For example, if ON(NUM,N05) results in overflow, you can use ON(NUM,N06) instead.

If E'pattern' is specified, Ndd or Udd is ignored, because d is determined from the pattern.

LIST(listdd)
Specifies the ddname of the list data set to be produced by ICETOOL for this operation. A listdd DD statement must be present. ICETOOL sets the attributes of the list data set as follows:

- If NOCC is not specified, RECFM is set to FBA. If NOCC is specified, RECFM is set to FB.
- LRECL is set to one of the following:
  - If WIDTH(n) is specified, LRECL is set to n. Use WIDTH(n) if your LRECL must be set to a particular value (for example, if you use DISP=MOD to place several reports in the same data set).
  - If WIDTH(n) is not specified and NOCC is not specified, LRECL is set to 121 or to the calculated required line length if it is greater than 121 characters.
  - If WIDTH(n) is not specified and NOCC is specified, LRECL is set to 120 or to the calculated required line length if it is greater than 120 characters.

  If your LRECL does not need to be set to a particular value, you can let ICETOOL determine and set the appropriate LRECL value by not specifying WIDTH(n).
- BLKSIZE is set to one of the following:

- The BLKSIZE from the DD statement, DSCB, or label, if it is a multiple of the LRECL used.
- The LRECL if the BLKSIZE from the DD statement, DSCB, or label is not a multiple of the LRECL used.
- The block size as directed by LISTSDB or LISTNOSDB if specified, or otherwise as directed by the SDBMSG installation option from ICEAM2 or ICEAM4 (see *z/OS DFSORT Installation and Customization*), if the BLKSIZE is not available from the DD statement, DSCB, or label.

Refer to "JCL restrictions" on page 546 for more information regarding the selection of ddnames.

TITLE('string')
Specifies printing of a title string in a title line. The title string is of the form:

```
string
```

Up to three TITLE operands can be specified.

The first TITLE operand, if specified, supplies a string for the first title line. Other title elements (PAGE for page number, DATE, DATE(abcd), DATENS(abc), YDDD(abc) or YDDDNS(ab) for the date, and TIME, TIME(abc) or TIMENS(ab) for the time) can also be specified for the first title line with or without a title string.

The second TITLE operand, if specified, specifies a string for the second title line. The third TITLE operand, if specified, specifies a string for the third title line.

If you specify any title elements (title string, page number, date or time), the first title line is printed at the top of each page of the list data set. It contains the title elements you specify in the order in which you specify them. By default, eight blanks appear between title elements on the first title line. You can change the space between title elements on the first title line with TBETWEEN(n). A blank line is printed after the first title line.

If you specify a second TITLE operand, the second title line containing the specified title string is printed after the first title line. A blank line is printed after the second title line. If you specify a third TITLE operand, the third title line containing the specified title string is printed after the second title line. A blank line is printed after the third title line.

By default, the title strings in the title lines are centered with respect to each other. TLEFT can be used to left-justify the title string in the title lines with respect to each other.

By default, the title lines are printed on every page. TFIRST can be used to only print the title lines on the first page.

The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes (''). Blanks at the start of the string move the text to the right. Blanks at the end of the string increase the spacing between the string and the next title element.

TITLE('string1','string2')
Specifies printing of a two part title string in a title line. The title string is of the form:

```
string1string2
```

Note: string1 and string2 can be any combination of inline constants, DFSORT symbol constants, and DFSORT symbol constants for system information (for example, S'&Sysname').

See TITLE('string') for additional informaton on title lines.

Each string (1 to 50 characters) must be enclosed in single apostrophes. The total combined length of string1 and string2 can be up to 50 characters. To include a single apostrophe (') in a string, specify two single apostrophes (''). Blanks at the start of a string move the text to the right. Blanks at the end of string1 increase the spacing between string1 and string2. Blanks at the end of string2 increase the spacing between string2 and the next title element.

TITLE('string1','string2','string3')
Specifies printing of a three part title string in a title line. The title string is of the form:

```
string1string2string3
```

Note: string1, string2 and string3 can be any combination of inline constants, DFSORT symbol constants, and DFSORT symbol constants for system information (for example, S'&Sysname').

See TITLE('string') for additional informaton on title lines.

Each string (1 to 50 characters) must be enclosed in single apostrophes. The total combined length of string1, string2 and string3 can be up to 50 characters. To include a single apostrophe (') in a string, specify two single apostrophes (''). Blanks at the start of a string move the text to the right. Blanks at the end of string1 increase the spacing between string1 and string2. Blanks at the end of string2 increase the spacing between string2 and string3. Blanks at the end of string3 increase the spacing between string3 and the next title element.

TLEFT
Specifies that the title strings in each title line are to be left justified with respect to each other (overriding the default of centering the title strings with respect to each other). See TITLE('string') for additional informaton on title lines.

TFIRST
Specifies that the title lines are only to appear on the first page of the report (overriding the default of having the title lines appear on every page of the report). See TITLE('string') for additional informaton on title lines.

PAGE
Specifies printing of the page number in the first title line. The page number is printed in the form - p - where p is in decimal with no leading zeros. The page number is 1 for the first page and is incremented by 1 for each subsequent page. See TITLE('string') for additional information on the first title line.

DATE
Specifies printing of the date in the first title line. The date is printed in the form mm/dd/yy where mm is the month, dd is the day, and yy is the year. DATE is equivalent to specifying DATE(MDY/). See TITLE('string') for additional information on the first title line.

DATE(abcd)
Specifies printing of the date in the first title line. The date is printed in the form 'adbdc' according to the specified values for abc and d. For example, on March 29, 2002, DATE(4MD-) would produce '2002-03-29' and DATE(MDY.) would produce '03.29.02'. See TITLE('string') for additional information on the first title line.

abc can be any combination of M, D, and Y or 4 (each specified once) where M represents the month (01-12), D represents the day (01-31), Y represents the last two digits of the year (for example, 02), and 4 represents the four digits of the year (for example, 2002).

d can be any character and is used to separate the month, day, and year.

DATENS(abc)
Specifies printing of the date in the first title line. The date is printed in the form 'abc' according to the specified values for abc. For example, on March 29, 2002, DATENS(4MD) would produce '20020329' and DATENS(MDY) would produce '032902'. See TITLE('string') for additional information on the first title line.

abc can be any combination of M, D, and Y or 4 (each specified once) where M represents the month (01-12), D represents the day (01-31), Y represents the last two digits of the year (for example, 02), and 4 represents the four digits of the year (for example, 2002).

YDDD(abc)
specifies printing of the date in the first title line. The date is printed in the form 'acb' according to the specified values for ab and c. For example, on April 7, 2004, YDDD(DY-) would produce '098-04' and YDDD(4D/) would produce '2004/098'. See TITLE('string') for additional information on the first title line.

ab can be any combination of D, and Y or 4 (each specified once) where D represents the day of the year (001-366), Y represents the last two digits of the year (for example, 04), and 4 represents the four digits of the year (for example, 2004).

c can be any character and is used to separate the month, day and year.

YDDDNS(ab)
specifies printing of the date in the first title line. The date is printed in the form 'ab' according to the specified values for ab. For example, on April 7, 2004, YDDDNS(DY) would produce '09804' and YDDD(4D) would produce '2004098'. See TITLE('string') for additional information on the first title line.

ab can be any combination of D, and Y or 4 (each specified once) where D represents the day of the year (001-366), Y represents the last two digits of the year (for example, 04), and 4 represents the four digits of the year (for example, 2004).

TIME
Specifies printing of the time in the first title line. The time is printed in the form hh:mm:ss where hh is hours, mm is minutes and ss is seconds. TIME is equivalent to specifying TIME(24:). See TITLE('string') for additional information on the first title line.

TIME(abc)
Specifies printing of the time in the first title line. The time is printed in the form 'hhcmmcss xx' according to the specified value for ab and c. For example, at 08:25:13 pm, TIME=(24:) would produce '20:25:13' and TIME=(12.) would produce '08.25.13 pm'. See TITLE('string') for additional information on the first title line.

ab can be:

- 12 to indicate 12-hour time. hh (hours) is 01-12, mm (minutes) is 00-59, ss (seconds) is 00-59 and xx is am or pm.
- 24 to indicate 24-hour time. hh (hours) is 00-23, mm (minutes) is 00-59, ss (seconds) is 00-59 and xx is not included.

c can be any character and is used to separate the hours, minutes, and seconds.

TIMENS(ab)
Specifies printing of the time in the first title line. The time is printed in the form 'hhmmss xx' according to the specified value for ab. For example, at 08:25:13 pm, TIMENS=(24) would produce '202513' and TIMENS=(12) would produce '082513 pm'. See TITLE('string') for additional information on the first title line.

ab can be:

- 12 to indicate 12-hour time. hh (hours) is 01-12, mm (minutes) is 00-59, ss (seconds) is 00-59 and xx is am or pm.
- 24 to indicate 24-hour time. hh (hours) is 00-23, mm (minutes) is 00-59, ss (seconds) is 00-59 and xx is not included.

BLANK
Specifies an alternate format for printing character and numeric data as follows:

- Numeric values for which formatting is not specified are printed with blank for plus sign, - for minus sign and no leading zeros (overriding the default of + for plus sign and leading zeros).

  Numeric values are thus displayed as:

  – d...d for positive values (blank sign immediately to the left of the digits and no leading zeros)
  – -d...d for negative values (- sign immediately to the left of the digits and no leading zeros)
- Column widths are dynamically adjusted according to the length of the headings and the maximum number of bytes needed for the character or numeric data
- Headings and data for numeric fields are right-justified (overriding the default of left-justified headings and data for numeric fields)

PLUS
Specifies an alternate format for printing character and numeric data as follows:

- Numeric values for which formatting is not specified are printed with + for plus sign, - for minus sign and no leading zeros (overriding the default of leading zeros).

  Numeric values are thus displayed as:

  – +d...d for positive values (- sign immediately to the left of the digits and no leading zeros)
  – -d...d for negative values (- sign immediately to the left of the digits and no leading zeros)
- Column widths are dynamically adjusted according to the length of the headings and the maximum number of bytes needed for the character or numeric data
- Headings and data for numeric fields are right-justified (overriding the default of left-justified headings and data for numeric fields)

For ON(NUM), PLUS is treated as BLANK.

NOCC
Specifies that carriage control characters are not to be included in the lines of the list data set (overriding the default of using a carriage control character as the first byte of each line). A blank line is used instead of a page eject control character to separate elements of the report (such as sections).

The RECFM of the list data set is set to FB.

The LRECL of the list data set will not include a byte for the carriage control character. If the line length is less than or equal to 120 bytes, the LRECL will be set to 120. If the line length is greater than 120 bytes, the LRECL will be set to the line length. The maximum line length is 2047 bytes if LONGLINE is not specified, or 8191 bytes if LONGLINE is specified.

If WIDTH(n) is specified, n can be 121 to 8191.

HEADER('string1')

Specifies a heading to be printed for the corresponding ON field. The specified string is used instead of the standard column heading for the corresponding ON field. (ON fields and HEADER operands correspond one-for-one according to the order in which they are specified; that is, the first HEADER operand corresponds to the first ON field, the second HEADER operand corresponds to the second ON field, and so on.)

The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes (''). If the string length is greater than the column width for the corresponding ON field, the column width is increased to the string length.

The heading is left-justified for character fields or right-justified for numeric fields and is underlined with hyphens for the entire column width (overriding the default of left-justified, non-underlined headings). Character values are left-justified and numeric values are right-justified (overriding the default of left-justified field values).

A null string ('') or blank string (' ') may be used to set string1 to blanks.

Blanks at the start or end of a heading string may alter the justification of the heading or the width of the column.

If HEADER('string1') is used for any ON field, HEADER('string1'), HEADER('string1','string2'), HEADER('string1','string2','string3'), or HEADER(NONE) must be used for each ON field.

HEADER('string1','string2')
Specifies a heading to be printed for the corresponding ON field. A two-line heading is used with string1 on line1 and string2 on line2. A null string ('') or blank string (' ') may be used to set string1 or string2 to blanks. A comma (,) may also be used to set string1 to blanks. For example, HEADER(,'string1') results in blanks for this heading in line1 and string1 for this heading in line2.

If HEADER('string1','string2') is used for any ON field, HEADER('string1'), HEADER('string1','string2'), HEADER('string1','string2','string3') or HEADER(NONE) must be used for each ON field.

If a HEADER('string1','string2') operand is specified, a two-line heading is also used for any HEADER('string1') operands you specify, with string1 for that heading on line1 and blanks for that heading on line2. HEADER(,'string1') can be used to put blanks for that heading on line1 and string1 for that heading on line2.

See HEADER('string1') for more details on the HEADER operand.

HEADER('string1','string2','string3')
Specifies a heading to be printed for the corresponding ON field. A three-line heading is used with string1 on line1, string2 on line2 and string3 on line3. A null string ('') or blank string (' ') may be used to set string1, string2 or string3 to blanks. A comma (,) may also be used to set string1 or string2 to blanks. For example, HEADER(,,'string1') results in blanks for this heading in line1 and line2 and string1 for this heading in line3.

If HEADER('string1','string2','string3') is used for any ON field, HEADER('string1'), HEADER('string1','string2'), HEADER('string1','string2','string3') or HEADER(NONE) must be used for each ON field.

If a HEADER('string1','string2','string3') operand is specified:

- a three-line heading is also used for any HEADER('string1') operands you specify, with string1 for that heading on line1 and blanks for that heading on line2 and line3. HEADER(,,'string1') can be used to put blanks for that heading on line1 and line2 and string1 for that heading on line3.
- a three-line heading is also used for any HEADER('string1','string2') operands you specify, with string1 for that heading on line1, string2 for that heading on line2 and blanks for that heading on line3. HEADER('string1','string2') can be used to put blanks for that heading on line1, string1 for that heading on line2 and string2 for that heading on line3.

See HEADER('string1') for more details on the HEADER operand.

HEADER(NONE)
Specifies that a heading is not to be printed for the corresponding ON field. The standard column heading for the corresponding ON field is suppressed.

If HEADER(NONE) is used for any ON field, HEADER('string1'), HEADER('string1','string2'), HEADER('string1','string2','string3'), or HEADER(NONE) must be used for each ON field. Specifying HEADER(NONE) for every ON field is equivalent to specifying NOHEADER.

NOHEADER

Specifies that headings for ON fields are not to be printed (overriding the default of printing standard headings for ON fields).

If NOHEADER is used, it must be specified only once and HEADER('string1'), HEADER('string1','string2'), HEADER('string1','string2','string3'), and HEADER(NONE) must not be used.

If NOHEADER is specified without any TITLE, DATE, TIME, or PAGE operands, the resulting list data set contains only data records. Data sets created in this way can be processed further by other operators (for example, STATS or UNIQUE) using CH for character values or FS for numeric values.

LINES(n)
Specifies the number of lines per page for the list data set (overriding the default of 58). n must be greater than 9, but less than 1000.

INDENT(n)
Specifies the number of blanks to be used to indent the report (overriding the default of 0). n can be from 0 to 50. For example, if INDENT(n) is not specified, the report starts in column 2 (after the

control character), whereas if INDENT(10) is specified, the report starts in column 12 (after the control character and 10 blanks).

BETWEEN(n)
Specifies the number of blanks to be used between the columns of data (overriding the default of 3). n can be from 0 to 50. For example, if BETWEEN(n) is not specified, three blanks appear between columns, whereas if BETWEEN(7) is specified, seven blanks appear between columns.

TBETWEEN(n)
Specifies the number of blanks to be used between title elements (title strings, page number, date and time) in the first title line (overriding the default of 8). n can be from 0 to 50. For example, if TBETWEEN(n) is not specified, eight blanks appear between the title strings and page number in the first title line, whereas if TBETWEEN(4) is specified, four blanks appear between the title strings and page number in the first title line.

TOTAL('string')
Specifies an overall TOTAL line is to be printed after the rows of data for the report. The specified string is printed starting at the indent column of the overall TOTAL line, followed by the overall total for each numeric data column. If STATLEFT is specified, the string is printed to the left of the first column of data with the totals on the same line as the string. If STATLEFT is not specified, the string is printed in the first column of data with the totals on the same line as the string, or on the next line, as appropriate. A blank line is printed before the overall TOTAL line.

The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes (''). To suppress printing of a string, specify TOTAL('') using two single apostrophes.

The overall total for each numeric ON field is printed in the format (formatting, PLUS, BLANK, or standard) you specify. The total for a specific numeric field is suppressed if the NOST formatting item is specified for that field. Totals are printed for ON(VLEN) fields, but not for ON(NUM) fields.

The default number of digits (d) for a total is 15 if the ON field is BI or FI with a length up to 4, PD with a length up to 8, or ZD, CSF, FS, UFF or SFF with a length up to 15. The default number of digits (d) for a total is 31 if the ON field is BI or FI with a length greater than 4, PD with a length greater than 8, ZD, CSF, FS, UFF or SFF with a length greater than 15, or FL. By default, column widths are adjusted to allow for a maximum of a sign and d digits for the totals. If the overall total for an ON field overflows d digits, ICETOOL prints asterisks for the overall total for that field and terminates the operation.

You can use the Ndd or Udd formatting item to decrease or increase the number of digits used for a total. If you use Ndd or Udd and the overall total for an ON field overflows dd digits, ICETOOL prints asterisks for the overall total for that field and terminates the operation.

You can prevent overflow by specifying an appropriate dd value for Ndd or Udd. For example, if ON(1,15,ZD) with TOTAL overflows the default of 15 digits, you can specify ON(1,15,ZD,U16) to prevent overflow. If ON(1,15,ZD,U16) still results in overflow, you can specify ON(1,15,ZD,U17), and so on.

Either Ndd or Udd can be used to set the number of digits greater than the maximum for a numeric field, but only Udd can be used to set the number of digits less than the maximum for a numeric field.

See the discussion of Ndd or Udd under ON(p,m,f,formatting) for more details on using Ndd or Udd with TOTAL.

The TOTAL, MAXIMUM, MINIMUM, AVERAGE, and COUNT lines are printed in the order in which you specify them.

MAXIMUM('string')
Specifies an overall MAXIMUM line is to be printed after the rows of data for the report. The specified string is printed starting at the indent column of the overall MAXIMUM line, followed by the overall maximum for each numeric data column. If STATLEFT is specified, the string is printed to the left of the first column of data with the maximums on the same line as the string. If STATLEFT is not specified, the string is printed in the first column of data with the maximums on

the same line as the string, or on the next line, as appropriate. A blank line is printed before the overall MAXIMUM line.

The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes (''). To suppress printing of a string, specify MAXIMUM('') using two single apostrophes.

The overall maximum for each numeric ON field is printed in the format (formatting, PLUS, BLANK, or standard) you specify. The maximum for a specific numeric field is suppressed if the NOST formatting item is specified for that field. Maximums are printed for ON(VLEN) fields, but not for ON(NUM) fields.

The TOTAL, MAXIMUM, MINIMUM, AVERAGE, and COUNT lines are printed in the order in which you specify them.

MINIMUM('string')
Specifies an overall MINIMUM line is to be printed after the rows of data for the report. The specified string is printed starting at the indent column of the overall MINIMUM line, followed by the overall minimum for each numeric data column. If STATLEFT is specified, the string is printed to the left of the first column of data with the minimums on the same line as the string. If STATLEFT is not specified, the string is printed in the first column of data with the minimums on the same line as the string, or on the next line, as appropriate. A blank line is printed before the overall MINIMUM line.

The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes (''). To suppress printing of a string, specify MINIMUM('') using two single apostrophes.

The overall minimum for each numeric ON field is printed in the format (formatting, PLUS, BLANK, or standard) you specify. The minimum for a specific numeric field is suppressed if the NOST formatting item is specified for that field. Minimums are printed for ON(VLEN) fields, but not for ON(NUM) fields.

The TOTAL, MAXIMUM, MINIMUM, AVERAGE, and COUNT lines are printed in the order in which you specify them.

AVERAGE('string')
Specifies an overall AVERAGE line is to be printed after the rows of data for the report. The specified string is printed starting at the indent column of the overall AVERAGE line, followed by the overall average for each numeric data column. If STATLEFT is specified, the string is printed to the left of the first column of data with the averages on the same line as the string. If STATLEFT is not specified, the string is printed in the first column of data with the averages on the same line as the string, or on the next line, as appropriate. A blank line is printed before the overall AVERAGE line.

The overall average (or mean) is calculated by dividing the overall total by the number of values in the report and rounding down to the nearest integer (examples: 23 / 5 = 4, -23 / 5 = -4).

The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes ('). To suppress printing of a string, specify AVERAGE('') using two single apostrophes.

The overall average for each numeric ON field is printed in the format (formatting, PLUS, BLANK, or standard) you specify. The average for a specific numeric field is suppressed if the NOST formatting item is specified for that field. Averages are printed for ON(VLEN) fields, but not for ON(NUM) fields.

You can use the Ndd or Udd formatting item to decrease or increase the number of digits used for a total. If the overall total for an ON field overflows d digits, ICETOOL prints asterisks for the overall average for that field and terminates the operation. You can prevent overflow by specifying an appropriate dd value for Ndd or Udd. For example, if ON(1,15,ZD) with AVERAGE overflows the default of 15 digits for the total, you can specify ON(1,15,ZD,U16) to prevent overflow.

See the discussion of Ndd or Udd under ON(p,m,f,formatting) for more details on using Ndd or Udd.

The TOTAL, MAXIMUM, MINIMUM, AVERAGE, and COUNT lines are printed in the order in which you specify them.

COUNT('string')
Specifies an overall COUNT line is to be printed after the rows of data for the report. The specified string is printed starting at the indent column of the overall COUNT line, followed by the overall count of data records. If STATLEFT is specified, the string is printed to the left of the first column of data. If STATLEFT is not specified, the string is printed in the first column of data. The count is printed on the same line as the string. A blank line is printed before the overall COUNT line.

The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes (''). To suppress printing of a string, specify COUNT('') using two single apostrophes.

The count is printed in the format (PLUS, BLANK, or standard) you specify. EDCOUNT(formatting) can be used to apply formatting items to the count. The default number of digits (d) for the count is 15.

The TOTAL, MAXIMUM, MINIMUM, AVERAGE, and COUNT lines are printed in the order in which you specify them.

EDCOUNT(formatting)
Specifies how the overall count is to be formatted for printing. Formatting items can be specified in any order, but each item can only be specified once. EDCOUNT can only be specified if COUNT('string') is specified.

mask
See ON(p,m,f,formatting) for a discussion of **mask**.

E'pattern'
specifies an edit pattern to be applied to the count. The pattern (1 to 24 characters) must be enclosed in single apostrophes. Each 9 in the pattern (up to 15) is replaced by a corresponding digit from the count. Characters other than 9 in the pattern appear as specified. To include a single apostrophe (') in the pattern, specify two single apostrophes (''). F'string' or a mask cannot be specified with E'pattern'.

When E'pattern' is specified for the count:

- If the number of significant digits in the count is less than the number of 9's in the pattern, 0's are filled in on the left. For example, 1234 is shown as 001234 with EDCOUNT(E'999999').
- If the number of significant digits in the count is greater than the number of 9's in the pattern, digits are truncated from the left. For example, 1234567 is shown as *4567* with EDCOUNT(E'*9999*').

L'string'
See ON(p,m,f,formatting) for a discussion of **L'string'**.

F'string'
See ON(p,m,f,formatting) for a discussion of **F'string'**.

T'string'
See ON(p,m,f,formatting) for a discussion of **T'string'**.

LZ
See ON(p,m,f,formatting) for a discussion of **LZ**.

Udd
specifies the number of digits to be used for the count. dd specifies the number of digits and must be a two-digit number between 01 and 15. The default number of digits (d) for the count is 15.

If you know that your count requires less than 15 digits, you can use a lower number of digits (dd) instead by specifying Udd. For example, if EDCOUNT(U09) is specified, 9 digits (from U09) is used instead of 15 (default for the count).

If you use Udd and the count overflows the number of digits used, ICETOOL terminates the operation. You can prevent the overflow by specifying an appropriately higher dd value for Udd. For example, if EDCOUNT(U05) results in overflow, you can use EDCOUNT(U06) instead.

If E'pattern' is specified, Udd is ignored, because d is determined from the pattern.

LIMIT(n)
Specifies a limit for the number of invalid decimal values (overriding the default of 200). If n invalid decimal values are found, ICETOOL terminates the operation. n can be 1 to 15 decimal digits, but must be greater than 0.

VSAMTYPE(x)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

WIDTH(n)
Specifies the line length and LRECL you want ICETOOL to use for your list data set. If NOCC is not specified, n can be from 121 to 8192. If NOCC is specified, n can be from 121 to 8191.

ICETOOL always calculates the line length required to print all titles, headings, data, and statistics and uses it as follows:

- If WIDTH(n) is specified and the calculated line length is less than or equal to n, ICETOOL sets the line length and LRECL to n.
- If WIDTH(n) is specified and the calculated line length is greater than n, ICETOOL issues an error message and terminates the operation.
- If WIDTH(n) is not specified, NOCC is not specified, and the calculated line length is less than or equal to 121, ICETOOL sets the line length and LRECL to 121.
- If WIDTH(n) is not specified, NOCC is specified, and the calculated line length is less than or equal to 120, ICETOOL sets the line length and LRECL to 120.
- If WIDTH(n) is not specified, NOCC is specified, and the calculated line length is between 121 and 2047, or between 121 and 8191 if LONGLINE is specified, ICETOOL sets the line length and LRECL to the calculated line length.
- If WIDTH(n) is not specified, NOCC is not specified, and the calculated line length is between 122 and 2048, or between 122 and 8192 if LONGLINE is specified, ICETOOL sets the line length and LRECL to the calculated line length.
- If WIDTH(n) is not specified, NOCC is not specified, and the calculated line length is greater than 2048, or greater than 8192 if LONGLINE is specified, ICETOOL issues an error message and terminates the operation.
- If WIDTH(n) is not specified, NOCC is specified, and the calculated line length is greater than 2047, or greater than 8191 if LONGLINE is specified, ICETOOL issues an error message and terminates the operation.

Use WIDTH(n) if your LRECL must be set to a particular value (for example, if you use DISP=MOD to place several reports in the same data set) or if you want to ensure that the line length for your report does not exceed a specific maximum (for example, 133 bytes). Otherwise, you can let ICETOOL calculate and set the appropriate line length and LRECL by not specifying WIDTH(n).

LONGLINE
Specifies the calculated line length can be a maximum of 8191 bytes if NOCC is specified, or a maximum of 8192 bytes if NOCC is not specified (overriding the defaults of 2047 bytes if NOCC is specified or 2048 bytes if NOCC is specified). Use LONGLINE if you do not specify WIDTH(n) and your calculated line length is larger than the default.

BREAK(p,m,f)
Specifies a numeric or character break field to be used to divide the report into sections. Each set of sequential input records, with the same value for the specified break field, results in a corresponding set of data lines that is treated as a section in the report. The DISPLAY operator

should be preceded by a SORT operator (or another application) that sorts the break field and any other appropriate fields in the desired sequence for the report.

Each section starts on a new page. Each page of a section includes a break title line showing the break value for the section. Numeric break values are printed with blank for plus sign, - for minus sign, and no leading zeros. BTITLE can be used to specify a string to appear in the break title line. The break value and break title string appear in the order in which you specify BREAK and BTITLE. Two blanks appear between break title elements. A blank line is printed after the break title line.

BTOTAL, BMAXIMUM, BMINIMUM, BAVERAGE, and BCOUNT can be used to produce break statistics for each numeric ON field and for the break count-for example, the maximum of the values in the section for ON(5,3,ZD) and the maximum of the values in the section for ON(22,2,BI). The break statistics for each section are printed at the end of the section (on one or more pages that include the break title). TOTAL, MAXIMUM, MINIMUM, AVERAGE, and COUNT can be used to produce overall statistics for each numeric ON field and for the overall count-for example, the maximum of the values in the report for ON(5,3,ZD) and the maximum of the values in the report for ON(22,2,BI). The overall statistics for each section are printed at the end of the report (on a separate page that does not include the break title).

See ON(p,m,f) for a discussion of **p** and **m**.

f specifies the format of the field as shown for ON(p,m,f).

Note: An FL (hexadecimal floating-point) field can be specified for ON, but not for BREAK.

For a CSF, FS, UFF, or SFF format break field:

- A maximum of 31 digits is allowed. If a value with more than 31 digits is found, ICETOOL issues an error message and terminates the operation.

For a ZD or PD format break field:

- If a decimal value with an invalid digit (A-F) is found, ICETOOL issues an error message and terminates the operation.
- A value is treated as positive if its sign is F, E, C, A, 8, 6, 4, 2, or 0.
- A value is treated as negative if its sign is D, B, 9, 7, 5, 3, or 1.

For a DT1, DT2 or DT3 format field:

- An invalid SMF date can result in a data exception (0C7 ABEND) or an incorrect ZD date.
- SMF date values are always treated as positive.

For a DC1, DC2, DC3, DE1, DE2, or DE3 format field:

- TOD and ETOD date values are always treated as positive.

For a TM1, TM2, TM3 or TM4 format field:

- An invalid SMF time can result in an incorrect ZD time.
- SMF time values are always treated as positive.

For a TC1, TC2, TC3, TC4, TE1, TE2, TE3, or TE4 format field:

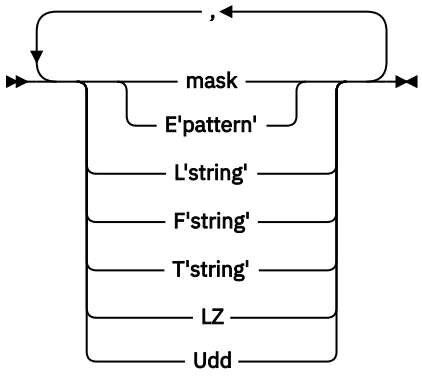
- TOD and ETOD time values are always treated as positive.

BREAK(p,m,f,formatting)

Specifies a numeric or character break field to be used to divide the report into sections, and how the data for this field is to be formatted for printing.

See BREAK(p,m,f) for further details.

formatting



specifies formatting items that indicate how the record number is to be formatted for printing. Formatting items can be specified in any order, but each item can only be specified once. Any formatting item can be specified for a numeric break field, but only L'string' and T'string' can be specified for a character break field.

mask
See ON(p,m,f,formatting) for a discussion of **mask**.

E'pattern'
See ON(p,m,f,formatting) for a discussion of **E'pattern'**.

L'string'
See ON(p,m,f,formatting) for a discussion of **L'string'**.

F'string'
See ON(p,m,f,formatting) for a discussion of **F'string'**.

T'string'
See ON(p,m,f,formatting) for a discussion of **T'string'**.

LZ
See ON(p,m,f,formatting) for a discussion of **LZ**.

Udd
specifies the number of digits to be used for a numeric break field. Udd can be used to change the column width for numeric break fields. dd specifies the number of digits and must be a two-digit number between 01 and 31.

The default number of digits (d) for a numeric break field is the maximum number of digits for that field. For example, d is 8 for BREAK(1,8,ZD). If you know that your break field requires less than d digits, you can use a lower number of digits (dd) instead by specifying Udd, thus reducing the break field width. For example, BREAK(1,8,ZD,U06) reduces d from 8 to 6. If you want your break field to be displayed with more than d digits, you can use a higher number of digits (dd) instead by specifying Udd, thus increasing the field width. For example, BREAK(1,8,ZD,U11) increases d from 8 to 11.

BTITLE('string')
Specifies a string to appear in the break title line printed for each page of a section. BTITLE can only be specified if BREAK is specified. The break value and break title string appear in the order in which you specify BREAK and BTITLE. Two blanks appear between break title elements. A blank line is printed after the break title line.

The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes (''). Blanks at the start of the string move the text to the right. Blanks at the end of the string increase the spacing between the string and the break value if BTITLE is specified before BREAK.

BTOTAL('string')
Specifies a break TOTAL (BTOTAL) line is to be printed after the rows of data for each section. BTOTAL can only be specified if BREAK is specified. The specified string is printed starting at

the indent column of the break TOTAL line, followed by the break total for each numeric data column.If STATLEFT is specified, the string is printed to the left of the first column of data with the totals on the same line as the string. If STATLEFT is not specified, the string is printed in the first column of data with the totals on the same line as the string, or on the next line, as appropriate. A blank line is printed before the break TOTAL line.

The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes (''). To suppress printing of a string, specify BTOTAL('') using two single apostrophes.

The break total for each numeric ON field is printed in the format (formatting, PLUS, BLANK, or standard) you specify. The total for a specific numeric field is suppressed if the NOST formatting item is specified for that field. Totals are printed for ON(VLEN) fields, but not for ON(NUM) fields.

The default number of digits (d) for a break total is 15 if the ON field is BI or FI with a length up to 4, PD with a length up to 8, or ZD, CSF, FS, UFF or SFF with a length up to 15. The default number of digits (d) for a break total is 31 if the ON field is BI or FI with a length greater than 4, PD with a length greater than 8, or ZD, CSF, FS, UFF or SFF with a length greater than 15. By default, column widths are adjusted to allow for a maximum of a sign and d digits for the totals. If the break total for an ON field overflows d digits, ICETOOL prints asterisks for the break total for that field and terminates the operation.

You can use the Ndd or Udd formatting item to decrease or increase the number of digits used for a break total. If you use Ndd or Udd and the break total for an ON field overflows dd digits, ICETOOL prints asterisks for the break total for that field and terminates the operation.

You can prevent overflow by specifying an appropriate dd value for Ndd or Udd. For example, if ON(1,15,ZD) with BTOTAL overflows the default of 15 digits, you can specify ON(1,15,ZD,U16) to prevent overflow. If ON(1,15,ZD,U16) still results in overflow, you can specify ON(1,15,ZD,U17), and so on.

Either Ndd or Udd can be used to set the number of digits greater than the maximum for a numeric field, but only Udd can be used to set the number of digits less than the maximum for a numeric field.

See the discussion of Ndd or Udd under ON(p,m,f,formatting) for more details on using Ndd or Udd with BTOTAL.

The BTOTAL, BMAXIMUM, BMINIMUM, BAVERAGE, and BCOUNT lines are printed in the order in which you specify them.

BMAXIMUM('string')

Specifies a break MAXIMUM line is to be printed after the rows of data for each section. BMAXIMUM can only be specified if BREAK is specified. The specified string is printed starting at the indent column of the break MAXIMUM line, followed by the break maximum for each numeric data column. If STATLEFT is specified, the string is printed to the left of the first column of data with the maximums on the same line as the string. If STATLEFT is not specified, the string is printed in the first column of data with the maximums on the same line as the string, or on the next line, as appropriate. A blank line is printed before the break MAXIMUM line.

The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes (''). To suppress printing of a string, specify BMAXIMUM('') using two single apostrophes.

The break maximum for each numeric ON field is printed in the format (formatting, PLUS, BLANK, or standard) you specify. The maximum for a specific numeric field is suppressed if the NOST formatting item is specified for that field. Maximums are printed for ON(VLEN) fields, but not for ON(NUM) fields.

The BTOTAL, BMAXIMUM, BMINIMUM, BAVERAGE, and BCOUNT lines are printed in the order in which you specify them.

BMINIMUM('string')
Specifies a break MINIMUM line is to be printed after the rows of data for each section. BMINIMUM can only be specified if BREAK is specified. The specified string is printed starting at the indent column of the break MINIMUM line, followed by the break minimum for each numeric data column. If STATLEFT is specified, the string is printed to the left of the first column of data with the minimums on the same line as the string. If STATLEFT is not specified, the string is printed in the first column of data with the minimums on the same line as the string, or on the next line, as appropriate. A blank line is printed before the break MINIMUM line.

The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes (''). To suppress printing of a string, specify BMINIMUM('') using two single apostrophes.

The break minimum for each numeric ON field is printed in the format (formatting, PLUS, BLANK, or standard) you specify. The minimum for a specific numeric field is suppressed if the NOST formatting item is specified for that field. Minimums are printed for ON(VLEN) fields, but not for ON(NUM) fields.

The BTOTAL, BMAXIMUM, BMINIMUM, BAVERAGE, and BCOUNT lines are printed in the order in which you specify them.

BAVERAGE('string')
Specifies a break AVERAGE line is to be printed after the rows of data for each section. BAVERAGE can only be specified if BREAK is specified. The specified string is printed starting at the indent column of the break AVERAGE line, followed by the break average for each numeric data column. If STATLEFT is specified, the string is printed to the left of the first column of data with the averages on the same line as the string. If STATLEFT is not specified, the string is printed in the first column of data with the averages on the same line as the string, or on the next line, as appropriate. A blank line is printed before the break AVERAGE line.

The break average (or mean) is calculated by dividing the break total by the number of values in the section and rounding down to the nearest integer (examples: 23 / 5 = 4, -23 / 5 = -4).

The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes (''). To suppress printing of a string, specify BAVERAGE('') using two single apostrophes.

The break average for each numeric ON field is printed in the format (formatting, PLUS, BLANK, or standard) you specify. The average for a specific numeric field is suppressed if the NOST formatting item is specified for that field. Averages are printed for ON(VLEN) fields, but not for ON(NUM) fields.

You can use the Ndd or Udd formatting item to decrease or increase the number of digits used for a break total. If the break total for an ON field overflows d digits, ICETOOL prints asterisks for the break average for that field and terminates the operation. You can prevent overflow by specifying an appropriate dd value for Ndd or Udd. For example, if ON(1,15,ZD) with BAVERAGE overflows the default of 15 digits for the total, you can specify ON(1,15,ZD,U16) to prevent overflow.

See the discussion of Ndd or Udd under ON(p,m,f,formatting) for more details on using Ndd or Udd.

The BTOTAL, BMAXIMUM, BMINIMUM, BAVERAGE, and BCOUNT lines are printed in the order in which you specify them.

BCOUNT('string')
Specifies a break COUNT line is to be printed after the rows of data for each section. BCOUNT can only be specified if BREAK is specified. The specified string is printed starting at the indent column of the break COUNT line, followed by the break count of data records in the section. If STATLEFT is specified, the string is printed to the left of the first column of data. If STATLEFT is not specified, the string is printed in the first column of data. The count is printed on the same line as the string. A blank line is printed before the break COUNT line.

The string (1 to 50 characters) must be enclosed in single apostrophes. To include a single apostrophe (') in the string, specify two single apostrophes (''). To suppress printing of a string, specify BCOUNT('') using two single apostrophes.

The count is printed in the format (PLUS, BLANK, or standard) you specify. EDBCOUNT(formatting) can be used to apply formatting items to the count. The default number of digits (d) for the count is 15.

The BTOTAL, BMAXIMUM, BMINIMUM, BAVERAGE, and BCOUNT lines are printed in the order in which you specify them.

EDBCOUNT(formatting)
Specifies how the break count is to be formatted for printing. Formatting items can be specified in any order, but each item can only be specified once. EDBCOUNT can only be specified if BCOUNT('string') is specified.

mask
See ON(p,m,f,formatting) for a discussion of **mask**.

E'pattern'
See ON(p,m,f,formatting) for a discussion of **E'pattern'**.

L'string'
See ON(p,m,f,formatting) for a discussion of **L'string'**.

F'string'
See ON(p,m,f,formatting) for a discussion of **F'string'**.

T'string'
See ON(p,m,f,formatting) for a discussion of **T'string'**.

LZ
See ON(p,m,f,formatting) for a discussion of **LZ**.

Udd
See EDCOUNT(formatting) for a discussion of **Udd**.

STATLEFT
Specifies that the strings for statistics (TOTAL, MAXIMUM, MINIMUM, AVERAGE, COUNT, BTOTAL, BMAXIMUM, BMINIMUM, BAVERAGE, BCOUNT) are to be placed to the left of the first column of data (overriding the default of placing the strings in the first column). STATLEFT ensures that each statistic appears on the same line as its string while making the statistics lines stand out from the columns of data.

UZERO
Specifies that -0 and +0 are to be treated as unsigned zero values, that is, as the same value. With UZERO, -0 and +0 are treated as positive for ON, MINIMUM, MAXIMUM, BREAK, BMINIMUM and BMAXIMUM processing.

UZERO overrides the default of treating -0 and +0 as signed zero values, that is, as different values. Without UZERO, -0 is treated as negative and +0 is treated as positive for ON, MINIMUM, MAXIMUM, BREAK, BMINIMUM and BMAXIMUM processing.

LISTSDB OR LISTNOSDB
Can be used to override the SDBMSG value for this LIST data set. LISTSDB directs ICETOOL to select the system-determined optimum block size for the LIST data set in the same way as for installation option SDBMSG=YES. LISTNOSDB directs ICETOOL to select the block size for the LIST data set in the same way as for installation option SDBMSG=NO. See the discussion of the LIST(listdd) operand previously in this section for more information on how LISTSDB or LISTNOSDB affects the LIST data set block size.

Attention: LISTSDB has no effect for SYSOUT data sets (for example, //RPT1 DD SYSOUT=*), because the system-determined optimum block size is not used for spool or dummy data sets.

DISPLAY examples

Although the DISPLAY operators in the examples in this section could all be contained in a single ICETOOL job step, they are shown and discussed separately for clarity. See “OCCUR operator” on page 612 for additional examples of tailoring the report format.

Example 1

```
DISPLAY FROM(SOURCE) LIST(FIELDS) ON(NUM) ON(40,12,CH) -
ON(20,8,PD)
```

Prints, in the FIELDS data set:

- A heading line containing the standard headings
- Data lines in the standard format containing:
  - The record number in the standard format
  - The characters from positions 40-51 of the SOURCE data set
  - The packed decimal values from positions 20-27 of the SOURCE data set in the standard format

The FIELDS output starts on a new page and looks as follows (the first 2 records are shown with illustrative values):

```
RECORD NUMBER     (40,12,CH)             (20,8,PD)
000000000000001   SAN JOSE               000000000003745
000000000000002   MORGAN HILL            000000000016502
              .   .                                  .
              .   .                                  .
              .   .                                  .
```

The heading line appears at the top of each page.

Example 2

```
 DISPLAY FROM(IN) LIST(LIST1) -
       TITLE('National Accounting Report') -
       PAGE DATE TIME -
       HEADER('Division') HEADER('Revenue') HEADER('Profit/Loss') -
       ON(1,25,CH)        ON(45,10,ZD)       ON(35,10,ZD) -
       BLANK -
       TOTAL('Company Totals') -
       AVERAGE('Company Averages')
```

Prints, in the LIST1 data set:

- A title line containing the specified title, the page number, the date and the time
- A heading line containing the specified underlined headings
- Data lines in the BLANK format containing:
  - The characters from positions 1-25 of the IN data set
  - The zoned decimal values from positions 45-54 of the IN data set
  - The zoned decimal values from positions 35-44 of the IN data set
- A TOTAL line containing the specified string and the total for each of the two zoned decimal fields in the BLANK format
- An AVERAGE line containing the specified string and the average for each of the two zoned decimal fields in the BLANK format.

The LIST1 output starts on a new page and looks as follows (the first 2 records are shown with illustrative values):

```
National Accounting Report        - 1 -        02/21/05        18:52:44

Division                             Revenue        Profit/Loss
-------------------------   ----------------   ----------------
Research and Development            54323456            -823325
Manufacturing                      159257631            1372610
.                                          .                  .
.                                          .                  .
.                                          .                  .

Company Totals                     612867321            5277836

Company Averages                    76608415             659729
```

The title line and underlined heading line appear at the top of each page.

Example 3

```
     DISPLAY FROM(DATA) LIST(JUSTDATA) -
       NOHEADER -
       ON(17,5,PD) ON(1,2,FI)
```

Prints, in the JUSTDATA data set:

- Data lines in the standard format containing:
  - The packed decimal values from positions 17-21 of the DATA data set in the standard format
  - The fixed-point values from positions 1-2 of the DATA data set in the standard format

The JUSTDATA output contains no page ejects or heading lines and looks as follows (the first 2 records are shown with illustrative values):

```
-0000000000273216   +0000000000000027
+0000000000993112   +0000000000000321
                .                   .
                .                   .
                .                   .
```

Example 4

```
   COPY FROM(INPUT) TO(TEMP) USING(TREG)
   DISPLAY FROM(TEMP) LIST(REGULAR) -
     PAGE TITLE('Report on Regular Tools') TBETWEEN(12)-
     HEADER(NONE) ON(1,18,CH) -
     HEADER(,'Item') ON(35,5,CH) -
     HEADER('Percent','Change') ON(28,4,FS,B1) -
     LINES(66)
   COPY FROM(INPUT) TO(TEMP) USING(TPOW)
   DISPLAY FROM(TEMP) LIST(POWER) -
     PAGE TITLE('Report on Power Tools ') TBETWEEN(12)-
     HEADER(NONE) ON(1,18,CH) -
     HEADER(,'Item') ON(35,5,CH) -
     HEADER('Percent','Change') ON(28,4,FS,B1) -
     LINES(66)
```

This example shows how reports for different subsets of data can be produced. Assume that:

- The TREGCNTL data set contains:

```
   INCLUDE COND=(44,8,CH,EQ,C'Regular')
```

- The TPOWCNTL data set contains:

```
   INCLUDE COND=(44,8,CH,EQ,C'Power')
```

The first COPY operator copies the records from the INPUT data set that contain 'Regular ' in positions 44-51 to the TEMP (temporary) data set

The first DISPLAY operator uses the first subset of records in the TEMP data set to print, in the REGULAR data set:

- A title line containing the page number and specified title, with twelve blanks between these report elements.
- A two-line heading containing the specified underlined strings (with no heading for the first ON field)). Note the comma in HEADER(,'Item') to place 'Item' on line2 of the heading.
- Data lines for the first subset of records containing:
  - The characters from positions 1-18
  - The characters from positions 35-39
  - The floating sign values from positions 28-31 formatted with one decimal place and a period as the decimal point

The second COPY operator copies the records from the INPUT data set that contain 'Power ' in positions 44-51 to the TEMP (temporary) data set

The second DISPLAY operator uses the second subset of records in the TEMP data set to print, in the POWER data set:

- A title line containing the page number and specified title, with twelve blanks between these report elements.
- A two-line heading containing the specified underlined strings (with no heading for the first ON field)). Note the comma in HEADER(,'Item') to place 'Item' on line2 of the heading.
- Data lines for the second subset of records containing:
  - The characters from positions 1-18
  - The characters from positions 35-39
  - The floating sign values from positions 28-31 formatted with one decimal place and a period as the decimal point

The REGULAR output starts on a new page and looks as follows (the first 2 records are shown with illustrative values):

```
- 1 -            Report on Regular Tools

                             Percent
                      Item    Change
                      -----   -------
Hammers               10325      -7.3
Wrenches              00273      15.8
```

The title line and underlined heading lines appear at the top of each page. The number of lines per page is 66, overriding the default of 58.

The POWER output starts on a new page and looks as follows (the first 2 records are shown with illustrative values):

```
- 1 -            Report on Power Tools

                             Percent
                      Item    Change
                      -----   -------
Hammers               10325       9.8
Wrenches              00273     123.0
```

The title line and underlined heading lines appear at the top of each page. The number of lines per page is 66, overriding the default of 58.

Example 5

```
      DISPLAY FROM(INV) LIST(RDWLIST1) -
        TITLE('No Frills RDW Report') -
        ON(NUM) -
        ON(VLEN) -
        ON(1,4,HEX) -
```

```
       MINIMUM('Smallest') -
       MAXIMUM('Largest')
```

Prints, in the RDWLIST1 data set:

- A title line containing the specified title
- A heading line containing the standard headings
- Data lines in the standard format containing:
  - The record number
  - The record length
  - The record descriptor word (RDW) in hexadecimal
- A MINIMUM line containing the specified string and the minimum record length in the standard format
- A MAXIMUM line containing the specified string and the maximum record length in the standard format.

The RDWLIST1 output starts on a new page and looks as follows (the first 2 records are shown with illustrative values):

```
No Frills RDW Report

RECORD NUMBER     RECORD LENGTH     (1,4,HEX)
000000000000001   +000000000000075  004B0000
000000000000002   +000000000000071  00470000
              .                 .   .
              .                 .   .
              .                 .   .

Smallest          +000000000000058

Largest           +000000000000078
```

The title line and heading line appear at the top of each page.

Example 6

```
     DISPLAY FROM(INV) LIST(RDWLIST2) -
       DATE(DMY.) -
       TFIRST TBETWEEN(3) -
       TITLE('Fancy RDW Report with') -
       TITLE('length in decimal and hex') -
       TITLE('and max and min length') -
       TIME(12:) -
       HEADER('Relative Record') ON(NUM) -
       HEADER('    RDW (length)') ON(VLEN) -
       HEADER('RDW (Hex)') ON(1,4,HEX) -
       BLANK -
       MINIMUM('Smallest Record:') -
       MAXIMUM('Largest Record:') -
       COUNT('Number of Records: ') EDCOUNT(U04)
```

Prints, in the RDWLIST2 data set:

- Title lines containing the date, the specified title strings and the time
- A heading line containing the specified underlined headings
- Data lines in the BLANK format containing:
  - The record number
  - The record length
  - The record descriptor word (RDW) in hexadecimal
- A MINIMUM line containing the specified string and the minimum record length in the BLANK format
- A MAXIMUM line containing the specified string and the maximum record length in the BLANK format
- A COUNT line containing the specified string and the edited overall record count.

RDWLIST2 output starts on a new page and looks as follows (the first 2 records are shown with illustrative values):

```
22.07.08     Fancy RDW Report with     05:37:43 pm

           length in decimal and hex

            and max and min length

Relative Record       RDW (length)   RDW (Hex)
---------------   ----------------   ---------
              1                 84   00540000
              2                 47   002F0000
              3                 31   001F0000
              4                 31   001F0000
              5                 31   001F0000
...

Relative Record       RDW (length)   RDW (Hex)
---------------   ----------------   ---------
             51                 31   001F0000
             52                 31   001F0000
             53                 31   001F0000
             54                 31   001F0000
             55                 31   001F0000
             56                 31   001F0000

Smallest Record:                31

Largest Record:                 84

Number of Records:    56
```

The title lines only appear at the top of the first page as specified by the TFIRST operand. The underlined heading line appears at the top of each page.

Example 7

```
    SORT FROM(PARTS) TO(TEMP) USING(SRT1)
    DISPLAY FROM(TEMP) LIST(USA) -
      TITLE('Parts Completion Report for USA') DATE -
      HEADER('Part')  HEADER('Completed')  HEADER('Value ($)') -
      ON(15,6,CH)     ON(3,4,ZD,A1)        ON(38,8,ZD,C1) -
      TOTAL('Total:')
    DISPLAY FROM(TEMP) LIST(FRANCE) -
      TITLE('Parts Completion Report for France') DATE(DM4/) -
      HEADER('Part')  HEADER('Completed')  HEADER('Value (F)') -
      ON(15,6,CH)     ON(3,4,ZD,A3)        ON(38,8,ZD,C3) -
      TOTAL('Total:')
    DISPLAY FROM(TEMP) LIST(DENMARK) -
      TITLE('Parts Completion Report for Denmark') DATE(DMY-) -
      HEADER('Part')  HEADER('Completed')  HEADER('Value (kr)') -
      ON(15,6,CH)     ON(3,4,ZD,A2)        ON(38,8,ZD,C2) -
      TOTAL('Total:')
```

This example shows how reports for three different countries can be produced. The reports differ only in the way that date and numeric values are displayed.

Assume that the SRT1CNTL data set contains:

```
  SORT FIELDS=(15,6,CH,A)
```

The SORT operator sorts the PARTS data set to the TEMP data set using the SORT statement in SRT1CNTL.

The first DISPLAY operator uses the sorted records in the TEMP data set to print, in the USA data set:

- A title line containing the specified title and the date in the format commonly used in the United States
- A heading line containing the specified underlined headings
- Data lines containing:
  - The characters from positions 15-20

  - The zoned decimal values from positions 3-6 formatted with the separators commonly used in the United States
  - The zoned decimal values from positions 38-45 formatted with two decimal places and the separators and decimal point commonly used in the United States.
- A TOTAL line containing the specified string and the total for each of the two zoned decimal fields formatted in the same way as the data values.

The second DISPLAY operator uses the sorted records in the TEMP data set to print, in the FRANCE data set:

- A title line containing the specified title and the date in the format commonly used in France
- A heading line containing the specified underlined headings
- Data lines containing:
  - The characters from positions 15-20
  - The zoned decimal values from positions 3-6 formatted with the separators commonly used in France
  - The zoned decimal values from positions 38-45 formatted with two decimal places and the separators and decimal point commonly used in France.
- A TOTAL line containing the specified string and the total for each of the two zoned decimal fields formatted in the same way as the data values.

The third DISPLAY operator uses the sorted records in the TEMP data set to print, in the DENMARK data set:

- A title line containing the specified title and the date in the format commonly used in Denmark
- A heading line containing the specified underlined headings
- Data lines containing:
  - The characters from positions 15-20
  - The zoned decimal values from positions 3-6 formatted with the separators commonly used in Denmark
  - The zoned decimal values from positions 38-45 formatted with two decimal places and the separators and decimal point commonly used in Denmark.
- A TOTAL line containing the specified string and the total for each of the two zoned decimal fields formatted in the same way as the data values.

The USA output starts on a new page and looks as follows (several records are shown with illustrative values):

```
Parts Completion Report for USA        01/14/05

Part                Completed               Value ($)
------   --------------------   --------------------
000310                    562                8,317.53
001184                  1,234               23,456.78
029633                     35                  642.10
192199                  3,150              121,934.65
821356                    233                2,212.34

Total:                  5,214              156,563.40
```

The title line and underlined heading line appear at the top of each page.

The FRANCE output starts on a new page and looks as follows (several record are shown with illustrative values):

```
Parts Completion Report for France        14/01/2005

Part                Completed               Value (F)
------   --------------------   ---------------------
000310                    562                 8 317,53
001184                  1 234                23 456,78
029633                     35                   642,10
192199                  3 150               121 934,65
821356                    233                 2 212,34

Total:                  5 214               156 563,40
```

The title line and underlined heading line appear at the top of each page.

The DENMARK output starts on a new page and looks as follows (several records are shown with illustrative values):

```
Parts Completion Report for Denmark        14-01-05

Part                Completed               Value (kr)
------   --------------------   ---------------------
000310                    562                 8.317,53
001184                  1.234                23.456,78
029633                     35                   642,10
192199                  3.150               121.934,65
821356                    233                 2.212,34

Total:                  5.214               156.563,40
```

The title line and underlined heading line appear at the top of each page.

Example 8

```
  SORT FROM(DATA) TO(TEMP) USING(SRTX)
  DISPLAY FROM(TEMP) LIST(WEST) -
    DATE TITLE('Western Region Profit/Loss Report') PAGE -
    BTITLE('Division:') BREAK(3,10,CH) -
    HEADER('Branch Office') ON(16,13,CH) -
    HEADER('Profit/Loss (K)') ON(41,4,PD,/K,E1) -
    BMINIMUM('Lowest Profit/Loss in this Division:') -
    BMAXIMUM('Highest Profit/Loss in this Division:') -
    BAVERAGE('Average Profit/Loss for this Division:') -
    MINIMUM('Lowest Profit/Loss for all Divisions:') -
    MAXIMUM('Highest Profit/Loss for all Divisions:') -
    AVERAGE('Average Profit/Loss for all Divisions:')
```

This example shows how a report with sections can be produced.

Assume that the SRTXCNTL data set contains:

```
  SORT FIELDS=(3,10,A,16,13,A),FORMAT=CH
```

The SORT operator sorts the DATA data set to the TEMP data set using the SORT statement in SRTXCNTL.

The DISPLAY operator uses the sorted records in the TEMP data set to print, in the WEST data set, sections with:

- A title line containing the date, the specified title string, and the page number
- A break title containing the specified break title string, and the break field characters from positions 3-12
- A heading line containing the specified underlined headings
- Data lines containing:
  - The characters from positions 16-28
  - The packed decimal values from positions 41-44 divided by 1000 and formatted with separators and signs as specified.
- Break MINIMUM, MAXIMUM, and AVERAGE lines containing the specified strings and statistics for the packed decimal field values in this section, formatted in the same way as the data values.

The last page of the report contains:

- A title line containing the date, the specified title string, and the page number
- A heading line containing the specified underlined headings
- Overall MINIMUM, MAXIMUM, and AVERAGE lines containing the specified strings and statistics for the packed decimal field values in the report, formatted in the same way as the data values.

The first section of the WEST output starts on a new page and looks as follows (several records are shown with illustrative values):

```
01/14/05        Western Region Profit/Loss Report        - 1 -

Division:  Chips

Branch Office   Profit/Loss (K)
-------------   ---------------
Gilroy                    3,293
Los Angeles                (141)
Morgan Hill                 213
Oakland                   1,067
San Francisco               (31)
San Jose                     92
San Martin                1,535

Lowest Profit/Loss in this Division:
                           (141)

Highest Profit/Loss in this Division:
                          3,293

Average Profit/Loss for this Division:
                            861
```

The title line, break title line, and underlined heading line appear at the top of each page of the section.

The second section of the WEST output starts on a new page and looks as follows (several records are shown with illustrative values):

```
01/14/05        Western Region Profit/Loss Report        - 2 -

Division:  Ice Cream

Branch Office   Profit/Loss (K)
-------------   ---------------
Marin                       673
Napa                         95
San Francisco              (321)
San Jose                  2,318
San Martin                   21

Lowest Profit/Loss in this Division:
                           (321)

Highest Profit/Loss in this Division:
                          2,318

Average Profit/Loss for this Division:
                            557
```

The title line, break title line, and underlined heading line appear at the top of each page of the section.

The last page of the WEST output starts on a new page and looks as follows:

```
01/14/05        Western Region Profit/Loss Report        - 3 -

Branch Office   Profit/Loss (K)
-------------   ---------------

Lowest Profit/Loss for all Divisions:
                          (321)

Highest Profit/Loss for all Divisions:
                         3,293

Average Profit/Loss for all Divisions:
                           734
```

Example 9

```
MODE CONTINUE
VERIFY FROM(CHECK) ON(2,3,PD) LIMIT(500)
DISPLAY FROM(CHECK) LIST(PDREPORT) BLANK LIMIT(500) -
  HEADER('Relative Record') ON(NUM) -
  HEADER('Numeric') ON(2,3,PD) -
  HEADER('Hexadecimal') ON(2,3,HEX) -
  HEADER('Associated Field') ON(21,20,CH)
```

This example shows how each record containing an invalid decimal value can be identified either by its relative record number or an associated field in the record.

The MODE operator ensures that the DISPLAY operator is processed if the VERIFY operator identifies an invalid decimal value.

The VERIFY operator checks for invalid digits (A-F) and invalid signs (0-9) in the packed decimal values from positions 2-4 of the CHECK data set. Messages ICE618A and ICE649A are printed in the TOOLMSG data set for each value (if any) that contains an invalid digit or sign. If 500 invalid values are found, the operation is terminated.

The DISPLAY operator checks for invalid digits (A-F) in the packed decimal values from positions 2-4 of the CHECK data set. Messages ICE618A and ICE649A are printed in the TOOLMSG data set for each value (if any) that contains an invalid digit. If 500 invalid values are found, the operation is terminated. If a check for invalid signs is required, the VERIFY operator must be used, because the DISPLAY operator only checks for invalid digits. The VERIFY operator is not required if signs need not be checked.

The DISPLAY operator also prints, in the PDREPORT data set:

- A heading line containing the specified underlined headings
- Data lines in the BLANK format containing:
  - The relative record number. This number can be matched against the RECORD numbers printed in the ICE618A messages to find the records with invalid signs.
  - The numeric representation of the packed decimal value in positions 2-4. Asterisks are shown for values with invalid digits, making them easy to identify. Asterisks are not shown for values with invalid signs; these must be identified by matching the relative record number against the RECORD number in ICE618A.
  - The hexadecimal representation of the packed decimal value in positions 2-4 (also shown in ICE649A). This makes it easy to find the specific hexadecimal digits or signs that are invalid.
  - The characters in positions 21-40. An associated field such as this can be used to make identification of the records with invalid values easier.

The ICE618A and ICE649A messages in TOOLMSG for the VERIFY operator are:

```
ICE618A 0 INVALID (2,3,PD)       VALUE - RECORD:  000000000000003
ICE649A 0   HEX VALUE:  53A54C
ICE618A 0 INVALID (2,3,PD)       VALUE - RECORD:  000000000000012
ICE649A 0   HEX VALUE:  621540
ICE618A 0 INVALID (2,3,PD)       VALUE - RECORD:  000000000000019
ICE649A 0   HEX VALUE:  400F3C
```

The ICE618A and ICE649A messages in TOOLMSG for the DISPLAY operator are:

```
ICE618A 0 INVALID (2,3,PD)       VALUE - RECORD:  000000000000003
ICE649A 0   HEX VALUE:  53A54C
ICE618A 0 INVALID (2,3,PD)       VALUE - RECORD:  000000000000019
ICE649A 0   HEX VALUE:  400F3C
```

The PDREPORT output looks as follows:

```
Relative Record   Numeric   Hexadecimal   Associated Field
---------------   -------   -----------   --------------------
              1     18600   18600C        Wagar
              2       -93   00093B        Gellai
              3    ******   53A54C        Giulianelli
              4     86399   86399C        Mehta
              5     24215   24215F        Johnson
              6      8351   08351C        Packer
              7     19003   19003C        Childers
              8    -31285   31285D        Burg
              9     88316   88316C        Monkman
             10      1860   01860C        Vezinaw
             11    -29285   29285D        Mead
             12     62154   621540        Wu
             13      -328   00328D        Madrid
             14    -11010   11010D        Warren
             15      1363   01363F        Burt
             16     92132   92132C        Mao
             17    -48500   48500D        Shen
             18       -55   00055D        Yamamoto-Smith
             19    ******   400F3C        Yaeger
             20     33218   33218C        Leung
             21     96031   96031C        Kaspar
```

PDREPORT can be used in conjunction with the ICE618A and ICE649A messages to identify that:

- Record 3 has an invalid digit of A and an associated field of "Giulianelli"
- Record 12 has an invalid sign of 0 and an associated field of "Wu"
- Record 19 has an invalid digit of F and an associated field of "Yaeger".

Example 10

```
  COPY FROM(IN) USING(OUTF)
  DISPLAY FROM(TEMP) LIST(EMPCT) BLANK -
        TITLE('Employees by Function') -
        DATE -
        HEADER('Function') HEADER('Employees') -
        ON(1,25,CH)        ON(30,4,ZD)
```

This example shows how the OUTFIL table lookup feature can be used to substitute meaningful phrases for cryptic values in ICETOOL reports. Assume that:

- The OUTFCNTL data set contains:

```
  OUTFIL FNAMES=TEMP,
    OUTREC=(1:9,2,CHANGE=(25,
                  C'MN',C'Manufacturing',
                  C'RD',C'Research and Development',
                  C'FN',C'Finance',
                  C'MR',C'Marketing',
                  C'IS',C'Information Systems'),
            30:4,4)
```

The COPY operator uses the OUTFIL statement in OUTFCNTL to reformat the IN data set records to the TEMP (temporary) data set. Two fields are extracted for use by the DISPLAY operator:

- The 2-character department code in positions 9-10 is changed to a 25-character name in positions 1-25 using the table lookup feature.
- The zoned decimal value in positions 4-7 is moved to positions 30-33.

The DISPLAY operator uses the reformatted fields in the TEMP data set to print, in the EMPCT data set:

- A title line containing the specified title and the date
- A heading line containing the specified underlined headings
- Data lines in the BLANK format containing:
  - The names from positions 1-25 that were substituted for the department codes
  - The zoned decimal values from positions 30-33.

The EMPCT output starts on a new page and looks as follows:

```
Employees by Function        02/14/05

Function                   Employees
-------------------------  ---------
Manufacturing                    486
Marketing                         21
Research and Development          55
Information Systems              123
Finance                           33
```

Example 11

```
    DISPLAY FROM(ACCTS) LIST(PLAIN) -
        TITLE('Accounts Report for First Quarter') -
        DATE(MD4/) BLANK -
        HEADER('Amount') ON(12,6,ZD) -
        HEADER(Id') ON(NUM) -
        HEADER('Acct#') ON(31,3,PD) -
        HEADER('Date') ON(1,4,ZD) -
        TOTAL('Total for Q1') -
        AVERAGE('Average for Q1')

 DISPLAY FROM(ACCTS) LIST(FANCY) -
        TITLE('Accounts Report for First Quarter') -
        DATE(MD4/) BLANK -
        HEADER('Amount') ON(12,6,ZD,C1,N08) -
        HEADER(Id') ON(NUM,N02) -
        HEADER('Acct#') ON(31,3,PD,NOST,LZ) -
        HEADER('Date') ON(1,4,ZD,E'99/99',NOST) -
        INDENT(2) BETWEEN(5) -
        STATLEFT -
        TOTAL('Total for Q1') -
        AVERAGE('Average for Q1')
```

This example shows some options you can use to improve the appearance of a DISPLAY report. The first DISPLAY operator produces a "plain" report, and the second DISPLAY operator uses the options shown in **bold** to produce a "fancy" report.

The PLAIN output starts on a new page and looks as follows:

```
Accounts Report for First Quarter              05/04/2001

         Amount                 Id                  Acct#                    Date
---------------    ---------------    -------------------    --------------------
          93271                  1                  15932                     106
         137622                  2                    187                     128
          83147                  3                  15932                     212
         183261                  4                   2158                     217
          76389                  5                    187                     305
         920013                  6                  15932                     319

Total for Q1
        1493703                                     50328                    1287

Average for Q1
         248950                                      8388                     214
```

The FANCY output starts on a new page and looks as follows:

```
Accounts Report for First Quarter             05/04/2001

                       Amount         Id        Acct#      Date
                     --------        ---       ------     -----
                       932.71          1        15932     01/06
                     1,376.22          2        00187     01/28
                       831.47          3        15932     02/12
                     1,832.61          4        02158     02/17
                       763.89          5        00187     03/05
                     9,200.13          6        15932     03/19

Total for Q1        14,937.03

Average for Q1       2,489.50
```

Here is an explanation of the extra options used for the "fancy" report:

- First ON field: In the PLAIN report, BLANK causes ICETOOL to print the 6-byte ZD values as unedited digits with leading zeros suppressed. But for this example, we know the digits really represent dollars and cents. So in the FANCY report, we use the **C1** formatting item (one of thirty-three available masks) to print the values with a comma (,) as the thousands separator and a period (.) as the decimal point.

  In the PLAIN report, TOTAL causes ICETOOL to allow 15 digits for the values because it does not know how many digits are needed. But for this example, we know the total amount will not exceed 8 digits. So in the FANCY report, we use the **N08** formatting item to set the number of digits to 8. This decreases the column width for the field.
- Second ON field: In the PLAIN report, NUM causes ICETOOL to allow 15 digits for the record number because it does not know how many digits are needed. But for this example, we know the number of records will not exceed 99. So in the FANCY report, we use the **N02** formatting item to set the number of digits to 2. This decreases the column width for the record number.
- Third ON field: In the PLAIN report, TOTAL and AVERAGE cause ICETOOL to print the total and average for this 3-byte PD field. But for this example, we know we do not want statistics for the field because it is an account number. So in the FANCY report, we use the **NOST** formatting item to suppress the statistics for this field.

  In the PLAIN report, the default mask of A0 causes ICETOOL to suppress leading zeros for this 3-byte PD field. But for this example, we know that we want to show leading zeros for the field because it is an account number. So in the FANCY report, we use the **LZ** formatting item to print leading zeros for this field.
- Fourth ON field: In the PLAIN report, BLANK causes ICETOOL to print the 4-byte ZD values as unedited digits with leading zeros suppressed. But for this example, we know the digits represent a date (month and day). So in the FANCY report, we use the **E'99/99'** formatting item to print the values with leading zeros and a slash (/) between the month and day.

  In the PLAIN report, TOTAL and AVERAGE cause ICETOOL to print the total and average for this 4-byte ZD field. But for this example, we know we do not want the total or average for this field because it is a date. So in the FANCY report, we use the **NOST** formatting item to suppress the statistics for this field.

  Note: In some applications, we might want the minimum and maximum for a date displayed with E'pattern', so we would not specify NOST for the date field.
- INDENT: In the PLAIN report, ICETOOL starts the report in column 2 (after the control character), by default. But for this example, we want to indent the report a bit. So in the FANCY report, we use the **INDENT(2)** operand to indent the report by 2 blanks so it starts in column 4.
- BETWEEN: In the PLAIN report, ICETOOL uses 3 blanks between the columns of data, by default. But for this example, we want more space between the columns. So in the FANCY report, we use the **BETWEEN(5)** operand to insert 5 blanks between the columns.
- STATLEFT: In the PLAIN report, ICETOOL prints the strings for TOTAL and AVERAGE under the first column of data, by default, and uses two lines for each statistic to avoid having the string overlay the value. But for this example, we would like to have the TOTAL and AVERAGE strings stand out in the report and also have each string on the same line as its value. So in the FANCY report, we use the **STATLEFT** operand to print the TOTAL and AVERAGE strings to the left of the first column of data.

Example 12

```
SORT FROM(RAWSMF) TO(SMF14) USING(SMFI)
DISPLAY FROM(SMF14) LIST(SMF14RPT) -
      TITLE('SMF Type-14 Records') DATE(4MD/) -
      HEADER('Date') ON(11,4,DT1,E'9999/99/99') -
      HEADER('Time') ON(7,4,TM1,E'99:99:99') -
      HEADER('Sys') ON(15,4,CH) -
      HEADER('Jobname') ON(19,8,CH) -
      HEADER('Datasetname') ON(69,44,CH)
```

This example shows how SMF date and time values can be displayed in a meaningful way in a report on SMF type-14 records.

The SORT operator selects the type-14 records from the RAWSMF data set and sorts them by date and time to the SMF14 data set. It uses the following control statements in SMFICNTL:

```
INCLUDE COND=(6,1,BI,EQ,14)
SORT FIELDS=(11,4,PD,A,7,4,BI,A)
```

The DISPLAY operator uses the selected type-14 records in SMF14 to print, in the SMF14RPT data set:

- A title line containing the specified title and the date
- A heading line containing the specified underlined headings
- Data lines containing:
  - The SMF date values in positions 11-14 displayed as C'yyyy/mm/dd'
  - The SMF time values in positions 7-10 displayed as C'hh:mm:ss'
  - The character values in positions 15-18
  - The character values in positions 19-26
  - The character values in positions 69-112

The SMF14RPT output starts on a new page and looks as follows:

```
SMF Type-14 Records        2001/04/24

      Date       Time   Sys    Jobname    Datasetname
----------   --------   ----   --------   ----------- ...
2001/04/20   06:03:15   ID03   JOB00003   SYS1.QRS
2001/04/20   10:03:22   ID02   JOB00002   SYS1.XYZ
2001/04/21   14:05:37   ID03   JOB00004   SYS1.MNO
2001/04/21   22:11:00   ID03   JOB00005   SYS1.MNO
2001/04/24   00:00:08   ID03   JOB00006   SYS1.MNO
```

Note: When you use SMF date formats (DTn) or SMF time formats (TMn), the SMF values are treated as numeric. This allows you to use numeric formatting items such as masks and patterns to edit the SMF values. By default, DTn and TMn headings, like other numeric headings, appear right-aligned as shown in the SMF14RPT output example shown previously in this section. If you want to center-align or left-align headings for numeric values, you can add an appropriate number of blanks at the end of HEADER('string1').

For example, if you wanted to left-align the SMF date heading, you could use six blanks at the end of the header string like so:

```
   HEADER('Date     ')
```

to get the following heading:

```
   Date
   ----------
```

If you wanted to center-align the SMF date heading, you could use three blanks at the end of the header string like so:

```
   HEADER('Date  ')
```

to get the following heading:

```
      Date
   ----------
```

Example 13

```
SORT FROM(SMFIN) TO(SMF71) USING(TY71)
DISPLAY FROM(SMF71) LIST(SMF71RPT) -
TITLE('Low impact central storage frames') -
BREAK(15,4,CH,L'System: ') -
HEADER('Date') ON(11,4,DT1,E'9999-99-99') -
HEADER('Time') ON(7,4,TM1,E'99:99:99') -
HEADER('Min Frames') ON(925,8,FL,U10) -
HEADER('Max Frames') ON(933,8,FL,U10) -
HEADER('Avg Frames') ON(941,8,FL,U10) -
BLANK PAGE
```

This example shows how floating point values can be displayed as integers in a report on SMF type-71 records with a section for each system id.

The SORT operator selects SMF type-71 records that are at least 19 bytes long and sorts them by system id, date and time to the SMF71 data set. It uses the following control statements in TY71CNTL:

```
OMIT COND=(6,1,BI,NE,+71,OR,1,2,BI,LE,+18)
SORT FIELDS=(15,4,CH,A,11,4,PD,A,7,4,BI,A)
```

The DISPLAY operator uses the selected type-71 records in SMF71 to print, in the SMF71RPT data set:

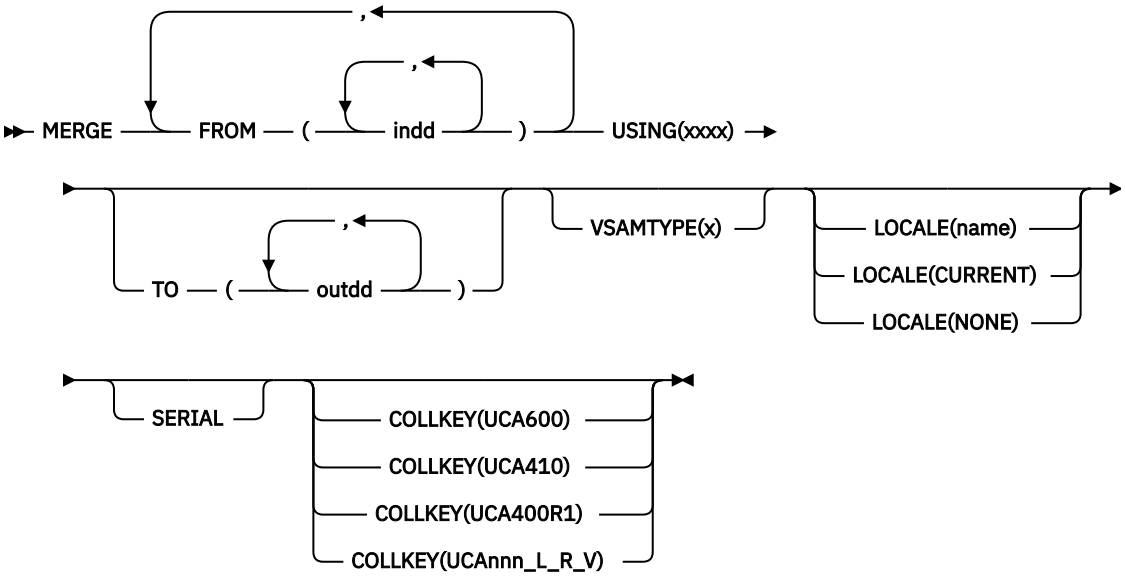
- A title line containing the specified title and the page number.
- A break title containing the specified leading string and the SMF71SID system id character values in positions 15-18.
- A heading line containing the specified underlined headings.
- Data lines containing:
  - The SMF71DTE date value. This SMF date value in positions 11-14 is displayed as a C'yyyy-mm-dd' value.
  - The SMF71TME time value. This SMF time value in positions 7-10 is displayed as a C'hh:mm:ss' value.
  - The SMF71CLM minimum number of low-impact central storage frames value. This floating-point value in positions 925-932 is displayed as a 10 digit integer value. (The U10 formatting item reduces the number of digits for the integer representation of the floating-point value from 20 to 10, decreasing the column width for the field.)
  - The SMF71CLX maximum number of low-impact central storage frames value. This floating-point value in positions 933-940 is displayed as a 10 digit integer value.
  - The SMF71CLA average number of low-impact central storage frames value. This floating-point value in positions 941-948 is displayed as a 10 digit integer value.

Each system id value starts on a new page and looks as follows (several sections and records are shown with illustrative values):

```
Low impact central storage frames        - 1 -

System: SYSA

      Date       Time    Min Frames    Max Frames    Avg Frames
----------   --------   -----------   -----------   -----------
2005-08-01   11:45:00        934215       1001596        963434
2005-08-01   12:00:00        971599       1004939        984437
2005-08-01   12:15:00        970192        982565        973768

Low impact central storage frames        - 2 -
```

```
System: SYSB

      Date       Time    Min Frames    Max Frames    Avg Frames
----------   --------   -----------   -----------   -----------
2005-08-01   11:45:00        947220        985444        966581
2005-08-01   12:00:00        980120       1018982        986360
2005-08-01   12:15:00        980387       1051920       1011873
```

MERGE operator



Merges up to 50 input data sets to an output data set. The records in each input data set to be merged must already be in sorted order as specified by the control fields in a supplied DFSORT MERGE statement.

You must specify at least one FROM operand and a USING(xxxx) operand. Each FROM operand can be used to specify one or more ddnames. You can specify up to 10 FROM operands. The maximum number of ddnames in all of the FROM operands must not exceed 50.

DFSORT is called to merge the indd data sets to the outdd data sets using the DFSORT control statements in xxxxCNTL. You must supply a DFSORT MERGE statement in the xxxxCNTL data set to indicate the control fields for the merge. You can use additional DFSORT statements in the xxxxCNTL data set to merge a subset of the input records (INCLUDE or OMIT statement; OUTFIL INCLUDE, OMIT, SAVE, STARTREC, ENDREC, SAMPLE, SPLIT, SPLITBY and SPLIT1R operands), reformat records for output (INREC, OUTREC, and OUTFIL statements; user exit routines), and so on.

If EQUALS is in effect, records that collate identically are output in the order of their ddnames in the FROM operands.

The active locale's collating rules affect MERGE processing as explained in “MERGE control statement” on page 163. If an INCLUDE or OMIT statement or an OUTFIL INCLUDE or OMIT operand is specified in the xxxxCNTL data set, the active locale's collating rules affect INCLUDE and OMIT processing as explained in the "Cultural Environment Considerations" discussion in “INCLUDE control statement” on page 91.

Note: For a merge application, records deleted during an E35 exit routine are not sequence checked. If you use an E35 exit routine without an output data set, sequence checking is not performed at the time the records are passed to the E35 user exit; therefore, you must ensure that input records are in correct sequence.

Operand descriptions

The operands described in this section can be specified in any order:

FROM(indd,...)
Specifies the ddnames of the input data sets to be read by DFSORT for this operation. Up to 10 FROM operands can be used to specify up to 50 input ddnames. An indd DD statement must be present for each indd name specified. Each indd input data set must conform to the rules for DFSORT's SORTINnn data sets.

USING(xxxx)
Specifies the first 4 characters of the ddname for the control statement data set to be used by DFSORT for this operation. xxxx must be four characters that are valid in a ddname of the form xxxxCNTL. xxxx must not be SYSx.

An xxxxCNTL DD statement must be present, and the control statements in it must conform to the rules for DFSORT's SORTCNTL data set.

The xxxxCNTL data set must contain a MERGE statement. If TO is not specified, the xxxxCNTL data set must also contain either one or more OUTFIL statements or a MODS statement for an E35 routine that disposes of all records. Other statements are optional.

TO(outdd,...)
Specifies the ddnames of the output data sets to be written by DFSORT for this operation. From 1 to 10 outdd names can be specified. An outdd DD statement must be present for each outdd name specified. If a single outdd data set is specified, DFSORT is called once to merge the indd data sets to the outdd data set using SORTOUT processing; the outdd data set must conform to the rules for DFSORT's SORTOUT data set. If multiple outdd data sets are specified and SERIAL is not specified, DFSORT is called once to merge the indd data sets to the outdd data sets using OUTFIL processing; the outdd data sets must conform to the rules for DFSORT's OUTFIL data sets.

A ddname specified in a FROM operand must not also be specified in the TO operand.

VSAMTYPE(x)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

LOCALE(name)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

LOCALE(CURRENT)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

LOCALE(NONE)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

SERIAL
Specifies that OUTFIL processing is not to be used when multiple outdd data sets are specified. DFSORT is called multiple times and uses SORTOUT processing; the outdd data sets must conform to the rules for DFSORT's SORTOUT data set. SERIAL is not recommended because the use of serial processing (that is, multiple calls to DFSORT) instead of OUTFIL processing can degrade performance and imposes certain restrictions as detailed later in this section. SERIAL is ignored if a single outdd data set is specified.

DFSORT is called to merge the indd data set to the first outdd data set using the DFSORT control statements in the xxxxCNTL data set. If the merge operation is successful, DFSORT is called as many times as necessary to copy the first outdd data set to the second and subsequent outdd data sets. Therefore, for maximum efficiency, use a disk data set as the first in a list of outdd data sets on both disk and tape. If more than one outdd data set is specified, DFSORT must be able to read the first outdd data set after it is written in order to copy it to the other outdd data sets. Do not use a SYSOUT or DUMMY data set as the first in a list of outdd data sets because:

- If the first data set is SYSOUT, DFSORT abends when it tries to copy the SYSOUT data set to the second outdd data set.
- If the first data set is DUMMY, DFSORT copies the empty DUMMY data set to the other outdd data sets (that is, all of the resulting outdd data sets are empty).

COLLKEY(UCA600)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547

COLLKEY(UCA410)
See the discussion of this operand on the COPY statement in “COPY operator” on page 547

COLLKEY(UCA400R1)
See the discussion of this operand on the COPY statement in “COPY operator” on page 547

COLLKEY(UCAnnn_L_R_V)
See the discussion of this operand on the COPY statement in “COPY operator” on page 547

MERGE examples

Although the MERGE operators in the examples in this section could all be contained in a single ICETOOL job step, they are shown and discussed separately for clarity.

Example 1

```
//TOOLIN DD *
MERGE FROM(IN01,IN02,IN03,IN04,IN05) TO(OUTPUT) USING(MERG)
//MERGCNTL DD *
  OPTION EQUALS
  MERGE FIELDS=(21,4,CH,A)
/*
```

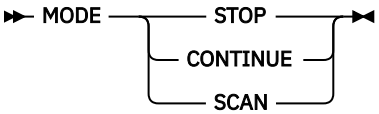
This example merges 5 input files to an output file. EQUALS is used to ensure that records that collate identically are output in the order specified in the FROM operand. For example, if IN01, IN03 and IN05 all have records with a key or 'AAAA' in positions 21-24, the output will contain the 'AAAA' record from IN01, the 'AAAA' record from IN03 and the 'AAAA' record from IN05, in that order.

Example 2

```
//TOOLIN DD *
MERGE FROM(INPUT1,INPUT2,INPUT3,INPUT4) -
      FROM(INPUT5,INPUT6,INPUT7) VSAMTYPE(F) USING(MRG1)
//MRG1CNTL DD *
  MERGE FIELDS=(52,8,UFF,D)
  OUTFIL FNAMES=OUT1,INCLUDE=(15,3,SS,EQ,C'D21,D33')
  OUTFIL FNAMES=OUT2,SAVE
/*
```

This example merges 7 input files to 2 output files. It uses two OUTFIL statements to create the two output files; each output file will have a different subset of the merged records. VSAMTYPE(F) tells DFSORT the record type is F (only needed for VSAM input files).

MODE operator



Specifies one of three modes to control error checking and actions after error detection. A MODE operator effects the "processing" (that is, error checking of ICETOOL statements and calling DFSORT) of the operators that follow it, up to the next MODE operator (if any). MODE operators allow you to do the following for groups of operators or all operators:

1. Stop or continue processing operators after a return code of 8, 12 or 16. A return code of 12 or 16 can be set as the result of a statement or run-time error detected by ICETOOL or DFSORT. A return code of 8 can be set as the result of a COUNT operator with RC8.
2. Check for errors in ICETOOL statements, but do not call DFSORT.

Operand descriptions

The operands described in this section can be specified in any order.

STOP
Stops subsequent operations if a return code of 8, 12 or 16 is set. If an error is detected for an operator, SCAN mode is automatically set in effect; DFSORT is not called for subsequent operators, although checking ICETOOL statements for errors continues.

STOP mode can be used to group dependent operators (that is, if an operation fails, do not process the remaining operators).

STOP MODE is set in effect automatically at the start of the ICETOOL run.

CONTINUE
Continues with subsequent operations regardless of whether or not a return code of 8, 12 or 16 is set. If an operator results in an error, processing continues for subsequent operators.

CONTINUE mode can be used to group independent operators (that is, process each operator regardless of the success or failure of the others).

SCAN
ICETOOL statements are checked for errors, but DFSORT is not called.

SCAN mode can be used to test ICETOOL statements for errors.

Note: SCAN mode is set automatically if an error is detected while in STOP mode.

MODE example

```
MODE SCAN
  RANGE ...
  UNIQUE ...
MODE STOP
  VERIFY ...
  DISPLAY ...
MODE CONTINUE
  COPY ...
  SORT ...
  STATS ...
```

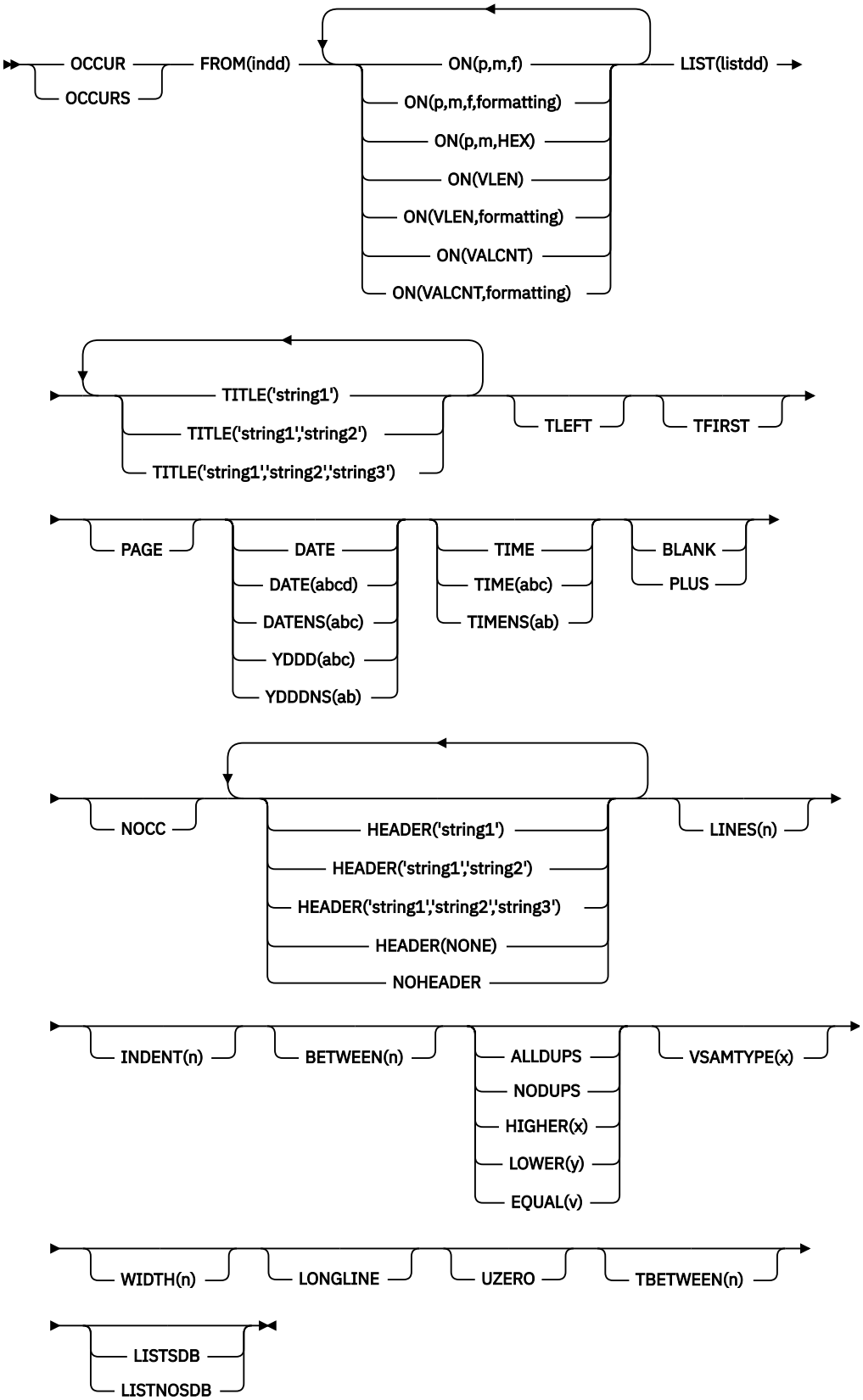
SCAN mode: RANGE and UNIQUE are checked for statement errors, but DFSORT is not called.

STOP mode: DISPLAY is dependent on VERIFY. If the return code for VERIFY is 12 or 16, SCAN mode is entered; DISPLAY is checked for statement errors, but DFSORT is not called.

CONTINUE mode: COPY, SORT, and STATS are independent of each other. SORT is processed even if the return code for COPY is 12 or 16. STATS is processed even if the return code for COPY or SORT is 12 or 16.

Note that the return codes for one group of operators does not affect the other groups of operators.

OCCUR operator



Prints each unique value for specified numeric fields (including SMF, TOD, and ETOD date and time) or character fields, and how many times it occurs, in a separate list data set. Simple or tailored reports

can be produced. The values printed can be limited to those for which the value count meets specified criteria.

From 1 to 10 fields can be specified, but the resulting list data set line length must not exceed the limit specified by the WIDTH operand or 8192 bytes if LONGLINE is specified and WIDTH is not specified, or 2048 bytes if LONGLINE and WIDTH are not specified. At least one ON(VLEN) or ON(p,m,f) field must be specified; all such ON fields specified are used to determine whether a record contains a unique value. A single list data set record is printed for each unique value. If ON(VALCNT) is specified, the "value count" (that is, the number of times the ON values occur) is printed in the list data set record along with the other ON values.

Specifying formatting items or the PLUS or BLANK operand, which can "compress" the columns of output data, can enable you to include more fields in your report, up to a maximum of 10, if your line length is limited by the character width your printer or display supports.

ALLDUPS, NODUPS, HIGHER(x), LOWER(y) or EQUAL(v) can be specified to limit the ON values printed to those for which the value count meets the specified criteria (for example, ALLDUPS for duplicate values only). The default criteria is HIGHER(0) resulting in the ON values being printed for each unique value.

DFSORT is called to sort the indd data set to ICETOOL's E35 user exit. ICETOOL uses its E35 exit to print appropriate titles, headings and data in the list data set.

You must not supply your own DFSORT MODS, INREC, OUTREC, SUM, or RECORD statement, because they override the DFSORT statements passed by ICETOOL for this operator.

The DYNALLOC option is passed to DFSORT to ensure that work space is available for the sort. If your installation defaults for dynamic allocation are inappropriate for an OCCUR operator, you can take one of the following actions:

1. Override the DYNALLOC option using an OPTION control statement such as:

   ```
   OPTION DYNALLOC=(,8)
   ```

   in the DFSPARM data set.
2. Use SORTWKdd DD statements to override the use of dynamic allocation. Refer to “SORTWKdd DD statement” on page 69 for details.

Attention: Either of these actions affects the work data sets used for a UNIQUE operator, or for a SELECT or SPLICE operator for which USING(xxxx) is not specified.

Tape work data sets **cannot** be used with ICETOOL.

Simple report

You can produce a simple report by specifying just the required operands. For example, if you specify FROM and LIST operands, and ON operands for 10-byte character and 7-byte zoned decimal fields and the value count, the output in the list data set can be represented as follows:

```
(p,m,f)               (p,m,f)            VALUE COUNT
characters            sddddddddddddddd   ddddddddddddddd
.                                  .                 .
.                                  .                 .
.                                  .                 .
```

A control character occupies the first byte of each list data set record. Left-justified standard headings are printed at the top of each page to indicate the contents of each column, followed by a line for each record showing the characters and numbers in the fields of that record, and the count of occurrences (value count) of the specified values.

The fields are printed in columns in the same order in which they are specified in the OCCUR statement. All fields are left-justified. For numeric fields, leading zeros are printed, a - is used for the minus sign, and a + is used for the plus sign. For the value count, leading zeros are printed.

By default, the first column of data starts immediately after the control character, and three blanks appear between columns. The NOCC operand can be used to suppress the control character. The INDENT operand can be used to change the number of blanks before the first column of data. The BETWEEN operand can be used to change the number of blanks between columns.

The standard column widths are as follows:

- Character data: the length of the character field or 20 bytes if the field length is less than 21 bytes
- Numeric data: 16 bytes, or 32 bytes if the numeric field is BI or FI with a length greater than 4, PD with a length greater than 8, or ZD, CSF, FS, UFF or SFF with a length greater than 15.
- Value count: 15 bytes

HEADER operands can be used to change or suppress the headings. PLUS or BLANK operands can be used to change the format of numeric fields. PLUS, BLANK and HEADER operands can be used to change the width of the columns for numeric and character fields and the justification of headings and fields.

The NOHEADER operand can be used to create list data sets containing only data records. Data sets created in this way can be processed further by other operators (for example, STATS or UNIQUE) using CH format for character values or FS format for numeric values (including the value count).

Tailored report

You can tailor the output in the list data set using various operands that control title strings, date, time, page number, headings, lines per page and field formats. The optional operands can be used in many different combinations to produce a wide variety of report formats. For example, if you specify FROM, LIST, BLANK, TITLE, PAGE, DATE, TIME, and HEADER operands, and ON operands for 10-byte character and 7-byte zoned decimal fields and the value count, the output in the list data set looks as follows:

```
title        - p -        mm/dd/yy        hh:mm:ss

header         header           header
----------   --------   --------------
characters         sd                d
.                   .                .
.                   .                .
.                   .                .
```

By default, a control character occupies the first byte of each list data set record. The NOCC operand can be used to suppress the control characters. The title lines (up to 3) are printed at the top of each page of the list data set. The first title line contains the elements you specify (title strings,page number, date and time) in the order in which you specify them. The second and third title lines contain the title strings you specify. By default, eight blanks appear between title elements, the title strings are centered with respect to each other, and the title lines appear on every page. The TBETWEEN(n) operand can be used to change the number of blanks between title elements. The TLEFT operand can be used to left-justify the title strings with respect to each other. The TFIRST operand can be used to only print the title lines on the first page. A blank line is printed after each title line.

Your specified headings (underlined) are printed after the title line on each page to indicate the contents of each column, followed by a line for each record showing the characters and numbers in the fields of that record. Your specified headings can be one, two or three lines. Headings for character fields are left-justified and headings for numeric fields are right-justified.

The fields are printed in columns in the same order in which they are specified in the OCCUR statement. Character fields are left-justified and numeric fields are right justified. For numeric fields, leading zeros are suppressed, a - is used for the minus sign, and a blank is used for the plus sign (you can specify PLUS rather than BLANK if you want a + to be used for the plus sign). For the value count, leading zeros are suppressed.

Formatting items can be used to change the appearance of individual numeric fields in the report with respect to separators, number of digits, decimal point, decimal places, signs, leading zeros, leading strings, floating strings, and trailing strings. Formatting items can also be used to insert leading or trailing strings for character fields.

The column widths are dynamically adjusted according to the length of the headings and the maximum number of bytes needed for the character or numeric data.

Operand descriptions

The operands described in this section can be specified in any order.

FROM(indd)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

ON(p,m,f)
Specifies the position, length, and format of a numeric or character field to be used for this operation. '(p,m,f)' is used for the standard column heading (see HEADER('string1'), HEADER('string1','string2'), HEADER('string1','string2','string3'), HEADER(NONE), and NOHEADER for alternative heading options).

By default, three blanks appear between columns. You can change the space between columns with BETWEEN(n).

p specifies the first byte of the field relative to the beginning of the input record. p is 1 for the first **data** byte of a fixed-length record and 5 for the first **data** byte of a variable-length record as illustrated in the following (RRRR represents the 4-byte record descriptor word):

```
    Fixed-length record    |    Variable-length record
    | D | A | T | A | ...  |    | R | R | R | R | D | A | T | A | ...
 p=   1   2   3   4        | p=   1   2   3   4   5   6   7   8
```

m specifies the length of the field in bytes. A field must not extend beyond position 32752 or beyond the end of a record. The maximum length for a field depends on its format.

f specifies the format of the field as follows:

Field formats of the ICETOOL RANGE operator Format Code	Length	Description
BI	1 to 8 bytes	Unsigned binary
FI	1 to 8 bytes	Signed fixed-point
PD	1 to 16 bytes	Signed packed decimal
ZD	1 to 31 bytes	Signed zoned decimal
CH	1 to 4000 bytes	Character
CSF or FS	1 to 32 bytes (31 digit limit)	Signed numeric with optional leading floating sign
UFF	1 to 44 bytes (31 digit limit)	Unsigned free form numeric
SFF	1 to 44 bytes (31 digit limit)	Signed free form numeric
DT1	4 bytes	SMF date interpreted as Z'yyyymmdd'
DT2	4 bytes	SMF date interpreted as Z'yyyymm'
DT3	4 bytes	SMF date interpreted as Z'yyyyddd'
DC1	8 bytes	TOD date interpreted as Z'yyyymmdd'
DC2	8 bytes	TOD date interpreted as Z'yyyymm'
DC3	8 bytes	TOD date interpreted as Z'yyyyddd'
DE1	8 bytes	ETOD date interpreted as Z'yyyymmdd'

Field formats of the ICETOOL RANGE operator Format Code	Length	Description
DE2	8 bytes	ETOD date interpreted as Z'yyyymm'
DE3	8 bytes	ETOD date interpreted as Z'yyyyddd'
TM1	4 bytes	SMF time interpreted as Z'hhmmss'
TM2	4 bytes	SMF time interpreted as Z'hhmm'
TM3	4 bytes	SMF time interpreted as Z'hh'
TM4	4 bytes	SMF time interpreted as Z'hhmmssxx'
TC1	8 bytes	TOD time interpreted as Z'hhmmss'
TC2	8 bytes	TOD time interpreted as Z'hhmm'
TC3	8 bytes	TOD time interpreted as Z'hh'
TC4	8 bytes	TOD time interpreted as Z'hhmmssxx'
TE1	8 bytes	ETOD time interpreted as Z'hhmmss'
TE2	8 bytes	ETOD time interpreted as Z'hhmm'
TE3	8 bytes	ETOD time interpreted as Z'hh'
TE4	8 bytes	ETOD time interpreted as Z'hhmmssxx'
Note: See Appendix C, "Data format descriptions," on page 823 for detailed format descriptions.		

For a CSF, FS, UFF, or SFF format field:

- A maximum of 31 digits is allowed. If a value with more than 31 digits is found, ICETOOL issues an error message and terminates the operation.

For a ZD or PD format field:

- If a decimal value contains an invalid digit (A-F), ICETOOL identifies the bad value in a message and terminates the operation.
- F, E, C, A, 8, 6, 4, 2, and 0 are treated as equivalent positive signs. Thus the zoned decimal values F2F3C1, F2F3F1 and 020301 are counted as only one unique value.
- D, B, 9, 7, 5, 3, and 1 are treated as equivalent negative signs. Thus the zoned decimal values F2F3B0, F2F3D0, and 020310 are counted as only one unique value.

The fields of records that do not meet the specified criteria are not checked for invalid digits (PD and ZD) or excessive digits (CSF, FS, UFF, and SFF).

For a DT1, DT2 or DT3 format field:

- An invalid SMF date can result in a data exception (0C7 ABEND) or an incorrect ZD date.
- SMF date values are always treated as positive.

For a DC1, DC2, DC3, DE1, DE2, or DE3 format field:

- TOD and ETOD date values are always treated as positive.

For a TM1, TM2, TM3 or TM4 format field:

- An invalid SMF time can result in an incorrect ZD time.
- SMF time values are always treated as positive.

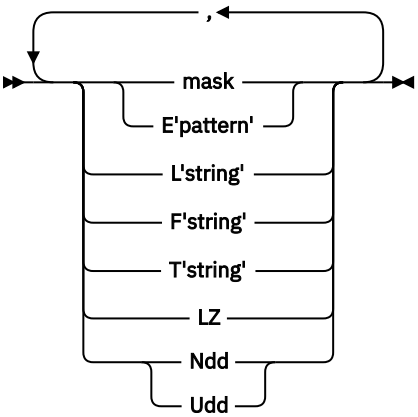
For a TC1, TC2, TC3, TC4, TE1, TE2, TE3, or TE4 format field:

- TOD and ETOD time values are always treated as positive.

ON(p,m,f,formatting)
Specifies the position, length and format of a numeric or character field to be used for this operation and how the data for this field is to be formatted for printing. The BLANK operand is automatically in effect.

See ON(p,m,f) for further details.

formatting



specifies formatting items that indicate how the data for this field is to be formatted for printing. Formatting items can be specified in any order, but each item can only be specified once. Any formatting item can be specified for a numeric field, but only L'string' and T'string' can be specified for a character field.

The column width is dynamically adjusted to accommodate the maximum bytes to be inserted as a result of all formatting items specified.

mask
See the discussion of **mask** under ON(p,m,f,formatting) in “DISPLAY operator” on page 566.

E'pattern'
See the discussion of **E'pattern'** under ON(p,m,f,formatting) in “DISPLAY operator” on page 566.

L'string'
See the discussion of **L'string'** under ON(p,m,f,formatting) in “DISPLAY operator” on page 566.

F'string'
See the discussion of **F'string'** under ON(p,m,f,formatting) in “DISPLAY operator” on page 566.

T'string'
See the discussion of **T'string'** under ON(p,m,f,formatting) in “DISPLAY operator” on page 566.

LZ
See the discussion of **LZ** under ON(p,m,f,formatting) in “DISPLAY operator” on page 566.

Ndd or Udd
specifies the number of digits to be used for the numeric field. Ndd or Udd can be used to change the column width for numeric fields. dd specifies the number of digits and must be a **two-digit** number between 01 and 31.

The default number of digits (d) for a numeric field is the maximum number of digits for that field. For example, d is 5 for ON(1,5,ZD). If you know that your numeric field requires less than d digits, you can use a lower number of digits (dd) instead by specifying Udd, thus reducing the column width if it is determined by d. For example, ON(1,5,ZD,U03) reduces d from 5 to 3. If you want your numeric field to be displayed with more than d digits, you can use a higher number of digits

(dd) instead by specifying Ndd or Udd, thus increasing the column width if it is determined by d. For example, ON(1,5,ZD,U10) increases d from 5 to 10.

Either Ndd or Udd can be used to set d greater than the maximum for a numeric field, but only Udd can be used to set d less than the maximum for a numeric field.

`For Udd:`

dd is used for d. For example:

- If ON(1,5,ZD) is specified, 5 digits (default for 5,ZD) are used.
- If ON(1,5,ZD,U10) is specified, 10 digits (from U10) are used.
- If ON(1,5,ZD,U03) is specified, 3 digits (from U03) are used.
- If ON(1,16,FS) is specified, 16 digits (default for 16,FS) are used.
- If ON(1,16,FS,U16) is specified, 16 digits (from U16) are used.
- If ON(1,16,FS,U15) is specified, 15 digits (from U15) are used.

If you use Udd and a numeric value overflows dd digits, ICETOOL terminates the operation. You can prevent the overflow by specifying an appropriately higher dd value for Udd. For example, if ON(1,12,ZD,U09) results in overflow, you can use ON(1,12,ZD,U10) instead.

If E'pattern' is specified, Udd is ignored, because the number of digits is determined from the pattern.

`For Ndd:`

If dd is greater than or equal to d, dd is used. If dd is less than d, d is used. For example:

- If ON(1,5,ZD) is specified, 5 digits (default for 5,ZD) are used.
- If ON(1,5,ZD,N10) is specified, 10 digits (from N10) are used.
- If ON(1,5,ZD,N03) is specified, 5 digits (from 5,ZD) are used.

If E'pattern' is specified, Ndd is ignored, because d is determined from the pattern.

ON(p,m,HEX)
Specifies the position and length of a character field to be used for this operation and printed in hexadecimal format (00-FF for each byte). '(p,m,HEX)' is used for the standard column heading (see HEADER('string1'), HEADER('string1','string2'), HEADER('string1','string2','string3'), HEADER(NONE), and NOHEADER for alternative heading options).

See ON(p,m,f) for a discussion of **p**.

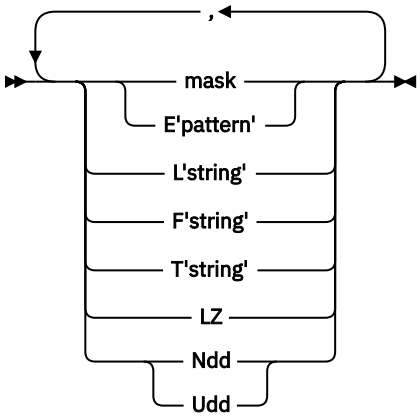
m specifies the length of the field in bytes. A field must not extend beyond position 32752 or beyond the end of a record. A field can be 1 to 2000 bytes.

ON(VLEN)
Equivalent to specifying ON(1,2,BI); a two-byte binary field starting at position 1. For variable-length records, ON(VLEN) represents the record-length for each record. 'RECORD LENGTH' is used for the standard column heading. See HEADER('string1'), HEADER('string1','string2'), HEADER('string1','string2','string3'), HEADER(NONE), and NOHEADER for alternative heading options.

ON(VLEN,formatting)
Equivalent to specifying ON(1,2,BI,formatting); a two-byte binary field starting at position 1, and how the data for this field is to be formatted for printing. The BLANK operand is automatically in effect.

See ON(VLEN) for further details.

formatting



specifies formatting items that indicate how the data for this field is to be formatted for printing. Formatting items can be specified in any order, but each item can only be specified once.

The column width is dynamically adjusted to accommodate the maximum bytes to be inserted as a result of all formatting items specified.

mask
See the discussion of **mask** under ON(p,m,f,formatting) in "DISPLAY operator" on page 566.

E'pattern'
See the discussion of **E'pattern'** under ON(p,m,f,formatting) in "DISPLAY operator" on page 566.

L'string'
See the discussion of **L'string'** under ON(p,m,f,formatting) in "DISPLAY operator" on page 566.

F'string'
See the discussion of **F'string'** under ON(p,m,f,formatting) in "DISPLAY operator" on page 566.

T'string'
See the discussion of **T'string'** under ON(p,m,f,formatting) in "DISPLAY operator" on page 566.

LZ
See the discussion of **LZ** under ON(p,m,f,formatting) in "DISPLAY operator" on page 566.

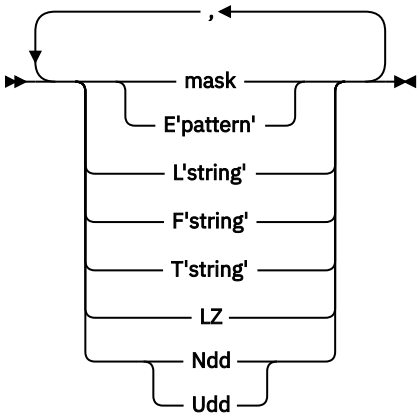
Ndd or Udd
See the discussion of Ndd or Udd under ON(p,m,f,formatting) in "OCCUR operator" on page 612.

ON(VALCNT)
Specifies that the number of occurrences for each unique value is to be printed. ' VALUE COUNT' is used for the standard column heading (see HEADER('string1'), HEADER('string1','string2'), HEADER('string1','string2','string3'), HEADER(NONE) and NOHEADER for alternative heading options).

ON(VALCNT,formatting)
Specifies that the number of occurrences for each unique value is to be printed, and how the value count is to be formatted for printing. The BLANK operand is automatically in effect.

See ON(VALCNT) for further details.

formatting



specifies formatting items that indicate how the value count is to be formatted for printing. Formatting items can be specified in any order, but each item can only be specified once.

The column width is dynamically adjusted to accommodate the maximum bytes to be inserted as a result of all formatting items specified.

mask
See the discussion of **mask** under ON(p,m,f,formatting) in "DISPLAY operator" on page 566.

E'pattern'
specifies an edit pattern to be applied to the value count. The pattern (1 to 24 characters) must be enclosed in single apostrophes. Each 9 in the pattern (up to 15) is replaced by a corresponding digit from the numeric value. Characters other than 9 in the pattern appear as specified. To include a single apostrophe (') in the pattern, specify two single apostrophes ('').

F'string' or a mask cannot be specified with E'pattern'.

When E'pattern' is specified for the value count:

- If the number of significant digits in a value count is less than the number of 9's in the pattern, 0's are filled in on the left. For example, 1234 is shown as 001234 with ON(VALCNT,E'999999').
- If the number of significant digits in a value count is greater than the number of 9's in the pattern, digits are truncated from the left. For example, 1234567 is shown as *4567* with ON(VALCNT,E'*9999*').

L'string'
See the discussion of **L'string'** under ON(p,m,f,formatting) in "DISPLAY operator" on page 566.

F'string'
See the discussion of **F'string'** under ON(p,m,f,formatting) in "DISPLAY operator" on page 566.

T'string'
See the discussion of **T'string'** under ON(p,m,f,formatting) in "DISPLAY operator" on page 566.

LZ
See the discussion of **LZ** under ON(p,m,f,formatting) in "DISPLAY operator" on page 566.

Ndd or Udd
Specifies the number of digits to be used for the value count when determining the column width. dd specifies the number of digits and must be a two-digit number between 01 and 15.

The default number of digits (d) for the value count is 15. If you know that your value counts require less than 15 digits, you can use a lower number of digits (dd) instead by specifying Ndd or Udd, thus reducing the column width if it is determined by d. For example, if ON(VALCNT,N06) or ON(VALCNT,U06) is specified, 6 digits (from N06 or U06) is used instead of 15 (default for value count).

If you use Ndd or Udd and a value count overflows the number of digits used, ICETOOL terminates the operation. You can prevent the overflow by specifying an appropriately higher dd value for

Ndd or Udd. For example, if ON(VALCNT,N05) results in overflow, you can use ON(VALCNT,N06) instead.

If E'pattern' is specified, Ndd or Udd is ignored, because d is determined from the pattern.

LIST(listdd)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

TITLE('string')
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

TITLE('string1','string2')
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

TITLE('string1','string2','string3')
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

TLEFT
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

TFIRST
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

PAGE
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

DATE
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

DATE(abcd)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

DATENS(abc)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

YDDD(abc)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

YDDDNS(ab)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

TIME
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

TIME(abc)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

TIMENS(ab)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

BLANK
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

PLUS
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

For ON(VALCNT), PLUS is treated as BLANK.

NOCC
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

HEADER('string1')

See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

HEADER('string1','string2')
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

HEADER('string1','string2','string3')
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

HEADER(NONE)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

NOHEADER

See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

LINES(n)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

INDENT(n)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

BETWEEN(n)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

TBETWEEN(n)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

ALLDUPS
Limits the ON values printed to those that occur more than once (that is, those with duplicate field values). The ON values are printed when value count > 1.

ALLDUPS is equivalent to HIGHER(1).

NODUPS
Limits the ON values printed to those that occur only once (that is, those with no duplicate field values). The ON values are printed when value count = 1.

NODUPS is equivalent to EQUAL(1) or LOWER(2).

HIGHER(x)
Limits the ON values printed to those that occur more than x times. The ON values are printed when value count > x.

x must be specified as n or +n where n can be 1 to 15 decimal digits.

LOWER(y)
Limits the ON values printed to those that occur less than y times. The ON values are printed when value count < y.

y must be specified as n or +n where n can be 1 to 15 decimal digits.

EQUAL(v)
Limits the ON values printed to those that occur v times. The ON values are printed when value count = v.

v must be specified as n or +n where n can be 1 to 15 decimal digits.

VSAMTYPE(x)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

WIDTH(n)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

LONGLINE
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

UZERO
Specifies that -0 and +0 are to be treated as unsigned zero values, that is, as the same value. With UZERO, -0 and +0 are treated as positive for ON processing.

UZERO overrides the default of treating -0 and +0 as signed zero values, that is, as different values. Without UZERO, -0 is treated as negative and +0 is treated as positive for ON processing.

LISTSDB or LISTNOSDB
See the discussion of these operands on the DISPLAY statement in "DISPLAY operator" on page 566.

OCCUR examples

Although the OCCUR operators in the examples in this section could all be contained in a single ICETOOL job step, they are shown and discussed separately for clarity. See “DISPLAY operator” on page 566 for additional examples of tailoring the report format.

Example 1

```
 OCCUR FROM(SOURCE) LIST(VOLSERS) ON(40,6,CH) ON(VALCNT)
```

Prints, in the VOLSERS data set:

- A heading line containing the standard headings
- A data line for each unique ON(40,6,CH) value in the standard format containing:
  - The characters from positions 40-45 of the SOURCE data set for the unique value
  - The count of occurrences in the SOURCE data set of the unique value

The VOLSERS output starts on a new page and looks as follows (the first 2 records are shown with illustrative values):

```
(40,6,CH)         VALUE COUNT
ABC001            000000000000025
ABC002            000000000000011
.                               .
.                               .
.                               .
```

The heading line appears at the top of each page.

Example 2

```
 OCCUR FROM(IN) LIST(LIST1) -
       TITLE('   3090 Distribution   ') -
       PAGE -
       HEADER('Data Centers')  ON(VALCNT) -
       HEADER('State') ON(1,16,CH) -
       HEADER('3090s') ON(25,3,PD) -
       BLANK
```

Prints, in the LIST1 data set:

- A title line containing the specified title and the page number
- A heading line containing the specified underlined headings
- A data line for each unique ON(1,16,CH) and ON(25,3,PD) value in the BLANK format containing:
  - The count of occurrences in the IN data set of the unique value
  - The characters from positions 1-16 of the IN data set for the unique value
  - The packed decimal values from positions 25-27 of the IN data set for the unique value

The LIST1 output starts on a new page and looks as follows (the first 2 records are shown with illustrative values):

```
   3090 Distribution          - 1 -

   Data Centers   State               3090s
---------------   ----------------   ------
             12   Alabama                 1
              6   Alabama                 2
              .   .                       .
              .   .                       .
              .   .                       .
```

The title line and underlined heading line appear at the top of each page.

Example 3

```
 OCCUR FROM(FAILURES) LIST(CHECKIT) BLANK HIGHER(4) -
   DATE TITLE('Possible System Intruders on ',System) -
        TITLE(Sysplex,' in ',Location) PAGE -
        TBETWEEN(2) -
        HEADER(,'Userid') ON(23,8,CH) -
        HEADER('Logon failures','(More than 4)') ON(VALCNT)
```

Assume that the SYMNAMES data set contains the following symbols:

```
System,S'&SYSNAME'
Sysplex,S'&SYSPLEX'
Location,'San Jose'
```

Prints, in the CHECKIT data set:

- A first title line containing the date, the string and System name specified in the first TITLE operand, and the page number. Two blanks appear between the title elements in the first title line as specified by the TBETWEEN(2) operand. A blank line follows the first title line.
- A second title line containing the Sysplex name, string and Location specified in the second TITLE operand. A blank line follows the second title line.
- An underlined three-line heading. The three-line heading for the first ON field has blanks on line1 and line2 and 'Userid' on line3. The heading for the second ON field has 'Number of' on line1, 'Logon Failures' on line2 and '(More than 4)' on line3.
- A data line for each unique ON(23,8,CH) value for which there are more than 4 occurrences, in the BLANK format, containing:
  - The characters from positions 23-30 of the FAILURES data set
  - The count of occurrences of the characters from positions 23-30 of the FAILURES data set

The CHECKIT output starts on a new page and looks as follows (the first 2 records are shown with illustrative values):

```
07/22/08  Possible System Intruders on EDS3  - 1 -

                  MAS3 in San Jose

            Logon failures
Userid       (More than 4)
--------   ---------------
B7234510                 5
D9853267                11
 ...
```

The title lines and underlined three-line heading lines appear at the top of each page.

Example 4

```
 OCCUR FROM(VARIN) LIST(ONCE) -
       TITLE('Record lengths that occur only once') -
       TIME(12:) DATE(DMY.) -
       ON(VLEN) NODUPS BLANK
```

Prints, in the ONCE data set:

- A title line containing the specified title and the time and date
- A heading line containing the standard heading
- A data line for each record length for which there is only one occurrence, in the BLANK format, containing the record length

The ONCE output starts on a new page and looks as follows (the first 2 records are shown with illustrative values):

```
Record lengths that occur only once        09:52:17 am        21.10.92

RECORD LENGTH
           57
           61
            .
            .
            .
```

The title line and heading line appear at the top of each page.

Example 5

```
OCCUR FROM(BRANCH) LIST(CALLRPT)-
      DATENS(4MD)-
      TITLE('Yearly Branch Phone Call Counts')-
      HEADER('Phone Number') ON(7,10,ZD,E'(999)-999-9999')-
      HEADER('Calls') ON(VALCNT,A1,N05)-
      INDENT(5) BETWEEN(10)
```

Prints, in the CALLRPT data set:

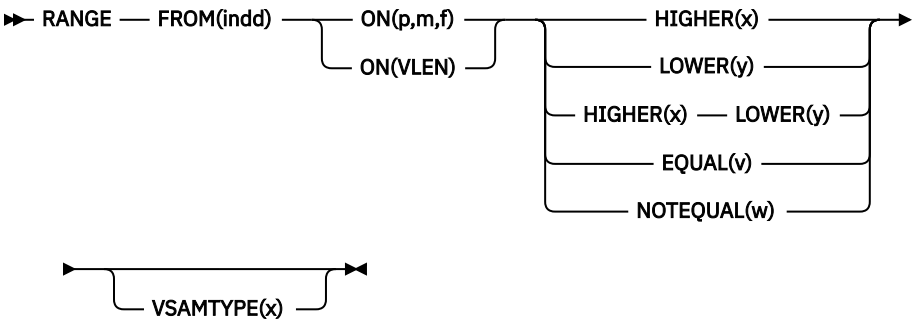
- A title line containing the date (without separators) and the specified title.
- A heading line containing the specified underlined headings.
- A data line for each unique ON(7,10,ZD) value containing:
  - The zoned decimal value from positions 7-16 of the BRANCH data set printed as (ddd)-ddd-dddd according to the E'pattern' formatting item.
  - The count of occurences of this value printed as dd,ddd according to the A1 and N05 formatting items.

The report is indented by five blanks as specified by the INDENT(5) operand, and ten blanks appear between the columns as specified by the BETWEEN(10) operand.

The CALLRPT output starts on a new page and looks as follows:

```
     20020316       Yearly Branch Phone Call Counts

       Phone Number            Calls
     --------------          -------
     (037)-325-1807            3,125
     (216)-721-5530            2,087
     (218)-062-7214              872
```

RANGE operator



Prints a message containing the count of values in a specified range for a specific numeric field.

DFSORT is called to copy the indd data set to ICETOOL's E35 user exit. ICETOOL prints a message containing the range count as determined by its E35 user exit.

The range can be specified as higher than x, lower than y, higher than x and lower than y, equal to v, or not equal to w, where x, y, v, and w are signed or unsigned decimal values. If the range is specified as higher than x and lower than y, it must be a valid range (for example, higher than 5 and lower than 6 is not a valid range, because there is no integer value that satisfies the criteria).

You must not supply your own DFSORT MODS, INREC, or OUTREC statement, because they would override the DFSORT statements passed by ICETOOL for this operator.

Operand descriptions

The operands described in this section can be specified in any order.

FROM(indd)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

ON(p,m,f)
Specifies the position, length, and format of the numeric field to be used for this operation.

p specifies the first byte of the field relative to the beginning of the input record. p is 1 for the first **data** byte of a fixed-length record and 5 for the first **data** byte of a variable-length record as illustrated in the following (RRRR represents the 4-byte record descriptor word):

```
    Fixed-length record    |    Variable-length record
    | D | A | T | A | ...  |    | R | R | R | R | D | A | T | A | ...
 p=   1   2   3   4        | p=   1   2   3   4   5   6   7   8
```

m specifies the length of the field in bytes. A field must not extend beyond position 32752 or beyond the end of a record. The maximum length for a field depends on its format.

f specifies the format of the field as follows:

Field formats of the ICETOOL RANGE operator Format Code	Length	Description
BI	1 to 8 bytes	Unsigned binary
FI	1 to 8 bytes	Signed fixed-point
PD	1 to 16 bytes	Signed packed decimal
ZD	1 to 31 bytes	Signed zoned decimal
CSF or FS	1 to 32 bytes (31 digit limit)	Signed numeric with optional leading floating sign
UFF	1 to 44 bytes (31 digit limit)	Unsigned free form numeric
SFF	1 to 44 bytes (31 digit limit)	Signed free form numeric
Note: See Appendix C, "Data format descriptions," on page 823 for detailed format descriptions.		

For a CSF, FS, UFF, or SFF format field:

- A maximum of 31 digits is allowed. If a value with more than 31 digits is found, ICETOOL issues an error message and terminates the operation.

For a ZD or PD format field:

- If a decimal value contains an invalid digit (A-F), ICETOOL identifies the bad value in a message and terminates the operation.
- A value is treated as positive if its sign is F, E, C, A, 8, 6, 4, 2, or 0.
- A value is treated as negative if its sign is D, B, 9, 7, 5, 3, or 1.

For a ZD, PD, CSF, FS, or SFF format field, a negative zero value is treated as a positive zero value.

ON(VLEN)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

HIGHER(x)
Values higher than x are counted as contained in the range. If only HIGHER(x) is specified, the range count is incremented when x < value. If LOWER(y) is also specified, the range count is incremented when x < value < y.

x must be specified as n, +n, or -n where n can be 1 to 31 decimal digits.

LOWER(y)
Values lower than y are counted as contained in the range. If only LOWER(y) is specified, the range count is incremented when value < y. If HIGHER(x) is also specified, the range count is incremented when x < value < y.

y must be specified as n, +n, or -n where n can be 1 to 31 decimal digits.

EQUAL(v)
Values equal to v are counted as contained in the range. The range count is incremented when v = value.

v must be specified as n, +n, or -n where n can be 1 to 31 decimal digits.

NOTEQUAL(w)
Values not equal to w are counted as contained in the range. The range count is incremented when w ¬= value.

w must be specified as n, +n, or -n where n can be 1 to 31 decimal digits.

VSAMTYPE(x)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

RANGE example

```
RANGE FROM(DATA1) ON(VLEN) HIGHER(10)
RANGE FROM(DATA2) ON(31,18,ZD) LOWER(+123456789012345678)
RANGE FROM(DATA3) ON(29001,4,FI) -
       HIGHER(-10000) LOWER(27)
RANGE FROM(DATA2) ON(45,3,PD) EQUAL(-999)
RANGE FROM(DATA3) ON(40,1,BI) NOTEQUAL(199)
```

The first RANGE operator prints a message containing the count of binary values from positions 1-2 of the DATA1 data set that are higher than 10.

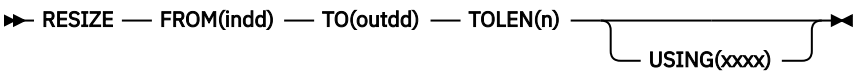
The second RANGE operator prints a message containing the count of zoned decimal values from positions 31-48 of the DATA2 data set that are lower than 123456789012345678.

The third RANGE operator prints a message containing the count of fixed-point values from positions 29 001-29 004 of the DATA3 data set that are higher than -10 000 but lower than 27.

The fourth RANGE operator prints a message containing the count of packed decimal values from positions 45-47 of the DATA2 data set that are equal to -999.

The fifth RANGE operator prints a message containing the count of binary values from position 40 of the DATA3 data set that are not equal to 199. This RANGE operator could be used to count the number of records that do not have 'G' in position 40, because 199 (X'C7') is the EBCDIC code for 'G'. Alternatively, the COUNT operator could be used with OMIT COND=(40,1,CH,EQ,C'G').

RESIZE operator



Produces fixed length output records, with the specified TOLEN length, from fixed length input records of a different length, as follows:

- If the input length is smaller than the requested TOLEN size, multiple smaller input records are combined into one larger output record of size TOLEN. If the combined input records do not completely fill up an output record, the output record will be padded on the right with blanks. For example, if we have five input records of 20 bytes and TOLEN is 70, two output records of 70 bytes will be created. The first output record will have input records 1-3 (60 bytes) followed by 10 blanks, and the second output record will have input records 4-5 (40 bytes) followed by 30 blanks. Note that the first 10 bytes of input record 4 are not used for the first output record; **bytes or words are not "wrapped" between records**.

  To illustrate, if the following RESIZE operator was specified:

  ```
  RESIZE FROM(FB20) TO(FB70) TOLEN(70)
  ```

  and FB20 had these 20 byte input records:

  ```
  <11111111111111111>
  <22222222222222222>
  <33333333333333333>
  <44444444444444444>
  <55555555555555555>
  ```

  FB70 would have these 70 byte output records:

  ```
  <11111111111111111><22222222222222222><33333333333333333>
  <44444444444444444><55555555555555555>
  ```

- If the input length is larger than the requested TOLEN size, each larger input record is broken up into multiple smaller output records of size TOLEN. If a broken up input record does not completely fill up the multiple output records, the last of the multiple output records will be padded on the right with blanks. For example, if we have two input records of 50 bytes and TOLEN is 20, six output records of 20 bytes each will be created. The first two output records will have bytes 1-20 and 21-40 of the first input record (40 bytes), respectively. The third output record will have bytes 41-50 of the first input record (10 bytes) followed by 10 blanks. Likewise, the fourth output record will have bytes 1-20 of the second input record, the fifth output record will have bytes 21-40 of the second input record, and the sixth output record will have bytes 41-50 of the second input record followed by 10 blanks.

  To illustrate, if the following RESIZE operator was specified:

  ```
  RESIZE FROM(F50) TO(F20) TOLEN(20)
  ```

  and F50 had these 50 byte input records:

  ```
  <11111111111111111><22222222222222222><33333333>
  <44444444444444444><55555555555555555><66666666>
  ```

  F20 would have these 20 byte output records:

  ```
  <11111111111111111>
  <22222222222222222>
  <33333333>
  <44444444444444444>
  <55555555555555555>
  <66666666>
  ```

You must specify the FROM(indd), TO(outdd) and TOLEN(n) operands. The FROM data set must be fixed-length (for example, RECFM=FB). The TO data set must also be fixed-length (unless you use an OUTFIL statement with the FTOV operand to change the fixed-length resized records to variable-length output records). If a VSAM input data set is used, it will be treated as TYPE=F (fixed-length) by default. If you specify a variable-length input data set (or use TYPE=V for a VSAM input data set), the RESIZE operation will be terminated.

The USING(xxxx) operand is optional; do not supply your own MODS or OUTREC statement.

DFSORT is called to copy or sort the indd data set, as appropriate, before the records are resized. ICETOOL uses its E35 exit to create a larger record from multiple smaller records, or to create multiple smaller records from a larger record. ICETOOL passes the EQUALS option to DFSORT to ensure that if records are sorted, duplicate records are kept in their original input order when resized.

If USING(xxxx) is specified, any SORT, INCLUDE, OMIT, INREC, or SUM statement specified in xxxxCNTL is processed **before** the records are resized. The order of the records, and the input length, will be affected by these control statements. Any OUTFIL statements specified in xxxxCNTL are processed **after** the records are resized.

The DYNALLOC option is passed to DFSORT to ensure that work space is available for the sort. If your installation defaults for dynamic allocation are inappropriate for a RESIZE operator, you can take one of the following actions:

1. Override the DYNALLOC option using an OPTION control statement such as:

   ```
   OPTION DYNALLOC=(,8)
   ```

   in the xxxxCNTL data set.
2. Use xxxxWKdd DD statements to override the use of dynamic allocation. Refer to "SORTWKdd DD statement" on page 69 for details.

Tape work data sets **cannot** be used with ICETOOL.

Operand descriptions

The operands described in this section can be specified in any order.

FROM(indd)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

TO(outdd)
See the discussion of this operand on the DATASORT statement in "DATASORT operator" on page 558.

TOLEN(n)
Specifies the record length you want ICETOOL to use for the resized output records. n can be 1 to 32760. n must not be equal to the input record length.

USING(xxxx)
Specifies the first 4 characters of the ddname for the control statement data set to be used by DFSORT for this operation. xxxx must be four characters that are valid in a ddname of the form xxxxCNTL. xxxx must not be SYSx. If USING(xxxx) is specified, an xxxxCNTL DD statement must be present, and the control statements in it must conform to the rules for DFSORT's SORTCNTL data set.

RESIZE examples

Although the RESIZE operators in the examples in this section could all be contained in a single ICETOOL job step, they are shown and discussed separately for clarity.

Example 1

```
RESIZE FROM(IN1) TO(OUT1) TOLEN(40)
```

This example illustrates how you can create larger records from smaller records.

The IN1 data set has RECFM=FB and LRECL=10 with these 10-byte records:

```
Bird
Bluejay
4
Charlie
Rodent
Rat
2
Sara
```

The OUT1 data set has RECFM=FB and LRECL=40 with these 40 byte records:

```
Bird      Bluejay   4         Charlie
Rodent    Rat       2         Sara
```

Example 2

```
RESIZE FROM(OLD) TO(NEW) TOLEN(15) USING(CTL1)
```

This example illustrates how you can create larger records from a subset of smaller sorted records.

The CTL1CNTL data set contains:

```
    OMIT COND=(2,4,ZD,EQ,0)
    SORT FIELDS=(1,1,CH,A)
```

The OLD data set has RECFM=FB and LRECL=5 with these 5-byte records:

```
C0005
B0000
A0008
I1234
F0053
D0123
H0001
G0000
E0022
```

The NEW data set will have RECFM=FB and LRECL=15 with these 15-byte records:

```
A0008C0005D0123
E0022F0053H0001
I1234
```

Note that before the records were resized, the two records with 0 in positions 2-5 were omitted, and the remaining records were sorted as directed by the DFSORT control statements in CTL1CNTL. The last output record was padded with blanks on the right to 15 bytes.

Example 3

```
RESIZE FROM(IN3) TO(OUT3) TOLEN(3) USING(CTL2)
```

This example illustrates how you can break up large records into multiple smaller records.

The CTL2CNTL data set contains:

```
  OUTFIL FNAMES=OUT3,OMIT=(1,3,CH,EQ,C' '),OVERLAY=(10:X)
```

The IN3 data set has RECFM=FB and LRECL=18 with these 18-byte records:

```
000111222333444555
666777888999
```

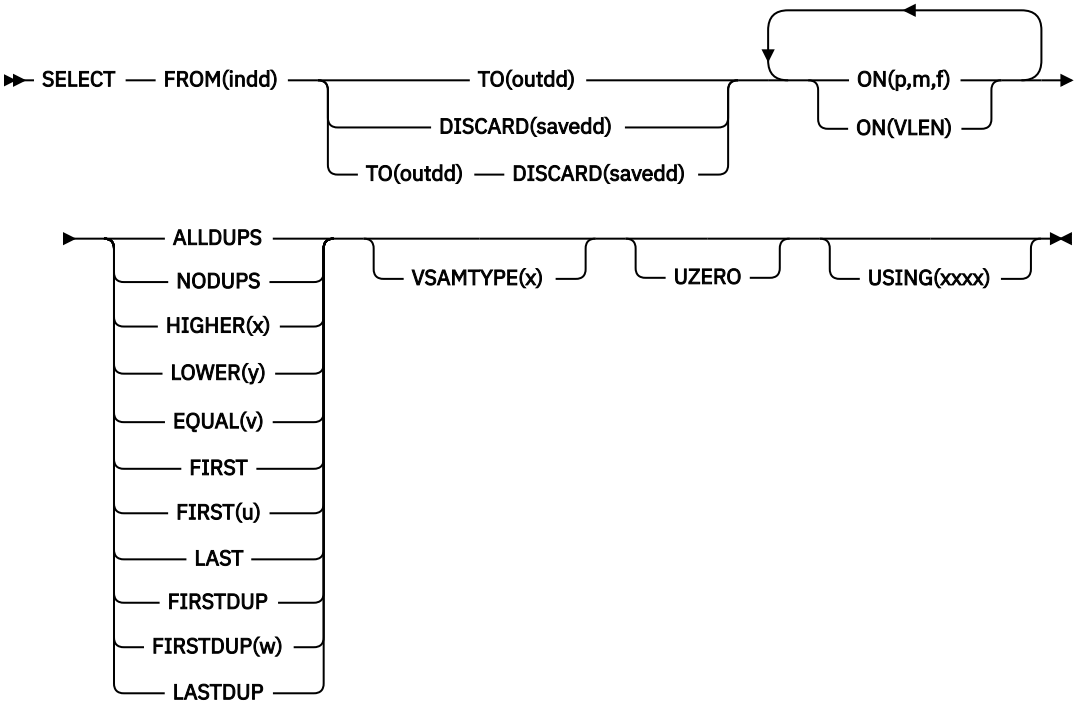
Every 3-byte field in each large IN3 record will be broken up into a single 3-byte field and then padded on the right with blanks to 10-bytes. TOLEN(3) indicates that the resized records will have a length of 3 bytes. OVERLAY=(10:X) expands each resized record to 10 bytes in OUT3. OMIT=(1,3,CH,EQ,C' ') removes any resized records that are completely blank (that is, the two blank resized records resulting from the blanks in positions 13-18 of the second input record).

The OUT3 data set will have RECFM=FB and LRECL=10 with these 10-byte records:

```
000
111
222
333
444
555
```

```
666
777
888
999
```

SELECT operator



Selects records from an input data set based on meeting criteria for the number of times specified numeric or character field values occur. This makes it possible to only keep records with duplicate field values, only keep records with no duplicate field values, only keep records with field values that occur more than, less than, or exactly n times, only keep the first or first n duplicate records with each field value, or only keep the first or last record with each unique or duplicate field value. From 1 to 10 fields can be specified. At least one ON(VLEN) or ON(p,m,f) field must be specified; all such ON fields specified will be used to determine the "value count" (that is, the number of times the ON values occur) to be matched against the criteria.

DISCARD(savedd) can be used to save the records that do not meet the criteria (that is, the discarded records), in the savedd data set. DISCARD(savedd) can be used with or without TO(outdd).

DFSORT is called to sort the indd data set. ICETOOL uses its E35 exit to determine which records to include in the outdd data set or savedd data set. ICETOOL passes the EQUALS option to DFSORT to ensure that duplicates are kept in their original input order.

The DFSORT control statements in xxxxCNTL are used if USING(xxxx) is specified.

Do not supply your own MODS, SUM or OUTREC statement.

You can use comment statements. You can use INCLUDE, OMIT, INREC, OPTION, SORT, or OUTFIL statements providing you observe these rules:

- You can use an INCLUDE or OMIT statement to remove input records **before** SELECT processing.
- You can use an INREC statement to reformat input records **before** SELECT processing. You can use INREC's PARSE, BUILD (FIELDS), OVERLAY, FINDREP, IFTHEN, or IFOUTLEN functions. If your INREC statement changes the starting position of an ON field, you must specify the new starting position for

that ON field. For example, if your input records have a CH key at positions 1-5 and you use an INREC statement like this:

```
INREC FIELDS=(25:1,50)
```

you must specify ON(25,5,CH) instead of ON(1,5,CH).

- If you specify a SORT statement, you must specify each ON field as a p,m,f,A sort field and these sort fields must be in the same order as the ON fields. After these sort fields, you can specify additional p,m,f,A or p,m,f,D sort fields. The additional sort fields will be used for sorting, but not for selecting. For example, if you use a SELECT statement like this:

  ```
  SELECT FROM(IN) TO(OUT) ON(21,5,CH) FIRST(3) USING(CTL1)
  ```

  you can use a SORT statement in CTL1CNTL like this:

  ```
  SORT FIELDS=(21,5,CH,A,41,6,CH,D)
  ```

  The records will be sorted by the 21,5,CH,A field and the 41,6,CH,D field, but only selected by the 21,5,CH field. This would allow you to select the three highest 41,6,CH values for each 21,5,CH value.
- If you specify TO(outdd) without DISCARD(savedd), you can further process the outdd records **after** SELECT processing using an OUTFIL statement like this:

  ```
  OUTFIL FNAMES=outdd,...
  ```

  or multiple OUTFIL statements like this:

  ```
  OUTFIL FNAMES=outdd,...
  OUTFIL FNAMES=outdd1,...
  ...
  ```

- If you specify DISCARD(savedd) without TO(outdd), you can further process the savedd records **after** SELECT processing using one (and only one) OUTFIL statement like this:

  ```
  OUTFIL FNAMES=savedd,...
  ```

- If you specify TO(outdd) and DISCARD(savedd), you can further process the outdd and savedd records **after** SELECT processing using two (and only two) OUTFIL statements like this:

  ```
  OUTFIL FNAMES=outdd,...
  OUTFIL FNAMES=savedd,...
  ```

  Both statements must be specified in the order shown with at least the FNAMES parameter. For example, to further modify only the DISCARD data set, you could use statements like this:

  ```
  OUTFIL FNAMES=OUT
  OUTFIL FNAMES=SAVE,INCLUDE=(21,3,ZD,GT,+25)
  ```

ICETOOL requires extra storage for SELECT processing, over and above what is normally needed by ICETOOL and DFSORT, in order to save your records until it can determine whether or not they meet your specified criteria. In most cases, only a small amount of storage is needed and can be obtained (above 16MB virtual). However, for a FROM data set with a large record length and criteria requiring many saved records, a large amount of storage is needed. For example, with a record length of 32756 and HIGHER(99), over 3 MBs of storage is needed. If ICETOOL cannot get the storage it needs, it issues a message and terminates the SELECT operation. Increasing the REGION by the amount indicated in the message may allow ICETOOL to run successfully.

The DYNALLOC option is passed to DFSORT to ensure that work space is available for the sort. If your installation defaults for dynamic allocation are inappropriate for a SELECT operator, you can specify USING(xxxx) and take one of the following actions:

1. Override the DYNALLOC option using an OPTION control statement such as:

   ```
   OPTION DYNALLOC=(,8)
   ```

in the xxxxCNTL data set.

2. Use xxxxWKdd DD statements to override the use of dynamic allocation. Refer to "SORTWKdd DD statement" on page 69 for details.

Tape work data sets **cannot** be used with ICETOOL.

Operand descriptions

The operands described in this section can be specified in any order.

FROM(indd)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

TO(outdd)
Specifies the ddname of the output data set to which DFSORT will write the records it selects for the operation (that is, the records that meet the specified criteria). Thus, the outdd data set will contain the records selected by ALLDUPS, NODUPS, HIGHER(x), LOWER(y), EQUAL(v), FIRST, FIRST(u), LAST, FIRSTDUP, FIRSTDUP(w) or LASTDUP.

An outdd DD statement must be present and must define an output data set that conforms to the rules for DFSORT's SORTOUT data set (if the DISCARD operand is not specified) or OUTFIL data set (if the DISCARD operand is specified).

TO and DISCARD can both be specified. If DISCARD is not specified, TO must be specified. If TO is not specified, DISCARD must be specified.

The ddname specified in the TO operand must not be the same as the ddname specified in the FROM or DISCARD operand.

Refer to "JCL restrictions" on page 546 for more information.

DISCARD(savedd)
Specifies the ddname of the output data set to which DFSORT will write the records it does not select for this operation (that is, the records that do not meet the specified criteria). Thus, the savedd data set will contain the records discarded by ALLDUPS, NODUPS, HIGHER(x), LOWER(y), EQUAL(v), FIRST, FIRST(u), LAST, FIRSTDUP, FIRSTDUP(w) or LASTDUP.

A savedd DD statement must be present and must define an output data set that conforms to the rules for DFSORT's OUTFIL data set.

TO and DISCARD can both be specified. If DISCARD is not specified, TO must be specified. If TO is not specified, DISCARD must be specified.

The ddname specified in the DISCARD operand must not be the same as the ddname specified in the FROM or TO operand.

Refer to "JCL restrictions" on page 546 for more information.

ON(p,m,f)
Specifies the position, length, and format of a numeric or character field to be used for this operation.

p specifies the first byte of the field relative to the beginning of the input record. p is 1 for the first **data** byte of a fixed-length record and 5 for the first **data** byte of a variable-length record as illustrated in the following (RRRR represents the 4-byte record descriptor word):

```
    Fixed-length record    |    Variable-length record
    | D | A | T | A | ...  |    | R | R | R | R | D | A | T | A | ...
 p=   1   2   3   4        | p=   1   2   3   4   5   6   7   8
```

If INREC is specified, p must refer to the record as reformatted by INREC.

m specifies the length of the field in bytes. A field must not extend beyond position 32752, or beyond the end of a record. If INREC is specified, a field must not extend beyond the end of the record as reformatted by INREC. The maximum length for a field depends on its format.

f specifies the format of the field as shown in the following table.

Format Code	Length	Description
BI	1 to 1500 bytes	Unsigned binary
FI	1 to 256 bytes	Signed fixed-point
PD	1 to 16 bytes	Signed packed decimal
ZD	1 to 31 bytes	Signed zoned decimal
CH	1 to 4000 bytes	Character
CSF or FS	1 to 32 bytes	Signed numeric with optional leading floating sign
UFF	1 to 44 bytes	Unsigned free form numeric
SFF	1 to 44 bytes	Signed free form numeric

Note: See Appendix C, "Data format descriptions," on page 823 for detailed format descriptions.

For a ZD or PD format field:

- F, E, C, A, 8, 6, 4, 2, and 0 are treated as equivalent positive signs. Thus the zoned decimal values F2F3C1, F2F3F1 and 020301 are counted as only one unique value.
- D, B, 9, 7, 5, 3, and 1 are treated as equivalent negative signs. Thus the zoned decimal values F2F3B0, F2F3D0, and 020310 are counted as only one unique value.
- Digits are not checked for validity.

ON(VLEN)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

ALLDUPS
Limits the records selected to those with ON values that occur more than once (value count > 1). You can use this operand to keep just those records with duplicate field values.

ALLDUPS is equivalent to HIGHER(1).

NODUPS
Limits the records selected to those with ON values that occur only once (value count = 1). You can use this operand to keep just those records with no duplicate field values.

NODUPS is equivalent to EQUAL(1) or LOWER(2).

HIGHER(x)
Limits the records selected to those with ON values that occur more than x times (value count > x). You can use this operand to keep just those records with field values that occur more than x times.

x must be specified as n or +n where n can be 0 to 99.

LOWER(y)
Limits the records selected to those with ON values that occur less than y times (value count < y). You can use this operand to keep just those records with field values that occur less than y times.

y must be specified as n or +n where n can be 0 to 99.

EQUAL(v)
Limits the records selected to those with ON values that occur v times (value count = v). You can use this operand to keep just those records with field values that occur v times.

v must be specified as n or +n where n can be 0 to 99.

FIRST
Limits the records selected to those with ON values that occur only once (value count = 1) and the first record of those with ON values that occur more than once (value count > 1). You can use this operand to keep just the first record for each unique field value.

FIRST is equivalent to FIRST(1).

FIRST(u)
Limits the records selected to those with ON values that occur only once (value count = 1) and the first u records of those with ON values that occur more than once (value count > 1). You can use this operand to keep just the first u records for each unique field value.

u must be specified as n or +n where n can be 1 to 999999999999999.

LAST
Limits the records selected to those with ON values that occur only once (value count = 1) and the last record of those with ON values that occur more than once (value count > 1). You can use this operand to keep just the last record for each unique field value.

FIRSTDUP
Limits the records selected to the first record of those with ON values that occur more than once (value count > 1). You can use this operand to keep just the first record of those records with duplicate field values.

FIRSTDUP is equivalent to FIRSTDUP(1).

FIRSTDUP(w)
Limits the records selected to the first w records of those with ON values that occur more than once (value count > 1). You can use this operand to keep just the first w records of those records with duplicate field values.

w must be specified as n or +n where n can be 1 to 999999999999999.

LASTDUP
Limits the records selected to the last record of those with ON values that occur more than once (value count > 1). You can use this operand to keep just the last record of those records with duplicate field values.

VSAMTYPE(x)
See the discussion of this operand on the COPY statement in “COPY operator” on page 547.

UZERO
See the discussion of this operand on the OCCUR statement in “OCCUR operator” on page 612.

USING(xxxx)
Specifies the first 4 characters of the ddname for the control statement data set to be used by DFSORT for this operation. xxxx must be four characters that are valid in a ddname of the form xxxxCNTL. xxxx must not be SYSx.

If USING is specified, an xxxxCNTL DD statement must be present and the control statements in it:

1. Must conform to the rules for DFSORT's SORTCNTL data set.
2. Should generally be used only for an INCLUDE or OMIT statement, an INREC statement, a SORT statement, comment statements, or OUTFIL statements as described for “SELECT operator” on page 631.

SELECT examples

Although the SELECT operators in the examples in this section could all be contained in a single ICETOOL job step, they are shown and discussed separately for clarity.

Example 1

```
SELECT FROM(INPUT) TO(DUPS) ON(11,8,CH) ON(30,44,CH) ALLDUPS
```

Sorts the INPUT data set to the DUPS data set, selecting only the records from INPUT with characters in positions 11-18 and characters in positions 30-73 that occur more than once (that is, only records with duplicate ON field values).

The DUPS data set might look as follows (several records are shown for illustrative purposes):

```
USR002    BETTEN       12    DOC.EXAMPLES
DFSRT2    BETTEN        5    DOC.EXAMPLES
DFSRT5    BOENIG       20    MYDATA
DFSRT1    BOENIG       20    MYDATA
SYS003    BOENIG       20    MYDATA
DFSRT2    BOENIG       20    SORTST1.TEST
USR003    BOENIG       20    SORTST1.TEST
.         .            .     .
.         .            .     .
.         .            .     .
```

Example 2

```
  SELECT FROM(INPUT) TO(ONLYONE) ON(23,3,FS) NODUPS
```

Sorts the INPUT data set to the ONLYONE data set, selecting only the records from INPUT with floating sign values in positions 23-25 that occur just once (that is, only records with no duplicate ON field values).

The ONLYONE data set might look as follows (several records are shown for illustrative purposes):

```
DFSRT2    BOENIG        5    DOC.EXAMPLES
DFSRT1    PACKER        8    ICETOOL.SMF.RUNS
USR002    BETTEN       12    DOC.EXAMPLES
SYS003    YAEGER       32    ICETOOL.TEST.CASES
DFSRT2    CORNELL     108    FS.TEST.CASES
.         .           .      .
.         .           .      .
.         .           .      .
```

Example 3

```
  SELECT FROM(FAILURES) TO(CHECKOUT) ON(28,8,CH) ON(1,5,CH) -
    HIGHER(3)
```

Sorts the FAILURES data set to the CHECKOUT data set, selecting only the records from FAILURES with characters in positions 28-35 and characters in positions 1-5 that occur more than three times (that is only records with four or more duplicate ON field values).

The CHECKOUT data set might look as follows (several records are shown for illustrative purposes):

```
03/12/04   08:36:59        A3275647
03/12/04   09:27:32        A3275647
03/12/04   09:03:18        A3275647
03/12/04   08:56:13        A3275647
03/06/04   15:12:01        C3275647
03/06/04   14:57:00        C3275647
03/06/04   15:43:19        C3275647
03/06/04   16:06:39        C3275647
03/06/04   15:22:08        C3275647
.          .               .
.          .               .
.          .               .
```

Example 4

```
  SELECT FROM(BOOKS) TO(PUBLISHR) ON(29,10,CH) FIRST
```

Sorts the BOOKS data set to the PUBLISHR data set, selecting only the records from BOOKS with characters in positions 29-38 that occur only once and the first record of those with characters in positions 29-38 that occur more than once (that is, one record for each unique ON field value).

The PUBLISHR data set might look as follows (several records are shown for illustrative purposes):

```
Banana Slugs I Have Known   Brent      Animals
Toads on Parade             Cooper     Animals
Pets Around the World       Davis      Animals
```

```
.                          .          .
.                          .          .
.                          .          .
```

Example 5

```
  SELECT FROM(BOOKS) TO(PUBLISHR) ON(29,10,CH) FIRST -
    DISCARD(SAVEREST)
```

This example creates the same PUBLISHR data set as Example 4. In addition, it creates a SAVEREST data set that contains all of the records not written to the PUBLISHR data set. The SAVEREST data set might look as follows (several records are shown for illustrative purposes):

```
How to Talk to Your Amoeba  Brent    Animals
What Buzzards Want          Davis    Animals
Birds of Costa Rica         Davis    Animals
.
.
.
```

Example 6

```
  SELECT FROM(MASTPULL) TO(MATCH) ON(5,8,CH) FIRSTDUP
```

This example shows how you can use a list of account numbers in a "pull" data set to only select records with those account numbers from a "master" data set. The MASTPULL DD would have the "master" data set and "pull" data set concatenated together (in that order).

The SELECT operator sorts the concatenated data sets and selects only the first record of those with characters in positions 5-12 that occur more than once (that is, one record for each duplicate ON field value). Because the "master" data set is first in the concatenation, the selected records will come from the "master" data set.

If the "master" data set looked like this:

```
A52 RB172832 2001/03/15
N92 MX328126 2001/01/27
B12 LB018725 2000/12/28
J73 AB007231 2001/02/13
Q28 SP973004 2000/11/19
```

and the "pull" data set looked like this:

```
   AB007231
   RS859276
   QN005001
   MX328126
```

the MATCH data set would look like this:

```
J73 AB007231 2001/02/13
N92 MX328126 2001/01/27
```

Note: This example assumes that there are not any duplicate account numbers in either the "master" or "pull" data sets. If that is not true, you can use SELECT with FIRST or LAST, for the appropriate data set, to make it true. For example, if your "master" data set has duplicate account numbers and you want to select the first account number from the "master" data set for each account number in the "pull" data set, you could use the following statements:

```
  SELECT FROM(MASTER) TO(TEMP) ON(5,8,CH) FIRST
  SELECT FROM(TEMPPULL) TO(MATCH) ON(5,8,CH) FIRSTDUP
```

The TEMPPULL DD would have the temporary data set and "pull" data set concatenated together (in that order).

Example 7

```
  SELECT FROM(INPUT) TO(ONLYONE) ON(23,3,FS) NODUPS USING(CTL1)
```

This example shows how you can use USING(xxxx) to supply an OUTFIL statement to modify the TO data set. SELECT chooses the same output records as for “Example 2” on page 636, but an OUTFIL statement is used to further modify those records for output to the ONLYONE data set.

The CTL1CNTL data set contains:

```
  OUTFIL FNAMES=ONLYONE,
    REMOVECC,
    INCLUDE=(23,3,FS,LT,100),
    OUTREC=(1:1,7,8:C'|',11:11,7,19:C'|',23:23,3,FS,M11,
            27:C'|',30:30,15),
    TRAILER1=(/,'TOTAL= ',TOT=(23,3,FS,M11,LENGTH=6))
```

and the ONLYONE data set might look as follows:

```
DFSRT2 |  EISSLER |   005 |  DOC.EXAMPLES
DFSRT1 |  PACKER  |   008 |  ICETOOL.SMF.RUN
USR002 |  EISSLER |   012 |  DOC.EXAMPLES
SYS003 |  YAEGER  |   032 |  ICETOOL.TEST.CA

TOTAL= 000057
```

Example 8

This example shows how you can use USING(xxxx) to supply an OMIT statement to remove certain input records and an INREC statement to reformat certain input records before SELECT processing begins.

```
 SELECT FROM(INB) TO(OUTB) ON(8,10,CH) -
   FIRSTDUP USING(BIRD)
```

Let's say the INB data set looks like this:

```
TYPE1  CROWS
TYPE1  PIGEONS
TYPE1  HAWKS
TYPE1  ROBINS
TYPE1  SPARROWS
TYPE1  CHICKENS
TYPE1  RAVENS
TYPE2            PIGEONS
TYPE2            ROBINS
TYPE2            HAWKS
TYPE2            TERNS
TYPE3  EAGLES
TYPE3  STARLINGS
```

We want to remove the TYPE3 records and then select one record for each type of bird that appears in both a TYPE1 and a TYPE2 record. We could use the following control statements in BIRDCNTL:

```
* Remove the TYPE3 records.
  OMIT COND=(1,5,CH,EQ,C'TYPE3')
* Reformat the TYPE2 records to place the bird name in
* same place as in the TYPE1 records so we can match them.
  INREC IFTHEN=(WHEN=(1,5,CH,EQ,C'TYPE2'),
         BUILD=(8:18,10))
```

The OUTB data set would look like this:

```
TYPE1  HAWKS
TYPE1  PIGEONS
TYPE1  ROBINS
```

Example 9

This example shows how you can use USING(xxxx) to supply a SORT statement to alter the records that are selected.

```
  SELECT FROM(IN) TO(OUT) ON(1,5,CH) FIRST(3) USING(CTL1)
```

Let's say the IN data set looks like this:

```
FRANK 00015
FRANK 00012
FRANK 00003
FRANK 00018
FRANK 00005
FRANK 00035
VICKY 00022
VICKY 00007
VICKY 00014
VICKY 00028
VICKY 00002
VICKY 00015
```

We want to select the three records with each name that have the highest counts. If we just used ON(1,5,CH) without any CTL1CNTL statements, we'd get the first three records for each name without regard to the count. The OUT data set would look like this:

```
FRANK 00015
FRANK 00012
FRANK 00003
VICKY 00022
VICKY 00007
VICKY 00014
```

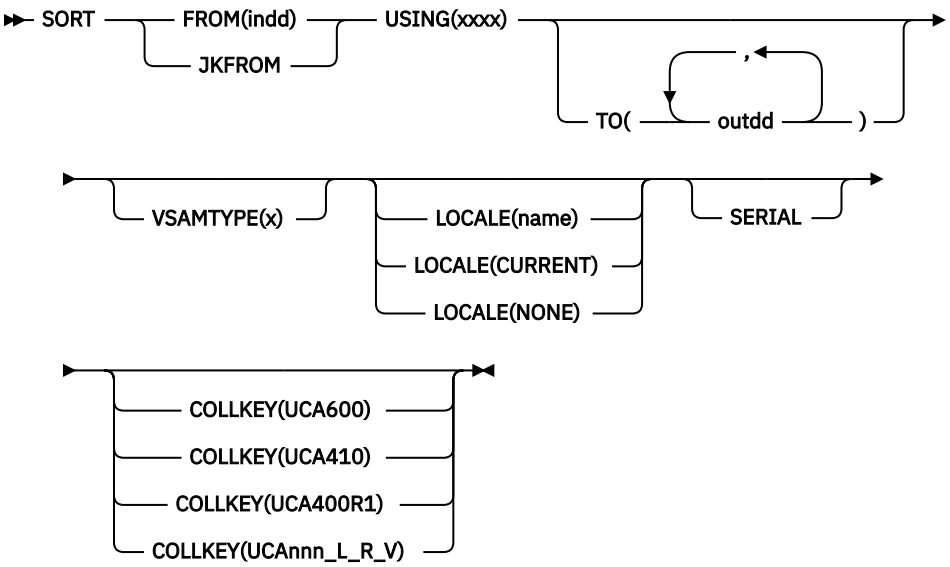
To get the three records with the highest counts for each name, we can use the following SORT statement in CTL1CNTL:

```
  SORT FIELDS=(1,5,CH,A,7,5,ZD,D)
```

The records will be sorted in ascending order on the name field, and in descending order on the count field. By sorting descending on the count, we ensure that the three records with the highest counts are the first three records for each name. Thus, when ON(1,5,CH) selects the first three records, they will be those with the highest counts. The OUT data set will look like this:

```
FRANK 00035
FRANK 00018
FRANK 00015
VICKY 00028
VICKY 00022
VICKY 00015
```

SORT operator



Sorts a data set to one or more output data sets.

DFSORT is called to sort the indd data set to the outdd data sets using the DFSORT control statements in xxxxCNTL. You must supply a DFSORT SORT statement in the xxxxCNTL data set to indicate the control fields for the sort. You can use additional DFSORT statements in the xxxxCNTL data set to sort a subset of the input records (INCLUDE or OMIT statement; SKIPREC and STOPAFT options; OUTFIL INCLUDE, OMIT, SAVE, STARTREC, ENDREC, SAMPLE, SPLIT, SPLITBY, and SPLIT1R operands; user exit routines), reformat records for output (INREC, OUTREC, and OUTFIL statements, user exit routines), and so on.

The active locale's collating rules affect SORT processing as explained in “SORT control statement” on page 423. If an INCLUDE or OMIT statement or an OUTFIL INCLUDE or OMIT operand is specified in the xxxxCNTL data set, the active locale's collating rules affect INCLUDE and OMIT processing as explained in the "Cultural Environment Considerations" discussion in “INCLUDE control statement” on page 91.

The DYNALLOC option is passed to DFSORT to ensure that work space is available for the sort. If your installation defaults for dynamic allocation are inappropriate for a SORT operator, you can take one of the following actions:

1. Override the DYNALLOC option using an OPTION control statement such as:

   ```
   OPTION DYNALLOC=(,8)
   ```

   in the xxxxCNTL data set in conjunction with the USING(xxxx) operand.
2. Use xxxxWKdd DD statements to override the use of dynamic allocation in conjunction with the USING(xxxx) operand. Refer to “SORTWKdd DD statement” on page 69 for details.

Tape work data sets **cannot** be used with ICETOOL.

Operand descriptions

The operands described in this section can be specified in any order.

FROM(indd)
See the discussion of this operand on the COPY statement in “COPY operator” on page 547.

JKFROM
Specifies you are using a JOINKEYS application with this SORT operator to sort the joined records. You must provide a USING(xxxx) operand. In xxxxCNTL, you must provide a SORT statement, a JOINKEYS statement with F1=ddname1 for the F1 file, and a JOINKEYS statement with F2=ddname2 for the F2 file, as well as JOIN and REFORMAT statements as needed.

USING(xxxx)
Specifies the first 4 characters of the ddname for the control statement data set to be used by DFSORT for this operation. xxxx must be four characters that are valid in a ddname of the form xxxxCNTL. xxxx must not be SYSx.

An xxxxCNTL DD statement must be present, and the control statements in it must conform to the rules for DFSORT's SORTCNTL data set.

The xxxxCNTL data set must contain a SORT statement. If TO is not specified, the xxxxCNTL data set must also contain either one or more OUTFIL statements or a MODS statement for an E35 routine that disposes of all records. Other statements are optional.

If you want to override dynamic allocation of work data sets for this operation, you can use xxxxWKdd DD statements for that purpose.

Refer to "JCL restrictions" on page 546 for more information regarding the selection of ddnames.

TO(outdd,...)
Specifies the ddnames of the output data sets to be written by DFSORT for this operation. From 1 to 10 outdd names can be specified. An outdd DD statement must be present for each outdd name specified. If a single outdd data set is specified, DFSORT is called once to sort the indd data set to the outdd data set using SORTOUT processing; the outdd data set must conform to the rules for DFSORT's SORTOUT data set. If multiple outdd data sets are specified and SERIAL is not specified, DFSORT is called once to sort the indd data set to the outdd data sets using OUTFIL processing; the outdd data sets must conform to the rules for DFSORT's OUTFIL data sets.

A ddname specified in the FROM operand must not also be specified in the TO operand.

Refer to "JCL restrictions" on page 546 for more information regarding the selection of ddnames.

VSAMTYPE(x)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

LOCALE(name)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

LOCALE(CURRENT)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

LOCALE(NONE)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

SERIAL
Specifies that OUTFIL processing is not to be used when multiple outdd data sets are specified. DFSORT is called multiple times and uses SORTOUT processing; the outdd data sets must conform to the rules for DFSORT's SORTOUT data set. SERIAL is not recommended because the use of serial processing (that is, multiple calls to DFSORT) instead of OUTFIL processing can degrade performance and imposes certain restrictions as detailed later in this section. SERIAL is ignored if a single outdd data set is specified.

DFSORT is called to sort the indd data set to the first outdd data set using the DFSORT control statements in the xxxxCNTL data set. If the sort operation is successful, DFSORT is called as many times as necessary to copy the first outdd data set to the second and subsequent outdd data sets. Therefore, for maximum efficiency, use a disk data set as the first in a list of outdd data sets on both disk and tape. If more than one outdd data set is specified, DFSORT must be able to read the *first* outdd data set after it is written in order to copy it to the other outdd data sets. Do not use a SYSOUT or DUMMY data set as the first in a list of outdd data sets because:

- If the first data set is SYSOUT, DFSORT abends when it tries to copy the SYSOUT data set to the second outdd data set.
- If the first data set is DUMMY, DFSORT copies the empty DUMMY data set to the other outdd data sets (that is, all of the resulting outdd data sets are empty).

COLLKEY(UCA600)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547

COLLKEY(UCA410)
See the discussion of this operand on the COPY statement in “COPY operator” on page 547

COLLKEY(UCA400R1)
See the discussion of this operand on the COPY statement in “COPY operator” on page 547

COLLKEY(UCAnnn_L_R_V)
See the discussion of this operand on the COPY statement in “COPY operator” on page 547

Sort examples

This section includes 14 sort examples.

Example 1

```
* Method 1
SORT FROM(MASTER) TO(PRINT,TAPE,DISK) USING(ABCD)
```

```
* Method 2
SORT FROM(MASTER) TO(DISK,TAPE,PRINT) USING(ABCD) SERIAL
```

This example shows two different methods for creating multiple sorted output data sets. Assume that the ABCDCNTL data set contains:

```
  SORT FIELDS=(15,20,CH,A,1,5,PD,D)
```

Method 1 requires one call to DFSORT, one pass over the input data set, and allows the output data sets to be specified in any order. The SORT operator sorts all records from the MASTER data set to the PRINT (SYSOUT), TAPE, and DISK data sets, using the SORT statement in the ABCDCNTL data set and OUTFIL processing.

Method 2 requires three calls to DFSORT, three passes over the input data set, and imposes the restriction that the SYSOUT data set must not be the first TO data set. The SORT operator sorts all records from the MASTER data set to the DISK data set, using the SORT statement in the ABCDCNTL data set, and then copies the resulting DISK data set to the TAPE and PRINT (SYSOUT) data sets. Because the first TO data set is processed three times (written, read, read), placing the DISK data set first is more efficient than placing the TAPE data set first. PRINT must not be the first in the TO list because a SYSOUT data set cannot be read.

Example 2

```
* Method 1
SORT FROM(IN) TO(DEPT1) USING(DPT1)
SORT FROM(IN) TO(DEPT2) USING(DPT2)
SORT FROM(IN) TO(DEPT3) USING(DPT3)
```

```
* Method 2
SORT FROM(IN) USING(ALL3)
```

This example shows two different methods for creating sorted subsets of an input data set. Assume that:

- The DPT1CNTL data set contains:

  ```
  SORT FIELDS=(51,2,BI,A,18,5,CH,A,58,4,BI,A)
  INCLUDE COND=(5,3,CH,EQ,C'D01')
  ```

- The DPT2CNTL data set contains:

  ```
  SORT FIELDS=(51,2,BI,A,18,5,CH,A,58,4,BI,A)
  INCLUDE COND=(5,3,CH,EQ,C'D02')
  ```

- The DPT3CNTL data set contains:

```
SORT FIELDS=(51,2,BI,A,18,5,CH,A,58,4,BI,A)
INCLUDE COND=(5,3,CH,EQ,C'D03')
```

- The ALL3CNTL data set contains:

```
SORT FIELDS=(51,2,BI,A,18,5,CH,A,58,4,BI,A)
OUTFIL FNAMES=DEPT1,INCLUDE=(5,3,CH,EQ,C'D01')
OUTFIL FNAMES=DEPT2,INCLUDE=(5,3,CH,EQ,C'D02')
OUTFIL FNAMES=DEPT3,INCLUDE=(5,3,CH,EQ,C'D03')
```

Method 1 requires three calls to DFSORT and three passes over the input data set:

- The first SORT operator sorts the records from the IN data set that contain D01 in positions 5-7 to the DEPT1 data set
- The second COPY operator sorts the records from the IN data set that contain D02 in positions 5-7 to the DEPT2 data set
- The third COPY operator sorts the records from the IN data set that contain D03 in positions 5-7 to the DEPT3 data set.

Method 2 accomplishes the same result as method 1 but, because it uses OUTFIL statements instead of TO operands, requires only one call to DFSORT and one pass over the input data set.

Example 3

```
SORT FROM(IN1) TO(FRANCE) USING(SRT1) LOCALE(FR_FR)
SORT FROM(IN1) TO(CANADA) USING(SRT1) LOCALE(FR_CA)
SORT FROM(IN1) TO(BELGIUM) USING(SRT1) LOCALE(FR_BE)
```

This example shows how sorted data for three different countries can be produced. Assume that the SRT1CNTL data set contains:

```
  SORT FIELDS=(5,20,CH,A,31,15,CH,A,1,4,FI,D,63,10,CH,D)
```

The first SORT operator sorts all records from the IN1 data set to the FRANCE data set, using the SORT statement in the SRT1CNTL data set. The character (CH) control fields are sorted according to the collating rules defined in locale FR_FR (French language for France).

The second SORT operator sorts all records from the IN1 data set to the CANADA data set, using the SORT statement in the SRT1CNTL data set. The character (CH) control fields are sorted according to the collating rules defined in locale FR_CA (French language for Canada).

The third SORT operator sorts all records from the IN1 data set to the BELGIUM data set, using the SORT statement in the SRT1CNTL data set. The character (CH) control fields are sorted according to the collating rules defined in locale FR_BE (French language for Belgium).

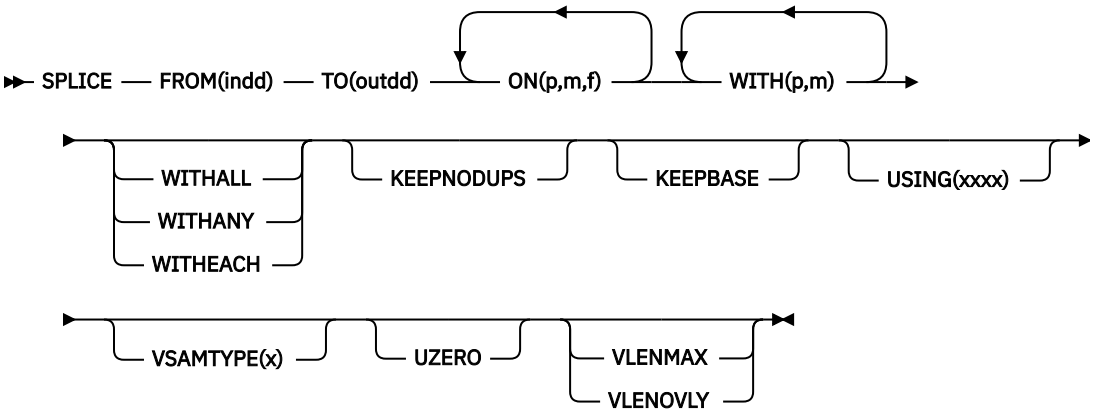
SORT operator with JOINKEYS example

Here is an example of using one SORT operator for a simple JOINKEYS application.

```
//SRTJK EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//JNA DD DSN=MY.INPUTA,DISP=SHR
//JNB DD DSN=MY.INPUTB,DISP=SHR
//OUT DD SYSOUT=*
//TOOLIN DD *
* SORT operator with JOINKEYS application.
SORT JKFROM TO(OUT) USING(CTL1)
/*
//CTL1CNTL DD *
* JOINKEYS application control statements for SORT operator.
  JOINKEYS F1=JNA,FIELDS=(5,4,A)
  JOINKEYS F2=JNB,FIELDS=(11,4,A),SORTED
  REFORMAT FIELDS=(F1:1,20,F2:5,15)
* Main task control statement for SORT operator
* (operates on joined records).
  OPTION EQUALS
```

```
  SORT FIELDS=(1,3,ZD,A)
/*
```

SPLICE operator



Splices together specified fields from records with matching numeric or character field values (that is, duplicate values), but different information. This makes it possible to join fields from different types of input records to create an output record with information from two or more records.

Typically, you will want to reformat the records from two or more data sets to a temporary MOD data set, and use that temporary MOD data set as input to the SPLICE operator.

"SPLICE examples" on page 652 shows some techniques for splicing records from different data sets together in a variety of ways to perform various file "join" and "match" operations.

By default (when WITHALL, WITHANY and WITHEACH are not specified), one spliced record is created for each set of duplicates by splicing the first duplicate with specified fields from the last duplicate.

The first duplicate is treated as a "base record". The last duplicate is treated as an "overlay record". Specified fields from the overlay record are overlaid on to the base record. Thus, the output record consists of fields from the base (first) record intermixed with specified fields from the overlay (last) record.

The records to be spliced can originate from two or more different input data sets.

From 1 to 10 ON fields can be used for the fields to match on. At least one ON(p,m,f) field must be specified; all such ON fields specified will be used to determine the matching records to be spliced together.

From 1 to 50 WITH fields can be used to specify the fields to be overlaid on the base record from the overlay record. At least one WITH(p,m) field must be specified; all such WITH fields specified will be overlaid on to the base record. All other fields in the base record will be kept unchanged.

To illustrate the splicing process, if we had the following two fixed-length input records with the base fields, ON field and WITH fields as shown:

```
BASE1  ON1    BASE2           BASE3 BASE4  GGGGG
       ON1            WITHA                WITHB
```

the resulting spliced output record would be:

```
BASE1  ON1    BASE2   WITHA   BASE3 BASE4  WITHB
```

For variable-length records, by default (without VLENMAX or VLENOVLY), the spliced record has the same length as the base record, and WITH fields from the overlay record that are beyond the end of the base record do not appear in the spliced record. For example, if we had the following two records with the lengths (in the RDW), ON field and WITH fields as shown:

```
25 | ON1 BASE1       BASE2
35 | ON1       WITH1           WITH2
```

the resulting spliced output record would be:

```
25 | ON1 BASE1 WITH1 BASE2
```

The WITH2 field is beyond the end of the base record, so it is not spliced.

However, if you specify VLENMAX, the spliced record is given the larger of the base record length or overlay record length. If we specify VLENOVLY, the spliced record is given the overlay record length. In either case, if the overlay record length is larger, bytes in the extended spliced record that are not overlaid are filled in with blanks.

The resulting spliced output record with either VLENMAX or VLENOVLY would be:

```
35  ON1 BASE1 WITH1 BASE2     WITH2
```

You can use VLENMAX when you want the spliced record to have the maximum length of the base or overlay record. You can use VLENOVLY when you want the spliced record to have the length of the overlay record, regardless of whether it's longer or shorter than the base record. Without VLENMAX or VLENOVLY, the spliced record has the length of the base record regardless of whether it's longer or shorter than the overlay record.

For fixed-length records, the length of the base, overlay and spliced records are all the same. Thus, VLENOVLY and VLENMAX have no meaning for fixed-length records and are ignored.

WITHALL can be used to create multiple spliced records for each set of duplicates. The first duplicate is spliced with the specified fields from the second duplicate. Then the first duplicate is spliced with the specified fields from the third duplicate, and so on.

The first duplicate is treated as a base record. Each subsequent duplicate is treated as an overlay record. The specified fields from each overlay record are overlaid on to the base record. Thus, the output records consist of fields from the base record intermixed with specified nonblank and blank fields from the overlay records.

The records to be spliced can originate from multiple input data sets.

To illustrate the splicing process when WITHALL is specified, if we had the following four fixed-length records with the base fields, ON field and WITH fields as shown:

```
BASE1  ON1    BASE2           BASE3 BASE4  GGGGG
       ON1            WITHA                WITHB
       ON1            WITHC
       ON1            WITHE                WITHF
```

The resulting three spliced output records would be:

```
BASE1  ON1    BASE2   WITHA   BASE3 BASE4  WITHB
BASE1  ON1    BASE2   WITHC   BASE3 BASE4
BASE1  ON1    BASE2   WITHE   BASE3 BASE4  WITHF
```

Note that without WITHALL, the resulting single spliced output record would be:

```
BASE1  ON1    BASE2   WITHE   BASE3 BASE4  WITHF
```

For variable-length records, by default (without VLENMAX or VLENOVLY), the spliced record has the same length as the base record, and WITH fields from the overlay record that are beyond the end of the base record do not appear in the spliced record. For example, with WITHALL, if we had the following four records with the lengths (in the RDW), ON field and WITH fields as shown:

```
30 | BASE1 ON1           BASE2
25 |       ON1     WITHA
50 |       ON1     WITHB         WITHC       WITHD
40 |       ON1     WITHE         WITHF
```

the resulting three spliced output records would be:

```
30 | BASE1 ON1     WITHA BASE2
30 | BASE1 ON1     WITHB BASE2
30 | BASE1 ON1     WITHE BASE2
```

The WITHC, WITHD and WITHF fields are beyond the end of the base record, so they are not spliced.

However, if you specify VLENMAX, the spliced record is given the larger of the base record length or overlay record length. If we specify VLENOVLY, the spliced record is given the overlay record length. In either case, if the overlay record length is larger, bytes in the extended spliced record that are not overlaid are filled in with blanks. The resulting three spliced output records with WITHALL and VLENMAX would be:

```
30 | BASE1 ON1     WITHA BASE2
50 | BASE1 ON1     WITHB BASE2   WITHC       WITHD
40 | BASE1 ON1     WITHE BASE2   WITHF
```

The resulting three spliced output records with WITHALL and VLENOVLY would be:

```
25 | BASE1 ON1     WITHA
50 | BASE1 ON1     WITHB BASE2   WITHC       WITHD
40 | BASE1 ON1     WITHE BASE2   WITHF
```

WITHANY can be used to create one spliced record for each set of duplicates. The first duplicate is spliced with the nonblank values of each subsequent duplicate for specified fields.

The first duplicate is treated as a base record. Each subsequent duplicate is treated as an overlay record. Each specified field with a nonblank value in each overlay record is overlaid on to the base record. Thus, the output record consists of fields from the base record intermixed with specified nonblank fields from each overlay record. The value from the last overlay record with each nonblank value will appear in the output record. Note that a specified "field" from an overlay record can actually consist of multiple fields from the record that have previously been reformatted into one contiguous field.

The records to be spliced can originate from multiple input data sets.

To illustrate the splicing process when WITHANY is specified, if we had the following four fixed-length records with the base fields, ON field and WITH fields as shown:

```
BASE1  ON1     BASE2
       ON1                    WITHC
       ON1                           WITHA
       ON1            WITHB
```

The resulting spliced output record would be:

```
BASE1  ON1     BASE2  WITHB   WITHC  WITHA
```

For variable-length records, by default (without VLENMAX), the spliced record has the same length as the base record. For example, with WITHANY, if we had the following four records with the lengths (in the RDW), ON field and WITH fields as shown:

```
30 | BASE1 ON1           BASE2
50 |       ON1                              WITHB
25 |       ON1     WITHA
40 |       ON1                   WITHC
```

the resulting spliced output records would be:

```
30 | BASE1 ON1     WITHA BASE2
```

The WITHB and WITHC fields are beyond the end of the base record, so they are not spliced. However, if you specify VLENMAX, the spliced record is given the largest of the base record length or overlay record lengths. If the largest overlay record length is larger than the base record length, bytes in the extended spliced record that are not overlaid are filled in with blanks. The resulting spliced output record with WITHANY and VLENMAX would be:

```
50 | BASE1 ON1     WITHA BASE2   WITHC       WITHB
```

VLENOVLY cannot be specified with WITHANY.

WITHEACH can be used to create one spliced record for each set of duplicates. The first duplicate is spliced with one specified field from each subsequent duplicate.

The first duplicate is treated as a base record. Each subsequent duplicate is treated as an overlay record. The specified blank or nonblank field from each overlay record is overlaid on to the base record. Thus, the output record consists of fields from the base record intermixed with a specified nonblank or blank field from each overlay record. Note that the specified "field" from an overlay record can actually consist of multiple fields from the record that have previously been reformatted into one contiguous field.

The records to be spliced can originate from multiple input data sets

To illustrate the splicing process when WITHEACH is specified, if we had the following four records with the base fields, ON field and WITH fields as shown:

```
BASE1  ON1     BASE2
       ON1             WITHA
       ON1                     WITHB
       ON1                             WITHC
```

The resulting spliced output record would be:

```
BASE1  ON1     BASE2  WITHA   WITHB  WITHC
```

For variable-length records, by default (without VLENMAX), the spliced record has the same length as the base record. For example, with WITHEACH, if we had the following four records with the lengths (in the RDW), ON field and WITH fields as shown:

```
30 | BASE1 ON1           BASE2
25 |       ON1     WITHA
50 |       ON1                               WITHB
40 |       ON1                   WITHC
```

the resulting spliced output records would be:

```
30 | BASE1 ON1     WITHA BASE2
```

The WITHB and WITHC fields are beyond the end of the base record, so they are not spliced.

However, if you specify VLENMAX, the spliced record is given the largest of the base record length or overlay record lengths. If the largest overlay record length is larger than the base record length, bytes in the extended spliced record that are not overlaid are filled in with blanks. The resulting spliced output record with WITHEACH and VLENMAX would be:

```
50 | BASE1 ON1     WITHA BASE2   WITHC       WITHB
```

VLENOVLY cannot be specified with WITHEACH.

KEEPNODUPS can be used to keep the non-duplicate records as well as the spliced records. The non-duplicate records will be unchanged.

To illustrate the splicing process when KEEPNODUPS is specified, if we had the following six records with the base fields, ON fields and WITH fields as shown:

```
UNIQA  ONA
BASEA  ONB
DUPAA  ONB    WITHA
UNIQB  ONC
BASEB  OND
DUPBB  OND    WITHB
```

The two unique records (ONA and OND) would be kept along with the two spliced records (ONB and OND). The resulting four unspliced and spliced output records would be:

```
UNIQA  ONA
BASEA  ONB    WITHA
UNIQB  ONC
BASEB  OND    WITHB
```

Note that without KEEPNODUPS, the two unique records (ONA and ONC) would not be kept. The resulting two spliced output records would be:

```
BASEA  ONB    WITHA
BASEB  OND    WITHB
```

KEEPBASE can be used to keep the base records (first duplicate) as well as the spliced records. The base records will be unchanged.

To illustrate the splicing process when KEEPBASE is specified, if we had the following six records with the base fields, ON fields and WITH fields as shown:

```
UNIQA  ONA
BASEA  ONB
DUPAA  ONB    WITHA
UNIQB  ONC
BASEB  OND
DUPBB  OND    WITHB
```

The two base records with duplicates (first ONB record and first OND record) would be kept along with the two spliced records (ONB and OND). The resulting four unspliced and spliced output records would be:

```
BASEA  ONB
BASEA  ONB    WITHA
BASEB  OND
BASEB  OND    WITHB
```

Note that without KEEPBASE, the two base records with duplicates (first ONB record and first OND record) would not be kept. The resulting two spliced output records would be:

```
BASEA  ONB    WITHA
BASEB  OND    WITHB
```

If we used KEEPNODUPS and KEEPBASE with the original six records, the resulting six unspliced and spliced output records would be:

```
UNIQA  ONA
BASEA  ONB
BASEA  ONB    WITHA
UNIQB  ONC
BASEB  OND
BASEB  OND    WITHB
```

DFSORT is called to sort the indd data set. ICETOOL uses its E35 exit to determine which records to splice and include in the outdd data set. ICETOOL passes the EQUALS option to DFSORT to ensure that duplicates are kept in their original input order.

The DFSORT control statements in xxxxCNTL are used if USING(xxxx) is specified.

Do not supply your own MODS, SUM, OUTREC, or SORT statement.

You can use comment statements. You can use INCLUDE, OMIT, INREC, OPTION, and OUTFIL statements providing you observe these rules:

- You can use an INCLUDE or OMIT statement to remove input records **before** SPLICE processing.
- You can use an INREC statement to reformat input records **before** SPLICE processing; the base and overlay records are reformatted according to the INREC statement. You can use INREC's PARSE, BUILD (FIELDS), OVERLAY, FINDREP, IFTHEN, or IFOUTLEN functions. If your INREC statement changes the starting position of an ON field or WITH field, you must specify the new starting position for that ON field

or WITH field. For example, if your input records have a CH key at positions 1-5 and a WITH field at 6-8 and you use an INREC statement like this:

```
INREC FIELDS=(31:1,50)
```

you must specify ON(31,5,CH) instead of ON(1,5,CH) and WITH(36,3) instead of WITH(6,3).

- You can further process the outdd records associated with TO(outdd) **after** SPLICE processing using an OUTFIL statement like this:

```
OUTFIL FNAMES=outdd,...
```

or multiple OUTFIL statements like this:

```
OUTFIL FNAMES=outdd,...
OUTFIL FNAMES=outdd1,...
...
```

For example, with TO(OUT1) you could further modify the OUT1 records after they have been spliced, with a statement like this:

```
OUTFIL FNAMES=OUT1,FTOV,VLTRIM=X'40'
```

The DYNALLOC option is passed to DFSORT to ensure that work space is available for the sort. If your installation defaults for dynamic allocation are inappropriate for a SPLICE operator, you can specify USING(xxxx) and take one of the following actions:

1. Override the DYNALLOC option using an OPTION control statement such as:

   ```
   OPTION DYNALLOC=(,8)
   ```

   in the xxxxCNTL data set.
2. Use xxxxWKdd DD statements to override the use of dynamic allocation. Refer to "SORTWKdd DD statement" on page 69 for details.

Tape work data sets **cannot** be used with ICETOOL.

Operand descriptions

FROM(indd)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

TO(outdd)
Specifies the ddname of the output data set to which DFSORT will write the records it produces for the operation (that is, the spliced records, the non-duplicate records if KEEPNODUPS is specified, and the base records if KEEPBASE is specified).

An outdd DD statement must be present and must define an output data set that conforms to the rules for DFSORT's SORTOUT data set.

The ddname specified in the TO operand must not be the same as the ddname specified in the FROM operand.

Refer to "JCL restrictions" on page 546 for more information.

ON(p,m,f)
See the discussion of this operand on the SELECT statement in "SELECT operator" on page 631.

WITH(p,m)
Specifies the position and length of a field to be overlaid from the overlay record on to the base record.

p specifies the first byte of the field relative to the beginning of the overlay record. p is 1 for the first **data** byte of a fixed-length record and 5 for the first **data** byte of a variable-length record as illustrated in the following (RRRR represents the 4-byte record descriptor word):

```
   Fixed-length record    |    Variable-length record
   | D | A | T | A | ...  |    | R | R | R | R | D | A | T | A | ...
p=   1   2   3   4        | p=   1   2   3   4   5   6   7   8
```

If INREC is specified, p must refer to the record as reformatted by INREC.

m specifies the length of the field in bytes. A field must not extend beyond position 32752.

A WITH field will not be used to overlay the RDW of a variable-length base record or to overlay bytes from beyond the end of an overlay record on to a base record. When necessary, WITH fields will be adjusted to prevent these situations. For example, if WITH(1,6) is specified for a variable-length record, it will be treated as WITH(5,2) and if WITH(75,10) is specified for an 80-byte overlay record, it will be treated as WITH(75,6).

A WITH field will not be used to overlay bytes beyond the end of a base record. When necessary, WITH fields will be adjusted to prevent this situation. For example, if WITH(75,10) is specified for an 80-byte base record, it will be treated as WITH(75,6). However, if you specify VLENMAX or VLENOVLY, a WITH field can be used to overlay bytes beyond the end of a base record provided that WITH field is present in the overlay record. For example, if VLENMAX and WITH(75,10) is specified for an 80-byte base record and a 90-byte overlay record, the spliced record will have a length of 90 bytes and the WITH(75,10) field will be present at positions 75-84 followed by 6 blanks in positions 85-90.

WITHALL
Specifies that the first duplicate is spliced with specified fields from the second duplicate, and then from each subsequent duplicate in turn. All of the WITH fields (nonblank and blank) from each overlay record are overlaid on to the base record.

With WITHALL, a spliced output record is created from each base record and overlay record, resulting in n-1 spliced records for each set of n duplicates.

WITHALL overrides the default of splicing the first duplicate with all of the specified fields from the last duplicate.

WITHANY
Specifies that the first duplicate is spliced with specified nonblank fields from each subsequent duplicate. Each nonblank WITH field from each overlay record is overlaid on to the base record.

With WITHANY, a single spliced output record is created using the base record and each nonblank field from each overlay record. If more than one overlay record has a nonblank value for a WITH field, the nonblank value from the last overlay record for that WITH field will appear in the output record. Note that the specified "field" from an overlay record can actually consist of multiple fields from the record that have previously been reformatted into one contiguous field.

WITHANY overrides the default of splicing the first duplicate with all of the specified fields from the last duplicate.

VLENOVLY cannot be specified with WITHANY.

WITHEACH
Specifies that the first duplicate is spliced with one specified field from each subsequent duplicate. One WITH field (nonblank or blank) from each overlay record is overlaid on to the base record. The first WITH field specifies the bytes to be overlaid from the second duplicate record on to the first duplicate record. The second WITH field specifies the bytes to be overlaid from the third duplicate record on to the first duplicate record, and so on. For any set of duplicates, extra overlay records without matching WITH fields, or extra WITH fields without matching overlay records are ignored.

With WITHEACH, a single spliced output record is created using the base record and one field from each overlay record. Note that the specified "field" from an overlay record can actually consist of multiple fields from the record that have previously been reformatted into one contiguous field.

WITHEACH overrides the default of splicing the first duplicate with all of the specified fields from the last duplicate.

VLENOVLY cannot be specified with WITHEACH.

KEEPNODUPS
Specifies that non-duplicate records are to be kept as well as spliced records. The non-duplicate records will be unchanged.

KEEPBASE
Specifies that base records (first duplicate) are to be kept as well as spliced records. The base records will be unchanged.

VLENMAX
Specifies that for variable-length records, the length of the spliced record is set to the maximum length of the base record and overlay record. VLENMAX overrides the default of setting the length of the spliced record to the length of the base record.

If VLENMAX is specified with or without WITHALL, the spliced record is given the larger of the base record length or overlay record length. If the overlay record length is larger than the base record length, bytes in the extended spliced record that are not overlaid are filled in with blanks.

If VLENMAX is specified with WITHANY or WITHEACH, the spliced record is given the largest of the base record length or overlay record lengths. If the largest overlay record length is larger than the base record length, bytes in the extended spliced record that are not overlaid are filled in with blanks.

For fixed-length records, VLENMAX is ignored since the base, overlay and spliced records all have the same length.

VLENOVLY
Specifies that for variable-length records, the length of the spliced record is set to the length of the overlay record. VLENOVLY overrides the default of setting the length of the spliced record to the length of the base record.

If VLENOVLY is specified with or without WITHALL, the spliced record is given the overlay record length. If the overlay record length is larger than the base record length, the spliced record is padded with blanks from the end of the base record to the new length. If the overlay record length is smaller than the base record length, bytes in the extended spliced record that are not overlaid are filled in with blanks.

VLENOVLY cannot be specified with WITHANY or WITHEACH.

For fixed-length records, VLENOVLY is ignored since the base, overlay and spliced records all have the same length.

USING(xxxx)
Specifies the first 4 characters of the ddname for the control statement data set to be used by DFSORT for this operation. xxxx must be four characters that are valid in a ddname of the form xxxxCNTL. xxxx must not be SYSx.

If USING is specified, an xxxxCNTL DD statement must be present and the control statements in it:

1. Must conform to the rules for DFSORT's SORTCNTL data set.
2. Should generally be used only for an INCLUDE or OMIT statement, an INREC statement, comment statements, or appropriate OUTFIL statements as described for “SPLICE operator” on page 644.

VSAMTYPE(x)
See the discussion of this operand on the COPY statement in “COPY operator” on page 547.

UZERO
See the discussion of this operand on the OCCUR statement in “OCCUR operator” on page 612.

SPLICE examples

SPLICE normally requires reformatting the records of two or more data sets so they can be joined, so complete JCL examples are shown in this section to illustrate the suggested techniques. These techniques and others can be employed with SPLICE to perform a variety of tasks.

Because SPLICE overlays the WITH fields from the overlay record to the base record using matching ON fields, it's usually necessary to do some initial setup before using SPLICE, to ensure that:

- the ON fields are in the same positions in the base and overlay records
- the WITH fields in the overlay records are in the positions they will occupy in the base records
- the base records and overlay records are the same length. This is always required for fixed-length records, and is required for variable-length records unless VLENMAX or VLENOVLY is specified.

For optimum efficiency, it is also a good idea to remove any records that are not needed for the SPLICE operation as part of the initial setup before the SPLICE operation, by using appropriate INCLUDE or OMIT statements.

Example 1 - Create one spliced record for each match in two files

This example shows how you can splice data together for each pair of records with the same ON field in two different input data sets.

```
//S1    EXEC  PGM=ICETOOL
//TOOLMSG   DD  SYSOUT=*
//DFSMSG    DD  SYSOUT=*
//IN1 DD *
Y12 89503 MKT
Y12 57301 MKT
Z35 02316 DEV
Y12 91073 MKT
Z35 18693 DEV
/*
//IN2 DD *
89503 27M $9,185,354 SAN JOSE   CA
72135 08M   $317,632 BOSTON     MA
18693 10M $8,732,105 BUFFALO    NY
57301 50M    $30,000 NEWARK     NJ
/*
//TEMP1 DD DSN=&&TEMP1,DISP=(MOD,PASS),SPACE=(TRK,(5,5)),UNIT=SYSDA
//COMBINE DD SYSOUT=*
//TOOLIN   DD *
* Reformat the File1 records for splicing
  COPY FROM(IN1) TO(TEMP1) USING(CTL1)
* Reformat the File2 records for splicing
  COPY FROM(IN2) TO(TEMP1) USING(CTL2)
* Splice the needed data from File1 and File2 together
  SPLICE FROM(TEMP1) TO(COMBINE) ON(5,5,ZD) WITH(15,17)
/*
//CTL1CNTL DD *
  OUTREC FIELDS=(1,14,      file1 data
                 31:X)      add blanks for spliced file2 data
/*
//CTL2CNTL DD *
  OUTREC FIELDS=(5:1,5,     put file2 key in same place as file1 key
                 15:7,15,   file2 data
                 30:33,2)   file2 data
/*
```

The base records originate from the IN1 data set and are copied and reformatted to the TEMP1 data set. The reformatted TEMP1 records are 31 bytes long and look like this:

```
Y12 89503 MKT
Y12 57301 MKT
Z35 02316 DEV
Y12 91073 MKT
Z35 18693 DEV
```

The overlay records originate from the IN2 data set and are copied and reformatted to the end (MOD) of the TEMP1 data set. The reformatted TEMP1 records are 31 bytes long and look like this:

```
    89503     27M $9,185,354 CA
    72135     08M   $317,632 MA
    18693     10M $8,732,105 NY
    57301     50M    $30,000 NJ
```

Note that MOD is used for the TEMP1 data set, so the reformatted records from IN1 and IN2 will be output to the TEMP1 data set in that order, ensuring that they are spliced in that order.

The base and overlay records from the TEMP1 data set are sorted and spliced to the COMBINE data set.

The records look like this **after** they are sorted on the 5,5,ZD field, but **before** they are spliced. As a visual aid, the WITH fields in the overlay records are shown in bold.

```
Z35 02316 DEV
Z35 18693 DEV
    18693     10M $8,732,105 NY
Y12 57301 MKT
    57301     50M    $30,000 NJ
    72135     08M   $317,632 MA
Y12 89503 MKT
    89503     27M $9,185,354 CA
Y12 91073 MKT
```

The spliced COMBINE records are 31 bytes long and look like this:

```
Z35 18693 DEV 10M $8,732,105 NY
Y12 57301 MKT 50M    $30,000 NJ
Y12 89503 MKT 27M $9,185,354 CA
```

Note that the base records for 18693, 57301 and 89503 have been spliced together with their respective overlay records.

Here is what the various ICETOOL operators do in this job:

- The first COPY operator creates reformatted IN1 records in TEMP1. The second COPY operator creates reformatted IN2 records in TEMP1. The reformatted IN1 records have blanks where the reformatted IN2 WITH fields will go. The reformatted IN2 records have the ON field from IN2 in the same place as in the reformatted IN1 records, and have the IN2 data where we want it to go in the reformatted IN1 records. We made the reformatted IN1 and reformatted IN2 records the same size so we can put them all in the TEMP1 data set and use TEMP1 as input to the SPLICE operator.
- The SPLICE operator sorts the records from TEMP1 using the ON field. TEMP1 has the reformatted IN1 records before the reformatted IN2 records. The spliced records are created from the base records and the overlay records in TEMP1. Whenever two records are found with the same ON field, the WITH field from the second record (reformatted IN2 overlay record) is overlaid on to the first record (reformatted IN1 base record). The resulting spliced records are written to the COMBINE data set.

Example 2 - Combine complete records from four files

This example shows how you can use the WITHEACH operand to splice complete records from four different data sets together.

```
//S2    EXEC  PGM=ICETOOL
//TOOLMSG   DD  SYSOUT=*
//DFSMSG    DD  SYSOUT=*
//FILE1 DD DSN=...  input file1 - 300-byte records
//FILE2 DD DSN=...  input file2 - 400-byte records
//FILE3 DD DSN=...  input file3 - 150-byte records
//FILE4 DD DSN=...  input file4 -  20-byte records
//** BE SURE TO USE MOD FOR T1
//T1 DD DSN=&&TX,UNIT=SYSDA,SPACE=(CYL,(5,5)),
// DISP=(MOD,PASS)
//ALLRCDS DD DSN=...  output file - 870-byte records
//TOOLIN DD *
* Reformat the File1 records for splicing on added sequence number
  COPY FROM(FILE1) TO(T1) USING(CTL1)
* Reformat the File2 records for splicing on added sequence number
  COPY FROM(FILE2) TO(T1) USING(CTL2)
* Reformat the File3 records for splicing on added sequence number
  COPY FROM(FILE3) TO(T1) USING(CTL3)
```

```
* Reformat the File4 records for splicing on added sequence number
  COPY FROM(FILE4) TO(T1) USING(CTL4)
* Splice record-by-record on added sequence number
  SPLICE FROM(T1) TO(ALLRCDS) ON(871,8,PD) WITHEACH -
   WITH(301,400) WITH(701,150) WITH(851,20) -
   USING(CTL5)
/*
//CTL1CNTL DD *
* Reformat records to:
* 1          301        701        851        871
* File1bytes|400 blanks|150 blanks|20 blanks |seqno
  OUTREC FIELDS=(1,300,871:SEQNUM,8,PD)
/*
//CTL2CNTL DD *
* Reformat records to:
* 1          301        701        851        871
* 300 blanks|File2bytes|150 blanks|20 blanks |seqno
  OUTREC FIELDS=(301:1,400,871:SEQNUM,8,PD)
/*
//CTL3CNTL DD *
* Reformat records to:
* 1          301        701        851        871
* 300 blanks|400 blanks|File3bytes|20 blanks |seqno
  OUTREC FIELDS=(701:1,150,871:SEQNUM,8,PD)
/*
//CTL4CNTL DD *
* Reformat records to:
* 1          301        701        851        871
* 300 blanks|400 blanks|150 blanks|File4bytes|seqno
  OUTREC FIELDS=(851:1,20,871:SEQNUM,8,PD)
/*
//CTL5CNTL DD *
* Remove added sequence number from spliced records to get:
* File1bytes|File2bytes|File3bytes|File4bytes
 OUTFIL FNAMES=ALLRCDS,OUTREC=(1,870)
/*
```

Because the data sets do not have a common key, we add sequence numbers to the records from each data set and use the sequence numbers as the ON field for SPLICE. Using this technique, we can splice together the 300-byte records from FILE1, the 400-byte records from FILE2, the 150-byte records from FILE3 and the 20-byte records from FILE4, to produce 870-byte records in ALLRCDS. Conceptually, the 870-byte records in ALLRCDS would look like this:

```
File1 Record1 ... File2Record1 ... File3Record1 ... File4Record1 ...
File1 Record2 ... File2Record2 ... File3Record2 ... File4Record2 ...
...
```

The base records originate from the FILE1 data set and the overlay records originate from the FILE2, FILE3 and FILE4 data sets.

Here is what the various ICETOOL operators do in this job:

The first COPY operator creates reformatted records in the T1 data set with the FILE1 records in positions 1-300, blanks in all other positions up to 870, and a sequence number in positions 871-878. The sequence number will be 1 for the first FILE1 record, 2 for the second FILE1 record, and so on.

The second COPY operator creates reformatted records in the T1 data set with the FILE2 records in positions 301-700, blanks in all other positions up to 870, and a sequence number in positions 871-878. The sequence number will be 1 for the first FILE2 record, 2 for the second FILE2 record, and so on.

The third COPY operator creates reformatted records in the T1 data set with the FILE3 records in positions 701-850, blanks in all other positions up to 870, and a sequence number in positions 871-878. The sequence number will be 1 for the first FILE3 record, 2 for the second FILE3 record, and so on.

The fourth COPY operator creates reformatted records in the T1 data set with the FILE4 records in positions 851-870, blanks in all other positions up to 850, and a sequence number in positions 871-878. The sequence number will be 1 for the first FILE4 record, 2 for the second FILE4 record, and so on.

Note that MOD is used for the T1 data set, so the reformatted records from FILE1, FILE2, FILE3 and FILE4 will be output in that order in T1, ensuring that they are sorted and spliced in that order.

The SPLICE operator sorts the records from T1 using the sequence number as the ON field. With WITHEACH, the reformatted FILE1 records are treated as the base records, and the reformatted FILE2,

FILE3 and FILE4 records are treated as the overlay records; each WITH field is associated with an overlay record in turn. So the first WITH field specifies the bytes to be used from the second duplicate (FILE2 record), the second WITH field specifies the bytes to be used from the third duplicate (FILE3 record) and the third WITH field specifies the bytes to be used from the fourth duplicate (FILE4 record).

SPLICE matches each base and overlay record by their sequence numbers, and creates a new combined 878-byte record. The OUTFIL statement in CTL5CNTL is used to remove the sequence number so that the 870-byte spliced record is written to the ALLRCDS data set.

Example 3 - Create files with matching and non-matching records

This example shows how you can match records in input data sets 1 and 2 to produce three output data sets with:

- ON fields that appear in both input data set 1 and input data set 2
- ON fields that appear only in input data set 1
- ON fields that appear only in input data set 2

```
//S3 EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG  DD SYSOUT=*
//IN1     DD *
Vicky
Frank
Carrie
Holly
Paul
/*
//IN2      DD *
Karen
Holly
Carrie
Vicky
Mary
/*
//OUT12    DD SYSOUT=*
//OUT1     DD SYSOUT=*
//OUT2     DD SYSOUT=*
//T1       DD DSN=&&T1,DISP=(MOD,PASS),UNIT=SYSDA,SPACE=(TRK,(5,5))
//TOOLIN   DD *
* Add '11' identifier for FILE1 records.
  COPY FROM(IN1) TO(T1) USING(CTL1)
* Add '22' identifier for FILE2 records.
  COPY FROM(IN2) TO(T1) USING(CTL2)
* SPLICE to match up records and write them to their
* appropriate output files.
  SPLICE FROM(T1) TO(OUT12) ON(1,10,CH) WITH(13,1) -
    USING(CTL3) KEEPNODUPS
/*
//CTL1CNTL DD *
* Mark FILE1 records with '11'
  OUTREC FIELDS=(1,10,12:C'11')
/*
//CTL2CNTL DD *
* Mark FILE2 records with '22'
  OUTREC FIELDS=(1,10,12:C'22')
/*
//CTL3CNTL DD *
* Write matching records to OUT12 file.  Remove id.
  OUTFIL FNAMES=OUT12,INCLUDE=(12,2,CH,EQ,C'12'),OUTREC=(1,10)
* Write FILE1 only records to OUT1 file.  Remove id.
  OUTFIL FNAMES=OUT1,INCLUDE=(12,2,CH,EQ,C'11'),OUTREC=(1,10)
* Write FILE2 only records to OUT2 file.  Remove id.
  OUTFIL FNAMES=OUT2,INCLUDE=(12,2,CH,EQ,C'22'),OUTREC=(1,10)
/*
```

We copy the IN1 records to the T1 data set and add an identifier of '11' to show they come from FILE1.

We copy the IN2 records to the end (MOD) of the T1 data set and add an identifier of '22' to show they come from FILE2.

We sort the records of T1 on positions 1-3 and splice the second id byte for matching records. We use KEEPNODUPS to keep non-duplicate records.

The records look like this **after** they are sorted, but **before** they are spliced:

```
Carrie     11
Carrie     22
Frank      11
Holly      11
Holly      22
Karen      22
Mary       22
Paul       11
Vicky      11
Vicky      22
```

The records look like this **after** they are spliced, but **before** we do the OUTFIL processing specified by CTL3CNTL with USING(CTL3) for SPLICE:

```
Carrie     12
Frank      11
Holly      12
Karen      22
Mary       22
Paul       11
Vicky      12
```

An id of 12 indicates an ON field that appears in IN1 and IN2. An id of 11 indicates an ON field that appears only in IN1. An id of 22 indicates an ON field that appears only in IN2.

The OUTFIL statements in CTL3CNTL write the records to their appropriate output data sets (without the ids) as follows:

OUT12 contains:

```
Carrie
Holly
Vicky
```

OUT1 contains:

```
Frank
Paul
```

OUT2 contains:

```
Karen
Mary
```

Example 4 - Create a matrix of values

This example shows how you can use the WITHANY operand to create a matrix of values. We combine multiple rows for different types of data with a common key into a single row of data for that key, even if some rows are missing.

```
//S4 EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG  DD SYSOUT=*
//IN  DD *
001 3  150
001 2  120
001 1  100
002 2  140
002 3  250
003 1  050
003 3  920
004 3  005
/*
//OUT DD SYSOUT=*
//TOOLIN DD *
* Splice nonblank type 1 values (in 5-7), type 2 values (9-11)
* and type 3 values (13-15) for each key.
SPLICE FROM(IN) TO(OUT) ON(1,3,CH) WITHANY KEEPNODUPS -
  WITH(5,3) WITH(9,3) WITH(13,3) USING(CTL1)
/*
```

```
//CTL1CNTL DD *
* Before SPLICE:
* Reformat type 1 records to:
* 1   5
* key val
  INREC IFTHEN=(WHEN=(5,1,CH,EQ,C'1'),
          BUILD=(1,3,5:8,3)),
* Reformat type 2 records to:
* 1       9
* key     val
     IFTHEN=(WHEN=(5,1,CH,EQ,C'2'),
          BUILD=(1,3,9:8,3)),
* Reformat type 3 records to:
* 1          13
* key         val
     IFTHEN=(WHEN=(5,1,CH,EQ,C'3'),
          BUILD=(1,3,13:8,3))
/*
```

The input records look like this:

```
001 3  150
001 2  120
001 1  100
002 2  140
002 3  250
003 1  050
003 3  920
004 3  005
```

We have a key (for example, '001') in positions 1-3, a record type (1, 2 or 3) in position 5 and a numeric value in positions 8-10. We want to set up a single row for each key with the values for the three record types and blanks for missing record types.

Before we SPLICE the records, we use the INREC IFTHEN clauses to put the values from the type 1 records in positions 5-7, the values from the type 2 records in positions 9-11 and the values from the type 3 records in positions 13-15. The reformatted INREC records look like this:

```
001         150
001     120
001 100
002     140
002         250
003 050
003         920
004         005
```

We use SPLICE with WITHANY and appropriate WITH fields to create one combined record for each key with the values for the record types. We use KEEPNODUPS to keep records for keys with only one value (for example, '004').

The OUT records are 15 bytes long and look like this:

```
001 100 120 150
002     140 250
003 050     920
004         005
```

Note that if we used WITHEACH instead of WITHANY, the output would not be what we wanted since record types are missing for some keys (for example, type 1 is missing for key 002), and record types are out of order for some keys (for example, we have type 3, type 2 and type 1 in that order for key 001). With WITHEACH, the OUT records would look like this:

```
001         150
002     140
003
004         005
```

Example 5 - Create multiple spliced records for each match in two types of records

This example shows how you can use the WITHALL operand to tell ICETOOL to splice data together for a single record of one type (A records) and multiple records of another type (H records), in the same input data set, that all have the same ON field (duplicate records). It also shows how to ensure that duplicates of the second type without a match of the first type are not written to the output data set. IFTHEN clauses are used in an INREC statement to reformat the two types of records appropriately before they are sorted and spliced

```
//S5    EXEC  PGM=ICETOOL
//TOOLMSG   DD  SYSOUT=*
//DFSMSG    DD  SYSOUT=*
//MAST DD *
A0000B0000KRSC0000D000000E0000F00G000
A1111B1111FLYC1111D111111E1111F11G111
H02KRSI000002J002K002L02
H03FLYI000003J003K003L03
H04VQXI000004J004K004L04
H05FLYI000005J005K005L05
H06KHNI000006J006K006L06
H07KRSI000007J007K007L07
H08FLYI000008J008K008L08
H09KHNI000009J009K009L09
/*
//OUT DD SYSOUT=*
//TOOLIN   DD *
* Splice needed base and overlay data together.
* Do NOT splice identifier.
  SPLICE FROM(MAST) TO(OUT) WITH(1,7) WITH(13,4) ON(20,3,CH) -
   WITH(23,3) WITH(26,3) WITHALL USING(CTL1)
/*
//CTL1CNTL DD *
* Before SPLICE:
* Set up fields in base (A) records.  Add 'B' id in position 33.
  INREC IFTHEN=(WHEN=(1,1,CH,EQ,C'A'),
     BUILD=(8:14,5,17:31,3,20:11,3,29:34,4,33:C'B')),
* Set up fields in overlay (H) records.  Add 'V' id in position 33.
   IFTHEN=(WHEN=(1,1,CH,EQ,C'H'),
   BUILD=(1:7,7,13:18,4,20:4,3,23:1,3,26:22,3,33:C'V'))
* After SPLICE:
* Remove duplicate overlay records without matching base record.
* Remove base or overlay indicator.
  OUTFIL FNAMES=OUT,OMIT=(33,1,CH,EQ,C'V'),OUTREC=(1,32)
/*
```

The base records are the records in MAST with an 'A' in column 1. They are reformatted by the INREC statement as 33 byte records that look like this:

```
       C0000    F00KRS      G000B
       C1111    F11FLY      G111B
```

We put a 'B' in position 33 to identify these records as base records.

The overlay records are the records in MAST with an 'H' in column 1. They are reformatted by the INREC statement as 33 bytes records that look like this:

```
I000002     K002   KRSH02L02    V
I000003     K003   FLYH03L03    V
I000004     K004   VQXH04L04    V
I000005     K005   FLYH05L05    V
I000006     K006   KHNH06L06    V
I000007     K007   KRSH07L07    V
I000008     K008   FLYH08L08    V
I000009     K009   KHNH09L09    V
```

We put a 'V' in position 33 to identify these records as overlay records.

The base and overlay records set up by the INREC statement are sorted and spliced.

The records look like this **after** they are sorted on the 20,3,CH field, but **before** they are spliced. As a visual aid, the WITH fields in the overlay records are shown in bold.

```
       C1111    F11FLY      G111B
I000003     K003   FLYH03L03    V
I000005     K005   FLYH05L05    V
I000008     K008   FLYH08L08    V
I000006     K006   KHNH06L06    V
I000009     K009   KHNH09L09    V
       C0000    F00KRS      G000B
I000002     K002   KRSH02L02    V
I000007     K007   KRSH07L07    V
I000004     K004   VQXH04L04    V
```

The spliced output records are 33 bytes long and look like this:

```
I000003C1111K003F11FLYH03L03G111B
I000005C1111K005F11FLYH05L05G111B
I000008C1111K008F11FLYH08L08G111B
I000009     K009   KHNH09L09    V
I000002C0000K002F00KRSH02L02G000B
I000007C0000K007F00KRSH07L07G000B
```

Note that the base record (type A) for FLY has been spliced together with each of the three overlay records (type H) for FLY. Likewise, the base record (type A) for KRS has been spliced together with each of the two overlay records (type H) for KRS.

But also note that the overlay records (type H) for KHN have been spliced together. Because KHN does not appear as a base record (type A) we don't want the KHN records to appear in the OUT data set. So we will use the 'V' we put in position 33 for the overlay records to identify and delete spliced overlay records without a matching base record. We only have to do this if we have duplicate overlay records without a matching base record. Single overlay records without a matching base record will be deleted automatically (unless you specify KEEPNODUPS).

After we eliminate the spliced overlay records and the position 33 indicator, the OUT records are 32 bytes long and look like this:

```
I000003C1111K003F11FLYH03L03G111
I000005C1111K005F11FLYH05L05G111
I000008C1111K008F11FLYH08L08G111
I000002C0000K002F00KRSH02L02G000
I000007C0000K007F00KRSH07L07G000
```

Note that if we had not specified WITHALL, only the first and last records for each set of duplicates would have been spliced, producing the following output:

```
I000008C1111K008F11FLYH08L08G111
I000007C0000K007F00KRSH07L07G000
```

Example 6 - Pull records from a master file in sorted order

This example shows how you can use the WITHALL operand to tell ICETOOL to use ON fields in a "PULL" data set to select one or more records from a "MASTER" data set. In other words, you can use a PULL list to select records from a MASTER list. In this case, the PULL data set has VB records and the MASTER data set has FB records. The ON field is a City Name that can be 1-20 bytes long, and the selected MASTER records are to be sorted by the City Name. (“Example 7 - Pull records from a master file in their original order” on page 661 shows how to keep the MASTER records in their original order.)

```
//S6    EXEC  PGM=ICETOOL
//TOOLMSG   DD  SYSOUT=*
//DFSMSG    DD  SYSOUT=*
//PULL DD DSN=VAR.PULL.FILE,DISP=SHR
//MASTER DD DSN=FIXED.MASTER.FILE,DISP=SHR
//TEMP1 DD DSN=&&T1,DISP=(MOD,PASS),SPACE=(TRK,(5,5)),UNIT=SYSDA
//OUT DD DSN=FIXED.OUTPUT.FILE,DISP=(NEW,CATLG,DELETE),
//       SPACE=(TRK,(5,5)),UNIT=SYSDA
//TOOLIN DD *
* Convert PULL records from VB to FB and add 'P' identifier.
 COPY FROM(PULL) USING(CTL1)
```

```
* Add 'M' identifier to MASTER records.
 COPY FROM(MASTER) TO(TEMP1) USING(CTL2)
* Splice PULL and MASTER records (do NOT splice identifier):
*   Spliced MASTER records with matching PULL records have 'P' id.
*   Spliced MASTER records without matching PULL records
*   have 'M' id.
* Eliminate records with 'M' id.
 SPLICE FROM(TEMP1) TO(OUT) ON(1,20,CH) WITHALL WITH(1,40) -
   USING(CTL3)
/*
//CTL1CNTL DD *
* Convert PULL records from VB to FB and add 'P' identifier.
  OUTFIL FNAMES=TEMP1,VTOF,OUTREC=(5,20,41:C'P')
/*
//CTL2CNTL DD *
* Add 'M' identifier to MASTER records.
  OUTREC FIELDS=(1,40,41:C'M')
/*
//CTL3CNTL DD *
* Eliminate MASTER records without matching PULL records.
  OUTFIL FNAMES=OUT,OMIT=(41,1,CH,EQ,C'M'),OUTREC=(1,40)
/*
```

The base records originate from the PULL data set (VAR.PULL.FILE). The PULL data set has variable-length (VB) records with the RDW in positions 1-4 and the variable-length City Name starting in position 5 for 1-20 bytes. Conceptually, the PULL records look like this:

```
Length | Data
    12 | SAN JOSE
    12 | NEW YORK
    11 | DENVER
    15 | LOS ANGELES
```

The overlay records originate from the MASTER data set (FIXED.MASTER.FILE). The MASTER data set has 40-byte fixed-length (FB) records with the City Name in positions 1-20.

The PULL records are copied and reformatted to the TEMP1 data set as 41-byte fixed-length (FB) records with the City Name in positions 1-20 (padded on the right with blanks as necessary), and a 'P' in position 41 to identify them as PULL records. The VTOF and OUTREC parameters of DFSORT's OUTFIL statement are used to convert the VB records to FB records with blank padding. The reformatted PULL records in TEMP1 look like this:

```
SAN JOSE                              P
NEW YORK                              P
DENVER                                P
LOS ANGELES                           P
```

The MASTER records are copied and reformatted to the end (MOD) of the TEMP1 data set as 41-byte fixed-length (FB) records with an 'M' added in position 41 to identify them as MASTER records. The reformatted MASTER records in TEMP1 look like this:

```
SAN JOSE             8630  SUSAN        M
PHOENIX              7993  PAUL         M
LOS ANGELES          9203  MICHAEL      M
SAN JOSE             0052  VICKY        M
NEW YORK             5218  CARRIE       M
SAN JOSE             3896  FRANK        M
TUCSON               1056  LISA         M
NEW YORK             6385  MICHAEL      M
PHOENIX              5831  HOLLY        M
```

The base and overlay records from the TEMP1 data set are sorted and spliced.

The records look like this **after** they are sorted on the 1,20,CH field, but **before** they are spliced. As a visual aid, the WITH fields in the overlay records are shown in bold.

```
DENVER                                P
LOS ANGELES                           P
LOS ANGELES          9203  MICHAEL      M
NEW YORK                              P
NEW YORK             5218  CARRIE       M
NEW YORK             6385  MICHAEL      M
PHOENIX              7993  PAUL         M
```

```
PHOENIX              5831  HOLLY        M
SAN JOSE                                P
SAN JOSE             8630  SUSAN        M
SAN JOSE             0052  VICKY        M
SAN JOSE             3896  FRANK        M
TUCSON               1056  LISA         M
```

The spliced records look like this:

```
LOS ANGELES          9203  MICHAEL      P
NEW YORK             5218  CARRIE       P
NEW YORK             6385  MICHAEL      P
PHOENIX              5831  HOLLY        M
SAN JOSE             8630  SUSAN        P
SAN JOSE             0052  VICKY        P
SAN JOSE             3896  FRANK        P
```

Finally, we use the OUTFIL statement for SPLICE to remove each spliced record with an 'M' in position 41, because that represents a base record without a matching overlay record. The OUTFIL statement also removes the 'P' indicator in position 41 from each record, because it is not needed in the OUT data set.

Thus, for each MASTER record that matches a PULL record, we've overlaid the PULL record with the MASTER record. This effectively selects all of the MASTER records on the PULL list. The resulting OUT data set (FIXED.OUTPUT.FILE) has the following 40-byte fixed-length records:

```
LOS ANGELES          9203  MICHAEL
NEW YORK             5218  CARRIE
NEW YORK             6385  MICHAEL
SAN JOSE             8630  SUSAN
SAN JOSE             0052  VICKY
SAN JOSE             3896  FRANK
```

Example 7 - Pull records from a master file in their original order

This example is similar to “Example 6 - Pull records from a master file in sorted order” on page 659, except that we want to keep the resulting MASTER records in their original order instead of sorting them by the City Name field. We use DFSORT's SEQNUM parameter to add a sequence number to each MASTER record before the records are spliced, and we splice that sequence number along with the data. After SPLICE sorts by the City Name, we SORT again by the sequence number to get the resulting MASTER records back in their original order.

```
//S7    EXEC  PGM=ICETOOL
//TOOLMSG   DD  SYSOUT=*
//DFSMSG    DD  SYSOUT=*
//PULL DD DSN=VAR.PULL.FILE,DISP=SHR
//MASTER DD DSN=FIXED.MASTER.FILE,DISP=SHR
//TEMP1 DD DSN=&&TEMP1,DISP=(MOD,PASS),SPACE=(TRK,(5,5)),UNIT=SYSDA
//TEMP2  DD DSN=&&TEMP2,DISP=(,PASS),SPACE=(TRK,(5,5)),UNIT=SYSDA
//OUT DD DSN=FIXED.OUTPUT.FILE,DISP=(NEW,CATLG,DELETE),
//       SPACE=(TRK,(5,5)),UNIT=SYSDA
//TOOLIN DD *
* Convert PULL records from VB to FB and add 'P' identifier.
 COPY FROM(PULL) USING(CTL1)
* Add sequence number and 'M' identifier to MASTER records.
 COPY FROM(MASTER) TO(TEMP1) USING(CTL2)
* Splice PULL and MASTER records (splice sequence number, but
* do NOT splice identifier):
*   Spliced MASTER records with matching PULL records have 'P' id.
*   Spliced MASTER records without matching PULL records
*   have 'M' id.
* Eliminate records with 'M' id.
 SPLICE FROM(TEMP1) TO(TEMP2) ON(1,20,CH) WITHALL WITH(1,48) -
   USING(CTL3)
* Sort resulting spliced records on original sequence number
* to get them back in their original order.
* Remove id and sequence number.
 SORT FROM(TEMP2) TO(OUT) USING(CTL4)
/*
//CTL1CNTL DD *
* Convert PULL records from VB to FB and add 'P' identifier
  OUTFIL FNAMES=TEMP1,VTOF,OUTREC=(5,20,49:C'P')
/*
//CTL2CNTL DD *
```

```
* Add sequence number and 'M' identifier to MASTER records.
  OUTREC FIELDS=(1,40,41:SEQNUM,8,BI,49:C'M')
/*
//CTL3CNTL DD *
* Eliminate MASTER records without matching PULL records.
  OUTFIL FNAMES=TEMP2,OMIT=(49,1,CH,EQ,C'M')
/*
//CTL4CNTL DD *
* Sort on sequence number and remove id and sequence number.
  SORT FIELDS=(41,8,BI,A)
  OUTREC FIELDS=(1,40)
/*
```

The resulting OUT data set (FIXED.OUTPUT.FILE) has the following 40-byte fixed-length records:

```
SAN JOSE             8630  SUSAN
LOS ANGELES          9203  MICHAEL
SAN JOSE             0052  VICKY
NEW YORK             5218  CARRIE
SAN JOSE             3896  FRANK
NEW YORK             6385  MICHAEL
```

Example 8 - Create a report showing if needed parts are on-hand

This example shows how you can use the KEEPNODUPS operand to tell ICETOOL to compare the ON fields in a list of needed parts to the ON fields in a list of on-hand parts, and produce a report showing if each needed part is on-hand or not.

```
//S8       EXEC PGM=ICETOOL
//TOOLMSG  DD SYSOUT=*
//DFSMSG   DD SYSOUT=*
//ONHAND DD *
P62 Blue
P62 Red
G73 Blue
A27 Green
L90 Red
P63 Blue
/*
//NEEDED DD *
 2003/05/07   A27 Green
 2002/12/29   P62 Blue
 2003/03/17   A27 Blue
 2003/06/14   M92 Yellow
 2002/12/18   L90 Red
/*
//COMBINED DD DSN=&&T1,UNIT=SYSDA,SPACE=(TRK,(5,5)),
// DISP=(MOD,PASS)
//RPT  DD SYSOUT=*
//TOOLIN   DD *
* Reformat the ONHAND records for splicing.
* Add 'Yes' for found in ONHAND data set.
* Add 'O' to indicate ONHAND record.
  COPY FROM(ONHAND) TO(COMBINED) USING(CTL1)
* Reformat the NEEDED records for splicing.
* Add 'No' for missing from ONHAND data set.
* Add 'N' to indicate NEEDED record.
  COPY FROM(NEEDED) TO(COMBINED) USING(CTL2)
* Splice ONHAND and NEEDED records (splice identifier):
*   NEEDED records found in ONHAND list will have 'Yes'
*   and 'N'.
*   NEEDED records not found in ONHAND list will have 'No'
*   and 'N'.
*   ONHAND records that are not needed will have 'Yes'
*   and 'O'.
* Eliminate records with 'O'.
  SPLICE FROM(COMBINED) TO(RPT) -
    ON(1,12,CH) WITH(24,10) WITH(40,1) -
    KEEPNODUPS -
    USING(CTL3)
/*
//CTL1CNTL DD *
* Reformat ONHAND records with part in 1-12, 'Yes' in 15-17
* and 'O' in 40.
  OUTREC FIELDS=(1:1,12,15:C'Yes',40:C'O')
/*
//CTL2CNTL DD *
```

```
* Reformat NEEDED records with part in 1-12, 'No ' in 15-17,
* date in 24-33 and 'N' in 40.
  OUTREC FIELDS=(1:15,12,15:C'No',24:2,10,40:C'N')
/*
//CTL3CNTL DD *
* Eliminate ONHAND parts that do not appear in NEEDED list.
* Create the report showing if needed parts are on-hand.
  OUTFIL FNAMES=RPT,OMIT=(40,1,CH,EQ,C'O'),OUTREC=(1,33),
  HEADER2=(1:'Part',15:'On-Hand',24:'Needed by',/,
           1:'------------',15:'-------',24:'----------')
/*
```

The base records originate from the ONHAND data set and are copied and reformatted to the COMBINED data set. We put an 'O' in position 40 to identify these records as ONHAND records. The overlay records originate from the NEEDED data set and are copied and reformatted to the COMBINED data set. We put an 'N' in position 40 to identify these records as NEEDED records. Because MOD is used for the COMBINED data set, it contains the reformatted ONHAND records followed by the reformatted NEEDED records. The COMBINED records are 40 bytes long and look like this:

```
P62 Blue      Yes                     O
P62 Red       Yes                     O
G73 Blue      Yes                     O
A27 Green     Yes                     O
L90 Red       Yes                     O
P63 Blue      Yes                     O
A27 Green     No       2003/05/07     N
P62 Blue      No       2002/12/29     N
A27 Blue      No       2003/03/17     N
M92 Yellow    No       2003/06/14     N
L90 Red       No       2002/12/18     N
```

The base and overlay records from the COMBINED data set are sorted and spliced.

The records look like this **after** they are sorted on the 1,12,CH field, but **before** they are spliced. As a visual aid, the WITH fields in the overlay records are shown in bold.

```
A27 Blue      No       2003/03/17     N
A27 Green     Yes                     O
A27 Green     No       2003/05/07     N
G73 Blue      Yes                     O
L90 Red       Yes                     O
L90 Red       No       2002/12/18     N
M92 Yellow    No       2003/06/14     N
P62 Blue      Yes                     O
P62 Blue      No       2002/12/29     N
P62 Red       Yes                     O
P63 Blue      Yes                     O
```

The spliced output records are 40 bytes long and look like this:

```
A27 Blue      No       2003/03/17     N
A27 Green     Yes      2003/05/07     N
G73 Blue      Yes                     O
L90 Red       Yes      2002/12/18     N
M92 Yellow    No       2003/06/14     N
P62 Blue      Yes      2002/12/29     N
P62 Red       Yes                     O
P63 Blue      Yes                     O
```

We have three types of records as follows:

1. Records with 'Yes and 'N' are NEEDED records with an ONHAND match that have been spliced together. We want these for our report.
2. Records with 'No' and 'N' are NEEDED records without an ONHAND match that have been kept because we used KEEPNODUPS. We want these for our report.
3. Records with 'Yes' and 'O' are ONHAND records without a NEEDED match that have been kept because we used KEEPNODUPS. We do not want these for our report.

We use the OUTFIL statement for SPLICE to further process the spliced records. It omits the 'O' records, removes the 'N' byte, and sets up the headers for the report. The resulting RPT data set looks like this:

```
Part          On-Hand  Needed by
------------  -------  ----------
A27 Blue      No       2003/03/17
A27 Green     Yes      2003/05/07
L90 Red       Yes      2002/12/18
M92 Yellow    No       2003/06/14
P62 Blue      Yes      2002/12/29
```

Example 9 - Create a report showing if needed parts are on-hand - advanced

This example is a more complex variation of “Example 8 - Create a report showing if needed parts are on-hand” on page 662. It shows how you can use the WITHALL, KEEPBASE, and KEEPNODUPS operands to tell ICETOOL to compare the ON fields in a list of needed parts to the ON fields in a list of on-hand parts, and produce a report showing if each needed part is on-hand or not. However, it also has duplicate parts in the NEEDED data set, and produces a report with more information from the ONHAND and NEEDED records.

```
//S9       EXEC PGM=ICETOOL
//TOOLMSG  DD SYSOUT=*
//DFSMSG   DD SYSOUT=*
//ONHAND DD *
P62 Blue           Dallas
G73 Blue           San Jose
A27 Green          Vancouver
/*
//NEEDED DD *
 Rachel       A27 Green       Phoenix
 Monica       P62 Blue        Phoenix
 Phoebe       A27 Blue        Toronto
 Chandler     M92 Yellow      Los Angeles
 Joey         M92 Yellow      Paris
 Ross         A27 Green       Paris
/*
//COMBINED DD DSN=&&C1,UNIT=SYSDA,SPACE=(TRK,(5,5)),
// DISP=(MOD,PASS)
//TEMP1 DD DSN=&&T1,UNIT=SYSDA,SPACE=(CYL,(5,5)),DISP=(,PASS)
//RPT  DD SYSOUT=*
//TOOLIN   DD *
* Reformat the ONHAND records for splicing.
* Add 'Yes' for found and 'D' for delete record.
  COPY FROM(ONHAND) TO(COMBINED) USING(CTL1)
* Reformat the NEEDED records for splicing.
* Add 'No' for missing and 'K' for keep record.
  COPY FROM(NEEDED) TO(COMBINED) USING(CTL2)
* Splice ONHAND and NEEDED records.
* Splice in Requested by, Ship to and id fields.
* Eliminate spliced records with 'D'.
  SPLICE FROM(COMBINED) TO(TEMP1) -
    ON(1,12,CH) WITHALL KEEPBASE KEEPNODUPS USING(CTL3) -
    WITH(24,10) WITH(53,13) WITH(66,1)
* Print report.
  DISPLAY FROM(TEMP1) LIST(RPT) -
   INDENT(2) BETWEEN(2) BLANK -
   HEADER('Part') ON(1,12,CH) -
   HEADER('On-Hand') ON(15,3,CH) -
   HEADER('Requested by') ON(24,12,CH) -
   HEADER('Ship from') ON(38,13,CH) -
   HEADER('Ship to') ON(53,13,CH)
/*
//CTL1CNTL DD *
* Reformat ONHAND records with Part in 1-12, 'Yes' for found in
* 15-17, From City in 38-50 and 'D' in 66.
  OUTREC FIELDS=(1:1,12,15:C'Yes',38:20,13,66:C'D')
/*
//CTL2CNTL DD *
* Reformat NEEDED records with Part in 1-12, 'No  ' for missing in
* 15-17, Requester Name in 24-35, 'n/a' for From City in 38-40,
* To City in 53-65 and 'K' in 66.
  OUTREC FIELDS=(1:15,12,15:C'No ',24:2,10,38:C'n/a',
                 53:31,13,66:C'K')
/*
//CTL3CNTL DD *
* Eliminate ONHAND parts that do not appear in NEEDED list.
  OUTFIL FNAMES=TEMP1,OMIT=(66,1,CH,EQ,C'D')
/*
```

The base records originate from the ONHAND data set. They are copied and reformatted to the COMBINED data set. The reformatted records look like this:

```
P62 Blue      Yes                 Dallas                    D
G73 Blue      Yes                 San Jose                  D
A27 Green     Yes                 Vancouver                 D
```

The overlay records originate from the NEEDED data set and are copied and reformatted to the COMBINED data set. The reformatted records look like this

```
A27 Green     No       Rachel       n/a           Phoenix      K
P62 Blue      No       Monica       n/a           Phoenix      K
A27 Blue      No       Phoebe       n/a           Toronto      K
M92 Yellow    No       Chandler     n/a           Los Angeles  K
M92 Yellow    No       Joey         n/a           Paris        K
A27 Green     No       Ross         n/a           Paris        K
```

The base and overlay records from the COMBINED data set are sorted and spliced.

However, we need to make sure that all parts which appear in more than one NEEDED record, but do not appear in the ONHAND list, will appear in the report. For example, we have two M92 Yellow parts in the NEEDED data set that do not appear in the ONHAND data set. These two records are reformatted and appear in the COMBINED data set as follows:

```
M92 Yellow    No       Chandler     n/a           Los Angeles  K
M92 Yellow    No       Joey         n/a           Paris        K
```

ICETOOL would normally treat the first record as the base record and the second record as the overlay record. As a result, these two records would be spliced together into one record instead of two. To prevent this, and ensure that we keep both M92 Yellow parts, we must specify KEEPBASE. As a result, two records are kept: the unchanged first M92 Yellow record, and the spliced first and second M92 Yellow records (which in this case looks identical to the unspliced second record).

The records look like this **after** they are sorted on the 1,12,CH field, but **before** they are spliced. As a visual aid, the WITH fields in the overlay records are shown in bold.:

```
A27 Blue      No       Phoebe       n/a           Toronto      K
A27 Green     Yes                   Vancouver                  D
A27 Green     No       Rachel       n/a           Phoenix      K
A27 Green     No       Ross         n/a           Paris        K
G73 Blue      Yes                   San Jose                   D
M92 Yellow    No       Chandler     n/a           Los Angeles  K
M92 Yellow    No       Joey         n/a           Paris        K
P62 Blue      Yes                   Dallas                     D
P62 Blue      No       Monica       n/a           Phoenix      K
```

The spliced records look like this:

```
A27 Blue      No       Phoebe       n/a           Toronto      K
A27 Green     Yes                   Vancouver                  D
A27 Green     Yes      Rachel       Vancouver     Phoenix      K
A27 Green     Yes      Ross         Vancouver     Paris        K
G73 Blue      Yes                   San Jose                   D
M92 Yellow    No       Chandler     n/a           Los Angeles  K
M92 Yellow    No       Joey         n/a           Paris        K
P62 Blue      Yes                   Dallas                     D
P62 Blue      Yes      Monica       Dallas        Phoenix      K
```

Records with 'D' are not needed, so we use the OUTFIL statement for SPLICE to omit them. The TEMP1 records look like this:

```
A27 Blue      No       Phoebe       n/a           Toronto      K
A27 Green     Yes      Rachel       Vancouver     Phoenix      K
A27 Green     Yes      Ross         Vancouver     Paris        K
M92 Yellow    No       Chandler     n/a           Los Angeles  K
M92 Yellow    No       Joey         n/a           Paris        K
P62 Blue      Yes      Monica       Dallas        Phoenix      K
```

Although we could have used the OUTFIL statement for SPLICE to print the report, we've chosen instead to use a separate DISPLAY operator. DISPLAY requires an extra pass over the spliced records in TEMP1, but is easier to use than OUTFIL for reports. The resulting RPT data set looks like this:

```
  Part          On-Hand  Requested by  Ship from      Ship to
  ------------  -------  ------------  -------------  -------------
  A27 Blue      No       Phoebe        n/a            Toronto
  A27 Green     Yes      Rachel        Vancouver      Phoenix
  A27 Green     Yes      Ross          Vancouver      Paris
  M92 Yellow    No       Chandler      n/a            Los Angeles
  M92 Yellow    No       Joey          n/a            Paris
  P62 Blue      Yes      Monica        Dallas         Phoenix
```

Example 10 - Create spliced variable-length records from two files

This example shows how you can splice data together for each pair of records with the same ON field in two different VB input data sets, even when records are of different lengths.

```
//S10 EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//CON DD DSN=VAR.INPUT1,DISP=SHR
//    DD DSN=VAR.INPUT2,DISP=SHR
//OUT DD DSN=VAR.OUTPUT,DISP=(NEW,CATLG,DELETE),
//   SPACE=(CYL,(5,5)),UNIT=SYSDA
//TOOLIN DD *
* Splice the needed data from the two VB files together
SPLICE FROM(CON) TO(OUT) ON(5,5,CH) WITHALL -
   WITH(12,5) WITH(22,20) VLENMAX
/*
```

VAR.INPUT1 has RECFM=VB and LRECL=25. It contains the base records which look like this:

```
Length | Data
    25 | DIV01        L2
    15 | DIV02
    25 | DIV03        L6
    25 | DIV05        L8
```

VAR.INPUT2 has RECFM=VB and LRECL=50. It contains the overlay records which look like this:

```
Length | Data
    42 | DIV01  83201     FERN BROTHERS INTL
    33 | DIV01  73268     ROSS INC.
    39 | DIV02  00589     ACME PAINT SHOP
    19 | DIV05  57003
    47 | DIV05  01381     FLOWERS BY RENEE
    43 | DIV06  37982     EVERYTHING FOR PETS
```

Because some of the overlay records are longer than their corresponding base records, we use VLENMAX to ensure that none of the data from the overlay records is lost. VLENMAX ensures that the larger length between the base record and overlay record is used for the spliced record, and that blanks are added to the end of the spliced record when needed.

The base and overlay records from the concatenated data sets are sorted and spliced. VAR.OUTPUT has RECFM=VB and LRECL=50. It contains the spliced records, which look like this:

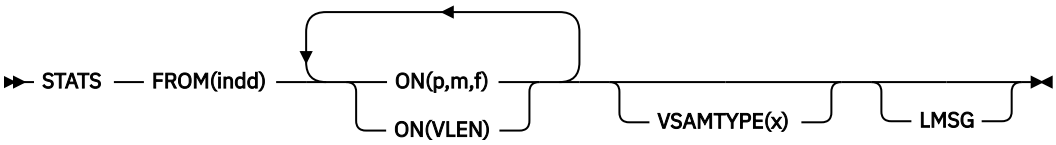
```
Length | Data
    42 | DIV01  83201 L2  FERN BROTHERS INTL
    33 | DIV01  73268 L2  ROSS INC.
    39 | DIV02  00589     ACME PAINT SHOP
    25 | DIV05  57003 L8
    47 | DIV05  01381 L8  FLOWERS BY RENEE
```

Notice that VLENMAX prevented any data from being lost. Without VLENMAX, data would have been lost; the spliced records would have looked like this:

```
Length | Data
    25 | DIV01  83201 L2  FERN
    25 | DIV01  73268 L2  ROSS
    15 | DIV02  0058
```

```
    25 | DIV05  57003 L8
    25 | DIV05  01381 L8  FLOW
```

STATS operator



Prints messages containing the minimum, maximum, average, and total for specified numeric fields. From 1 to 10 fields can be specified.

DFSORT is called to copy the indd data set to ICETOOL's E35 user exit. ICETOOL prints messages containing the minimum, maximum, average, and total for each field as determined by its E35 exit.

The average (or mean) is calculated by dividing the total by the record count and rounding down to the nearest integer (examples: 23 / 5 = 4, -23 / 5 = - 4).

You must not supply your own DFSORT MODS, INREC, or OUTREC statement, because they would override the DFSORT statements passed by ICETOOL for this operator.

Operand descriptions

The operands described in this section can be specified in any order.

FROM(indd)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

ON(p,m,f)
Specifies the position, length, and format of a numeric field to be used for this operation.

p specifies the first byte of the field relative to the beginning of the input record. p is 1 for the first **data** byte of a fixed-length record and 5 for the first **data** byte of a variable-length record as illustrated in the following (RRRR represents the 4-byte record descriptor word):

```
    Fixed-length record    |    Variable-length record
    | D | A | T | A | ...  |    | R | R | R | R | D | A | T | A | ...
 p=   1   2   3   4        | p=   1   2   3   4   5   6   7   8
```

m specifies the length of the field in bytes. A field must not extend beyond position 32752 or beyond the end of a record. The maximum length for a field depends on its format.

f specifies the format of the field as follows:

Format Code	Length	Description
BI	1 to 8 bytes	Unsigned binary
FI	1 to 8 bytes	Signed fixed-point
PD	1 to 16 bytes	Signed packed decimal
ZD	1 to 31 bytes	Signed zoned decimal
CSF or FS	1 to 32 bytes (31 digit limit)	Signed numeric with optional leading floating sign
UFF	1 to 44 bytes (31 digit limit)	Unsigned free form numeric
SFF	1 to 44 bytes (31 digit limit)	Signed free form numeric

Note: See Appendix C, "Data format descriptions," on page 823 for detailed format descriptions.

If the total for a field overflows 31 digits, ICETOOL continues processing, but prints asterisks for the average and total for that field.

For a CSF, FS, UFF, or SFF format field:

- A maximum of 31 digits is allowed. If a value with more than 31 digits is found, ICETOOL issues an error message and terminates the operation.

For a ZD or PD format field:

- If a decimal value contains an invalid digit (A-F), ICETOOL identifies the bad value in a message and prints asterisks for the minimum, maximum, average and total for that field.
- A value is treated as positive if its sign is F, E, C, A, 8, 6, 4, 2, or 0.
- A value is treated as negative if its sign is D, B, 9, 7, 5, 3, or 1.

For a ZD, PD, CSF, FS, or SFF format field, a negative zero value is treated as a positive zero value.

ON(VLEN)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

VSAMTYPE(x)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

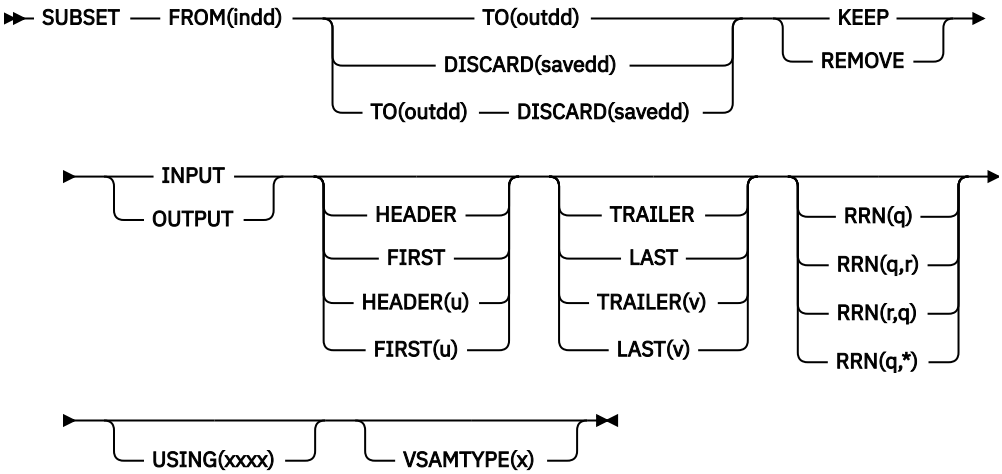
LMSG
Specifies that the minimum, maximum, average and total for all numeric fields are to be printed using messages that display 31 digits (overriding the default of printing messages that display 15 digits when possible). LMSG ensures that only message ICE648I is used to display the statistics. Without LMSG, a combination of messages ICE608I, ICE609I and ICE648I can be used to display the statistics.

STATS example

```
    STATS  FROM(DATA1)  ON(VLEN)  ON(15,4,ZD)
```

Prints messages containing the minimum, maximum, average and total of the binary values in positions 1-2 of the DATA1 data set. For variable-length records, this gives statistics about the length of the records. Prints messages containing the minimum, maximum, average and total of the zoned decimal values in positions 15-18 of the DATA1 data set.

SUBSET operator



Keeps or removes input or output records based on meeting criteria for the first n records, specific relative record numbers, and the last n records. DFSORT writes the records that are kept or not removed to the outdd data set.

DFSORT is called to copy or sort the indd data set, as appropriate. ICETOOL uses its E15 or E35 exit to determine which records to include in the outdd or savedd data set. ICETOOL passes the EQUALS option to DFSORT to ensure that duplicates are kept in their original input order if records are sorted.

If the criteria includes the last n records, ICETOOL may call DFSORT twice. For the first pass, ICETOOL counts the indd records without opening the output data sets. For the second pass, ICETOOL opens the output data sets and does SUBSET processing against the indd data set using the count obtained in the first pass.

DISCARD(savedd) can be used to write the records that are removed or not kept in the savedd data set. DISCARD(savedd) can be used with or without TO(outdd).

If you do not specify a header operand (HEADER, FIRST, HEADER(u), FIRST(u)), a relative record number operand (RRN(q), RRN(q,r), RRN(q,*)), or a trailer operand (TRAILER, LAST, TRAILER(v), LAST(v)), all of the records will be kept or removed. You can specify a header operand, relative record number operands, and a trailer operand in any combination. Records will be kept or removed according to the criteria you specify.

You can only specify one header operand. You can specify from 1 to 300 relative record number operands in any combination. You can only specify one trailer operand.

The DFSORT control statements in xxxxCNTL are used if USING(xxxx) is specified. However, you must observe these rules for control statements in the xxxxCNTL data set:

- MODS and OUTREC statements should not be present.
- SKIPREC and STOPAFT operands, and INCLUDE and OMIT statements, should not be present.
- A SORT statement can be present unless INPUT and DISCARD(savedd) are specified.
- INREC and SUM statements can be present.
- If INPUT is specified, the records selected will not be affected by INREC, SORT or SUM. If OUTPUT is specified, the records selected will be affected by INREC, SORT and SUM.
- Comment statements can be present.
- If you specify TO(outdd) without DISCARD(savedd), you can further process the outdd records **after** SUBSET processing using an OUTFIL statement like this:

  ```
  OUTFIL FNAMES=outdd,...
  ```

  or multiple OUTFIL statements like this:

  ```
  OUTFIL FNAMES=outdd,...
  OUTFIL FNAMES=outdd1,...
  ```

- If you specify DISCARD(savedd) without TO(outdd), you can further process the savedd records **after** SUBSET processing using one (and only one) OUTFIL statement like this:

  ```
  OUTFIL FNAMES=savedd,...
  ```

- If you specify TO(outdd) and DISCARD(savedd), you can further process the outdd and savedd records **after** SUBSET processing using two (and only two) OUTFIL statements like this:

  ```
  OUTFIL FNAMES=outdd,...
  ```

  or multiple OUTFIL statements like this:

  ```
  OUTFIL FNAMES=outdd,...
  OUTFIL FNAMES=savedd,...
  ```

  Both statements must be specified in the order shown with at least the FNAMES parameter. For example, to further modify only the DISCARD data set, you could use statements like this:

  ```
  OUTFIL FNAMES=OUT
  OUTFIL FNAMES=SAVE,OMIT=(21,3,ZD,GT,+25)
  ```

If you specify a SORT statement in xxxxCNTL, the DYNALLOC option is passed to DFSORT to ensure that work space is available for the sort. If your installation defaults for dynamic allocation are inappropriate for a SUBSET operator, you can take one of the following actions:

1. Override the DYNALLOC option using an OPTION control statement such as:

   ```
   OPTION DYNALLOC=(,8)
   ```

   in the xxxxCNTL data set.
2. Use xxxxWKdd DD statements to override the use of dynamic allocation. Refer to "SORTWKdd DD statement" on page 69 for details.

Tape work data sets **cannot** be used with ICETOOL.

Operand descriptions

The operands described in this section can be specified in any order.

FROM(indd)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

TO(outdd)
Specifies the ddname of the output data set to which DFSORT will write the records it selects for the operation (that is, the records that are kept or not removed according to the specified criteria).

An outdd DD statement must be present and must define an output data set that conforms to the rules for DFSORT's SORTOUT data set.

TO and DISCARD can both be specified. If DISCARD is not specified, TO must be specified. If TO is not specified, DISCARD must be specified.

The ddname specified in the TO operand must not be the same as the ddname specified in the FROM or DISCARD operand.

Refer to "JCL restrictions" on page 546 for more information.

DISCARD(savedd)
Specifies the ddname of the output data set to which DFSORT will write the records it does not select for the operation.

A savedd DD statement must be present and must define an output data set that conforms to the rules for DFSORT's OUTFIL data set.

TO and DISCARD can both be specified. If DISCARD is not specified, TO must be specified. If TO is not specified, DISCARD must be specified.

The ddname specified in the DISCARD operand must not be the same as the ddname specified in the FROM or TO operand.

Refer to "JCL restrictions" on page 546 for more information.

KEEP
Specifies that the records that meet the criteria are to be kept.

REMOVE
Specifies that the records that meet the criteria are to be removed.

INPUT
Specifies that the criteria are to be applied using the first n input records, relative input record numbers, and the last n input records. Use INPUT if you want to apply the criteria directly to the records from the indd data set.

The criteria will be applied to keep or remove records before they are reformatted by INREC, sorted by SORT and summed by SUM. The kept records will subsequently be reformatted, sorted and summed. OUTFIL will be applied to the resulting records.

As an example, if the input data set contains these records:

```
AAAA R01
AAAA R02
BBBB R03
CCCC R04
CCCC R05
CCCC R06
DDDD R07
DDDD R08
EEEE R09
EEEE R10
EEEE R11
```

and the following SUBSET operator and DFSORT control statements were specified:

```
  ...
//TOOLIN DD *
SUBSET FROM(IN) TO(OUT) KEEP INPUT RRN(3,10) USING(CTL1)
//CTL1CNTL DD *
  SORT FIELDS=(1,5,CH,D)
  SUM FIELDS=NONE
/*
```

First, input records 3-10 would be kept, so the intermediate result would be:

```
BBBB R03
CCCC R04
CCCC R05
CCCC R06
DDDD R07
DDDD R08
EEEE R09
EEEE R10
```

Then the SORT statement would be applied, so the intermediate result would be:

```
EEEE R09
EEEE R10
DDDD R07
DDDD R08
CCCC R04
CCCC R05
CCCC R06
BBBB R03
```

Finally, the SUM statement would be applied, so the final output in the OUT data set would be:

```
EEEE R09
DDDD R07
CCCC R04
BBBB R03
```

OUTPUT

Specifies that the criteria are to be applied using the first n output records, relative output record numbers, and the last n output records. Use OUTPUT if you want to apply the criteria to the indd records after they are processed by INREC, SORT and SUM as specified.

The criteria will be applied to keep or remove records after they are reformatted by INREC, sorted by SORT and summed by SUM. OUTFIL will be applied to the resulting records.

As an example, if the input data set contains these records:

```
AAAA R01
AAAA R02
BBBB R03
CCCC R04
CCCC R05
CCCC R06
DDDD R07
DDDD R08
EEEE R09
EEEE R10
EEEE R11
```

and the following SUBSET operator and DFSORT control statements were specified:

```
  ...
//TOOLIN DD *
SUBSET FROM(IN) TO(OUT) KEEP OUTPUT RRN(2,3) USING(CTL1)
//CTL1CNTL DD *
  SORT FIELDS=(1,5,CH,D)
  SUM FIELDS=NONE
/*
```

First, the SORT statement would be applied, so the intermediate result would be:

```
EEEE R09
EEEE R10
EEEE R11
DDDD R07
DDDD R08
CCCC R04
CCCC R05
CCCC R06
BBBB R03
AAAA R01
AAAA R02
```

Then the SUM statement would be applied, so the intermediate result would be:

```
EEEE R09
DDDD R07
CCCC R04
BBBB R03
AAAA R01
```

Finally, output records 2-3 would be kept, so the final output in the OUT data set would be:

```
DDDD R07
CCCC R04
```

HEADER or FIRST
Specifies one header record (the first record) is to be kept or removed.

HEADER and FIRST are equivalent to HEADER(1) and FIRST(1).

HEADER(u) or FIRST(u)
Specifies u header records (the first u records) are to be kept or removed. For example, HEADER(3) or FIRST(3) keeps or removes the first three records.

u must be specified as n or +n where n can be 1 to 999999999999999.

TRAILER or LAST
Specifies one trailer record (the last record) is to be kept or removed.

TRAILER and LAST are equivalent to TRAILER(1) and LAST(1).

TRAILER(v) or LAST(v)
Specifies v trailer records (the last v records) are to be kept or removed. For example, TRAILER(4) or LAST(4) keeps or removes the last 4 records.

v must be specified as n or +n where n can be 1 to 999999999999999.

RRN(q)
Specifies relative record number q is to be kept or removed. For example, RRN(8) keeps or removes the eighth record.

q must be specified as n or +n where n can be 1 to 999999999999999.

RRN(q,r) or RRN(r,q)
Specifies relative record numbers q through r are to be kept or removed. q can be less than, equal to, or greater than r. For example, RRN(5,10) and RRN(10,5) both keep or remove the fifth through tenth records.

q and r must be specified as n or +n where n can be 1 to 99999999999999.

RRN(q,*)
Specifies relative record numbers q through the last record are to be kept or removed. For example, RRN(7,*) keeps or removes the seventh through last records.

q must be specified as n or +n where n can be 1 to 99999999999999.

Note: RRN(*,q) is not valid

USING(xxxx)
Specifies the first 4 characters of the ddname for the control statement data set to be used by DFSORT for this operation. xxxx must be four characters that are valid in a ddname of the form xxxxCNTL. xxxx must not be SYSx.

If USING is specified, an xxxxCNTL DD statement must be present. Control statements in xxxxCNTL can be used as described for "SUBSET Operator" on page xxx, and must conform to the rules for DFSORT's SORTCNTL data set.

VSAMTYPE(x)
See the discussion of this operand on the COPY statement in “COPY operator” on page 547.

SUBSET examples

Although the SUBSET operators in the examples in this section could all be contained in a single ICETOOL job step, they are shown and discussed separately for clarity.

Example 1

```
SUBSET FROM(IN1) TO(OUT1) REMOVE INPUT HEADER TRAILER
```

This example shows how you can remove the header record (first record) and trailer record (last record).

The variable-length input records look like this:

```
Len|Data
 33|01A   RODENTS           FFFFF
 23|01A   VOLE    BINKY
 26|02B   HAMSTER GARFIELD
 22|03A   RAT     JUNE
 24|04B   MOUSE   MICKEY
 21|01A   COUNT   004
```

We just want to keep the data records. We use REMOVE, INPUT, HEADER and TRAILER to indicate we want to remove the header and trailer input records.

The output records would look like this:

```
 23|01A   VOLE    BINKY
 26|02B   HAMSTER GARFIELD
 22|03A   RAT     JUNE
 24|04B   MOUSE   MICKEY
```

Note that we could use FIRST and LAST instead of HEADER and TRAILER.

Example 2

```
SUBSET FROM(IN2) TO(OUT2) DISCARD(OUT3) -
  KEEP INPUT RRN(3,4) LAST(3)
```

This example shows how you can create one output file with relative records and the last n records, and another output file with the remaining records.

The input records look like this:

```
Algebra
Astronomy
Biology
Calculus
```

```
French
Geography
Geometry
Greek
History
Latin
Psychology
Russian
```

In the first output file (OUT2), we want the third and fourth input records and the last three input records. In the second output file (OUT3), we want the records that are not in the first output file. We use TO(OUT2) and DISCARD(OUT3) for the two output files. We use KEEP, INPUT, RRN(3,4) and LAST(3) to indicate we want to keep relative input records 3 and 4 and the last 3 input records.

The OUT2 records would look like this:

```
Biology
Calculus
Latin
Psychology
Russian
```

The OUT3 records would look like this:

```
Algebra
Astronomy
French
Geography
Geometry
Greek
History
```

Note that we could use TRAILER(3) instead of LAST(3).

Example 3

```
SUBSET FROM(IN3) TO(OUT4) KEEP OUTPUT -
  LAST(5) USING(CTL1)
```

This example illustrates how you can keep the last 5 sorted records.

The CTL1CNTL data set contains:

```
  SORT FIELDS=(1,15,CH,A)
```

The input records look like this:

```
Psychology
Biology
Russian
French
History
Geography
Calculus
Geometry
Algebra
Greek
Astronomy
Latin
```

We want to sort the records by the CH field in positions 1-15 and keep the last 5 sorted records. We use KEEP, OUTPUT and LAST(5) to keep the last 5 output records. We use the SORT statement to sort the records before they are output. After the SORT statement is executed, the intermediate records look like this:

```
Algebra
Astronomy
Biology
Calculus
French
Geography
```

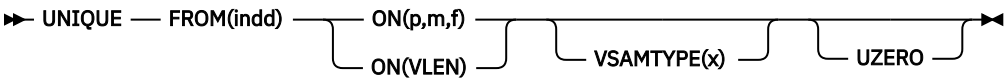
```
Geometry
Greek
History
Latin
Psychology
Russian
```

After the SUBSET operator is executed, the final output records look like this:

```
Greek
History
Latin
Psychology
Russian
```

Note that we could use TRAILER(5) instead of LAST(5).

UNIQUE operator



Prints a message containing the count of unique values for a specified numeric or character field.

DFSORT is called to sort the indd data set to ICETOOL's E35 user exit. ICETOOL prints a message containing the unique count as determined by its E35 user exit.

The DYNALLOC option is passed to DFSORT to ensure that work space is available for the sort. If your installation defaults for dynamic allocation are inappropriate for a UNIQUE operator, you can take one of the following actions:

1. Override the DYNALLOC option using an OPTION control statement such as:

   ```
   OPTION DYNALLOC=(8)
   ```

   in the DFSPARM data set.
2. Use SORTWKdd DD statements to override the use of dynamic allocation. Refer to "SORTWKdd DD statement" on page 69 for details.

Attention: Either of these actions affects the work data sets used for an OCCUR operator, or for a SELECT or SPLICE operator for which USING(xxxx) is not specified.

Tape work data sets **cannot** be used with ICETOOL.

You must not supply your own DFSORT MODS, INREC, OUTREC, SUM or RECORD statement, because they override the DFSORT statements passed by ICETOOL for this operator.

Operand descriptions

The operands described in this section can be specified in any order.

FROM(indd)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

ON(p,m,f)
Specifies the position, length, and format of a numeric or character field to be used with this operation.

p specifies the first byte of the field relative to the beginning of the input record. p is 1 for the first **data** byte of a fixed-length record and 5 for the first **data** byte of a variable-length record as illustrated in the following (RRRR represents the 4-byte record descriptor word):

```
Fixed-length record    |    Variable-length record
| D | A | T | A | ...  |    | R | R | R | R | D | A | T | A | ...
```

```
 p=   1   2   3   4        | p=   1   2   3   4   5   6   7   8
```

m specifies the length of the field in bytes. A field must not extend beyond position 32752 or beyond the end of a record. The maximum length for a field depends on its format.

f specifies the format of the field as shown in the following table:

Format Code	Length	Description
BI	1 to 256 bytes	Unsigned binary
FI	1 to 256 bytes	Signed fixed-point
PD	1 to 32 bytes	Signed packed decimal
ZD	1 to 32 bytes	Signed zoned decimal
CH	1 to 4000 bytes	Character
CSF or FS	1 to 32 bytes	Signed numeric with optional leading floating sign
UFF	1 to 44 bytes	Unsigned free form numeric
SFF	1 to 44 bytes	Signed free form numeric

Note: See Appendix C, “Data format descriptions,” on page 823 for detailed format descriptions.

For a ZD or PD format field:

- F, E, C, A, 8, 6, 4, 2, and 0 are treated as equivalent positive signs. Thus the zoned decimal values F2F3C1, F2F3F1, and 020301 are counted as only one unique value.
- D, B, 9, 7, 5, 3, and 1 are treated as equivalent negative signs. Thus the zoned decimal values F2F3B0, F2F3D0, and 020310 are counted as only one unique value.
- Digits are not checked for validity.

ON(VLEN)
See the discussion of this operand on the DISPLAY statement in “DISPLAY operator” on page 566.

VSAMTYPE(x)
See the discussion of this operand on the COPY statement in “COPY operator” on page 547.

UZERO
See the discussion of this operand on the OCCUR statement on “OCCUR operator” on page 612.

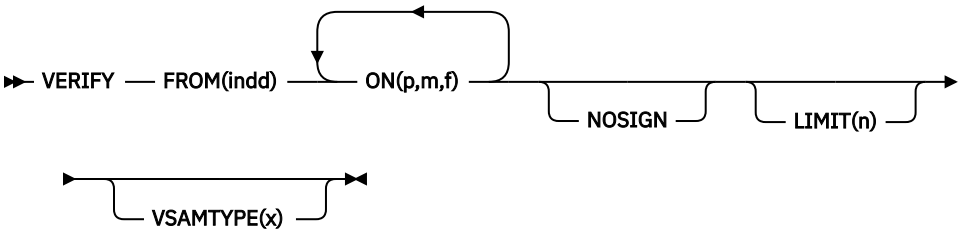
UNIQUE example

```
  UNIQUE FROM(DATAIN)  ON(20,40,CH)
  UNIQUE FROM(DATAIN)  ON(5,3,ZD)
```

The first UNIQUE operator prints a message containing the count of unique character data in positions 20-59 of the DATAIN data set.

The second UNIQUE operator prints a message containing the count of unique zoned decimal values in positions 5-7 of the DATAIN data set.

VERIFY operator



Examines particular decimal fields in a data set and prints a message identifying each invalid value found for each field. From 1 to 10 fields can be specified.

DFSORT is called to copy the indd data set to ICETOOL's E35 user exit. ICETOOL uses its E35 user exit to examine the digits and sign of each value for validity, and for each invalid value found, prints an error message containing the record number and field value (in hexadecimal).

You must not supply your own DFSORT MODS, INREC, or OUTREC statement, because they would override the DFSORT statements passed by ICETOOL for this operator.

Note:

1. Values with invalid decimal digits are also identified for the DISPLAY, OCCUR, RANGE, and STATS operators.
2. The DISPLAY operator can be used to print a report identifying the relative record number, hexadecimal value, and associated fields for each invalid (and valid) decimal value, as shown in Example 9 under "DISPLAY operator" on page 566.

Operand descriptions

The operands described in this section can be specified in any order.

FROM(indd)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

ON(p,m,f)
Specifies the position, length, and format of a decimal field to be used for this operation.

p specifies the first byte of the field relative to the beginning of the input record. p is 1 for the first **data** byte of a fixed-length record and 5 for the first **data** byte of a variable-length record as illustrated in the following (RRRR represents the 4-byte record descriptor word):

```
    Fixed-length record    |    Variable-length record
    | D | A | T | A | ...  |    | R | R | R | R | D | A | T | A | ...
 p=   1   2   3   4        | p=   1   2   3   4   5   6   7   8
```

m specifies the length of the field in bytes. A field must not extend beyond position 32752 or beyond the end of a record. The maximum length for a field depends on its format.

f specifies the format of the field as shown later in this section:

Format Code	Length	Description
PD	1 to 16 bytes	Signed packed decimal
ZD	1 to 31 bytes	Signed zoned decimal

Note: See Appendix C, "Data format descriptions," on page 823 for detailed format descriptions.

A value is considered invalid under one of the following circumstances:

- it contains A-F as a digit (examples: a PD field of 00AF or a ZD field of F2FB)

- it contains 0-9 as a sign and the NOSIGN operand is not specified (examples: a PD field of 3218 or a ZD field of F235).

If the number of bad values reaches the LIMIT for invalid decimal values, ICETOOL terminates the operation. If the LIMIT operand is not specified, a default of 200 is used for the invalid decimal value limit.

NOSIGN
Specifies that the sign of the decimal values is not to be validity checked (overriding the default of checking for 0-9 as invalid signs).

LIMIT(n)
See the discussion of this operand on the DISPLAY statement in "DISPLAY operator" on page 566.

VSAMTYPE(x)
See the discussion of this operand on the COPY statement in "COPY operator" on page 547.

VERIFY example

```
    VERIFY FROM(NEW) ON(22,16,PD) ON(7,9,ZD)
    VERIFY FROM(NEW) ON(22,16,PD) ON(7,9,ZD) NOSIGN LIMIT(10)
```

The first VERIFY operator checks for invalid digits (A-F) and invalid signs (0-9) in the packed decimal values from positions 22-37 and the zoned decimal values from positions 7-15 of the NEW data set. A message is printed identifying each value (if any) that contains an invalid digit or sign. If 200 invalid values are found, the operation is terminated.

The second VERIFY operator checks for invalid digits (A-F) in the packed decimal values from positions 22-37 and the zoned decimal values from positions 7-15 of the NEW data set. A message is printed identifying each value (if any) that contains an invalid digit. If 10 invalid values are found, the operation is terminated.

Note: The DISPLAY operator can be used to print a report identifying the relative record number, hexadecimal value, and associated fields for each invalid (and valid) decimal value, as shown in Example 9 under "DISPLAY operator" on page 566.

Calling ICETOOL from a program

ICETOOL can be called from an assembler program using the LINK, ATTACH, or XCTL system macros. Standard linkage conventions must be used. When ICETOOL finishes processing, it returns to the calling program with register 15 set to the highest operation return code encountered. See "ICETOOL return codes" on page 684 for an explanation of the ICETOOL return codes.

When you call ICETOOL from a program, you have a choice of two different interfaces: the TOOLIN Interface and the Parameter List Interface.

TOOLIN interface

With the TOOLIN Interface, you supply ICETOOL statements in the TOOLIN data set. ICETOOL prints messages in the TOOLMSG data set, but does not return information directly to your program.

To use the TOOLIN interface, set a value of 0 in register 1, or place the address of a 4-byte field containing X'80000000' in register 1, before calling ICETOOL as follows:

```
         ...
         SLR   R1,R1            TOOLIN INTERFACE - METHOD 1
         LINK  EP=ICETOOL       CALL ICETOOL
         ...
         LA    R1,NOPLIST       TOOLIN INTERFACE - METHOD 2
         LINK  EP=ICETOOL       CALL ICETOOL
         ...
NOPLIST  DC    X'80',AL3(0)     TOOLIN INTERFACE INDICATOR
         ...
```

Parameter list interface

The Parameter List Interface allows you to use the information derived by ICETOOL in your program. With this interface, you supply ICETOOL statements in a parameter list. ICETOOL prints messages in the TOOLMSG data set, and puts an operation status indicator and "operation-specific values" in the parameter list for use by your calling program.

Figure 32 on page 679 shows the format of the parameter list used with the Parameter List Interface. Table 92 on page 680 shows the operation-specific values returned to the calling program.

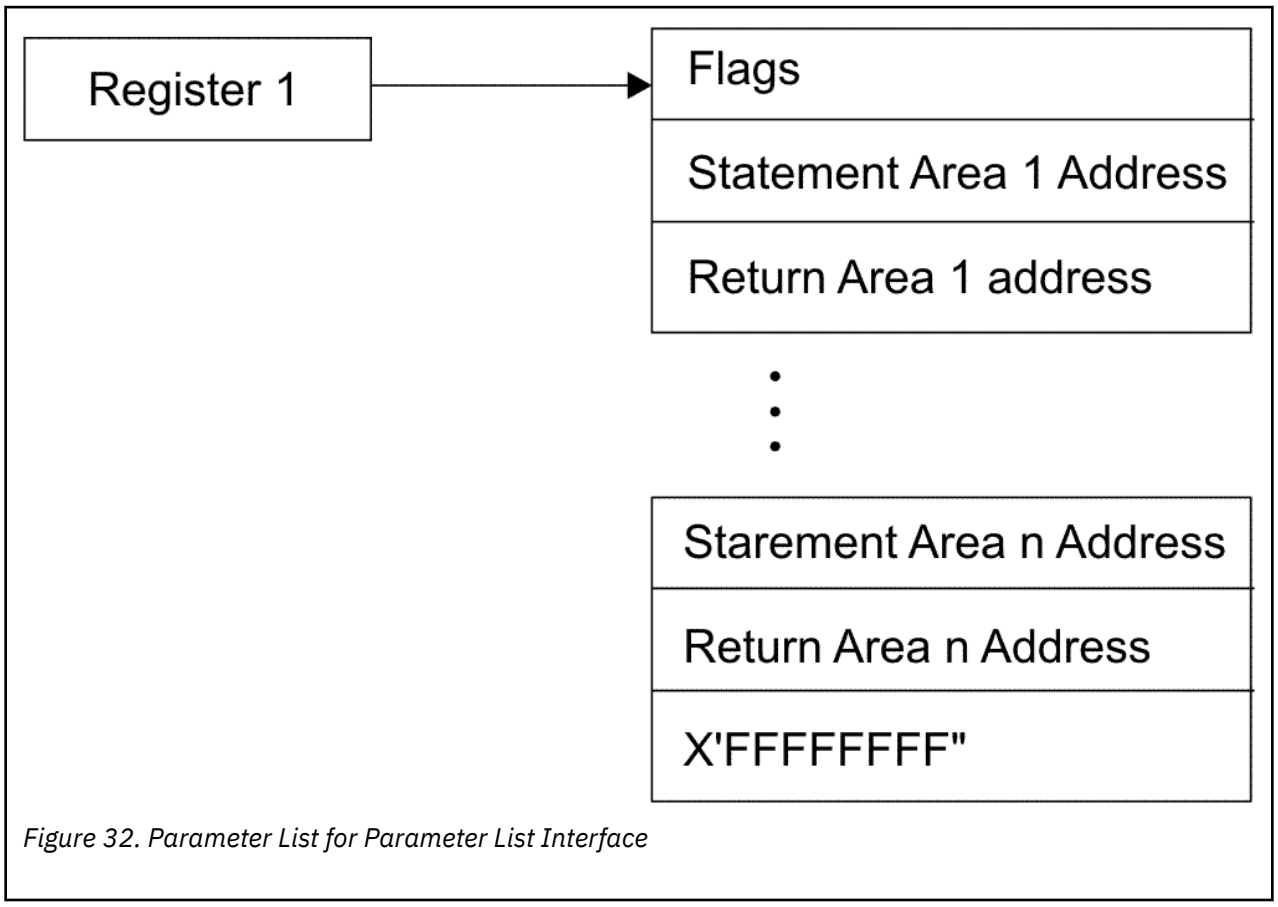


Figure 32. Parameter List for Parameter List Interface

The flags field must be specified. A 4-byte field containing X' FFFFFFFF' must be used to indicate the end of the parameter list. It can be coded after any pair of statement/return addresses.

All addresses in the parameter list must be 31-bit addresses or clean 24-bit addresses (the first 8 bits contain zeros).

Explanation of fields

Flags

Bit 0 = 0:
Use the Parameter List Interface. Process ICETOOL statements from this parameter list and return information to this parameter list. Ignore TOOLIN.

Bit 0 = 1:
Use the TOOLIN Interface. Process ICETOOL statements from TOOLIN. Ignore this parameter list.

Bits 1-31:
Reserved. Must be set to zero to ensure future extendability.

Statement Area Address and Statement Area
Each statement area address gives the location of a statement area that describes an ICETOOL operation to be performed. If the statement area address is 0, ICETOOL ignores this statement

area/return area pair. Otherwise, the statement area address must point to a statement area in the following format:

- A 2-byte length field containing the length of the statement area for this operation. If this field is 0, ICETOOL ignores this statement area/return area pair.
- One or more 80-character ICETOOL statement images in the format described in "ICETOOL statements" on page 546. Each statement area must have **one** operator statement. Comment and blank statements before the operator statement are processed. Comment, blank, and additional operator statements after the end of the first operator statement are ignored.

Return Area Address and Return Area

Each return area address gives the location of a return area in which ICETOOL is to return operation-specific information for the operation described in the corresponding statement area. If the return area address is 0, ICETOOL does not return any information for this operation. Otherwise, the return area address must point to a return area in the following general format:

- A 2-byte length field containing the length of the return area for this operation. If this field is 0, ICETOOL does not return any information for this operation.
- A 1-byte operation status indicator which is set by ICETOOL as follows:

  0 =
  This operation was run and completed with return code 0 or 4. Operation-specific values (see the following bullet) were returned.

  4 =
  This operation was not run (for example, scan mode was in effect) or did not complete with return code 0 or 4. Operation-specific values (see the following bullet) were not returned.
- Operation-specific values. Each count value returned by ICETOOL is an 8-byte packed decimal value with a C for a positive sign or a D for a negative sign. For a STATS operator, each minimum, maximum, average and total value returned by ICETOOL is a 16-byte packed decimal value with a C for a positive sign or a D for a negative sign. If ICETOOL set the operation status to 4, it did not return any values for this operation.

Note: Programs in LPALIB that call ICETOOL must provide return areas that ICETOOL can store into.

The required return area length and the operation-specific values returned for each operator are shown in Table 92 on page 680. If the return area length is less than the length required, ICETOOL issues a message and terminates the operation.

Table 92. Return Area Lengths/Operation-Specific Values

Return Area Lengths/Operation-Specific Values Operator	Return Area Length (Bytes)	Operation-Specific Values Returned
COPY	1	None
COUNT	9	Count of records processed, or 0 if any criteria specified. If SUB(q) or ADD(r) is specified, the modified count is returned.
DATASORT	1	None
DEFAULTS	1	None
DISPLAY	9	Count of records processed
MERGE	1	None
MODE	1	None
OCCUR	17	Count of records processed, count of records resulting from criteria
RANGE	17	Count of records processed, count of values in range

Table 92. Return Area Lengths/Operation-Specific Values (continued)

Return Area Lengths/Operation-Specific Values Operator	Return Area Length (Bytes)	Operation-Specific Values Returned
RESIZE	1	None
SELECT	17	Count of records processed, count of records resulting from criteria
SORT	1	None
SPLICE	17	Count of records processed, count of records resulting from criteria
STATS	64*n+9	Count of records processed, minimum for ON field 1, maximum for ON field 1, average for ON field 1, total for ON field 1, ... minimum for ON field n, maximum for ON field n, average for ON field n, total for ON field n
SUBSET	1	None
UNIQUE	17	Count of records processed, count of unique values
VERIFY	9	Count of records processed

Parameter list interface example

The following example shows a portion of an assembler language program that uses the Parameter List Interface. Figure 33 on page 682 shows the JCL you might use to run the program in this example.

```
DEPTVIEW CSECT
         ...
* SET UP PARAMETER LIST AND CALL ICETOOL
         LA    R1,PARLST        LOAD ADDRESS OF PARAMETER LIST
         LINK  EP=ICETOOL       CALL ICETOOL
         LTR   R15,R15          IF ANY OPERATIONS WERE NOT SUCCESSFUL,
         BNZ   CKSTAT1            DETERMINE WHICH ONE FAILED
* ALL OPERATIONS WERE SUCCESSFUL
*  CHECK EMPLOYEES PER DEPARTMENT AGAINST ACCEPTABLE LEVEL
         CP    RT2AVG1,EMAVGCK    IF AVERAGE IS ACCEPTABLE,
         BNH   CKQUAL               NO MESSAGE IS NEEDED
* ISSUE A MESSAGE SHOWING AVERAGE, MINIMUM, MAXIMUM, AND
* TOTAL NUMBER OF EMPLOYEES PER DEPARTMENT.
         ...
*  CHECK EXPENSES PER DEPARTMENT AGAINST ACCEPTABLE LEVEL
CKQUAL   CP    RT2AVG2,TLAVGCK    IF AVERAGE IS ACCEPTABLE,
         BNH   PCTCALC              NO MESSAGE IS NEEDED
* ISSUE A MESSAGE SHOWING AVERAGE, MINIMUM, MAXIMUM, AND
* TOTAL EXPENSES PER DEPARTMENT.
         ...
*  CALCULATE THE PERCENTAGE OF DEPARTMENTS OVER/UNDER EMPLOYEE LIMIT
PCTCALC  MVC   WORK+2(4),RT3RCDS+4   COPY NUMBER OF DEPARTMENTS
         SP    WORK+2(4),RT3RNG+4(4) SUBTRACT 'NUMBER WITHIN LIMITS' TO
*                                     GET 'NUMBER OVER/UNDER LIMIT'
         CP    WORK+2(4),P0          IF NONE OVER/UNDER LIMIT,
         BE    PCTPRT                 PERCENTAGE IS ZERO
         MP    WORK+2(4),P100        MULTIPLY NUMBER OVER/UNDER BY 100
         DP    WORK(6),RT3RCDS+4(4)  DIVIDE BY NUMBER OF DEPARTMENTS
* ISSUE A MESSAGE SHOWING THE PERCENTAGE OF DEPARTMENTS THAT ARE
* OVER/UNDER EMPLOYEE LIMIT
PCTPRT   UNPK  PCTVAL,WORK(2)     CONVERT AVERAGE TO PRINTABLE EBCDIC
         OI    PCTVAL+2,X'F0'     ENSURE LAST DIGIT IS PRINTABLE
         ...
* ONE OR MORE OPERATIONS FAILED
CKSTAT1  CLI   RT1STAT,0          IF OPERATION 1 WORKED,
         BNE   CKSTAT2              CHECK OPERATION 2
* ISSUE MESSAGE: OPERATION 1 FAILED - CHECK TOOLMSG
         ...
* PARAMETER LIST
PARLST   DC    A(0)               USE PARAMETER LIST INTERFACE
```

```
         DC    A(ST1A)            STATEMENT AREA 1 ADDRESS
         DC    A(RT1A)            RETURN AREA 1 ADDRESS
         DC    A(ST2A)            STATEMENT AREA 2 ADDRESS
         DC    A(RT2A)            RETURN AREA 2 ADDRESS
         DC    A(ST3A)            STATEMENT AREA 3 ADDRESS
         DC    A(RT3A)            RETURN AREA 3 ADDRESS
         DC    F'.*-1'              END OF PARAMETER LIST* OPERATOR STATEMENT AREAS
*   COPY OPERATION
ST1A     DC    AL2(ST1E-ST1)      LENGTH OF STATEMENT AREA 1
ST1      DC    CL80'* CREATE TWO COPIES OF THE DENVER SITE'
         DC    CL80'* DEPARTMENT RECORDS'
         DC    CL80'COPY FROM(IN) TO(OUT1,OUT2) USING(CTL1)'
ST1E     EQU   *
*   STATS OPERATION
ST2A     DC    AL2(ST2E-ST2)      LENGTH OF STATEMENT AREA 2
ST2      DC    CL80'* GET STATISTICS FOR NUMBER OF EMPLOYEES'
         DC    CL80'* AND TRAVEL EXPENSES PER DEPARTMENT'
         DC    CL80'STATS FROM(OUT1) ON(15,2,PD) ON(28,8,ZD)'
ST2E     EQU   *
*   RANGE OPERATION
ST3A     DC    AL2(ST3E-ST3)      LENGTH OF STATEMENT AREA 3
ST3      DC    CL80'* DETERMINE THE NUMBER OF DEPARTMENTS THAT ARE'
         DC    CL80'* WITHIN THE LIMIT FOR NUMBER OF EMPLOYEES'
         DC    CL80'RANGE FROM(OUT1) ON(15,2,PD) -'
         DC    CL80' HIGHER(10) LOWER(21)'
ST3E     EQU   *
* RETURN AREAS
   COPY OPERATION
RT1A     DC    AL2(RT1E-RT1STAT)  LENGTH OF RETURN AREA 1
RT1STAT  DS    C                  OPERATION STATUS
RT1E     EQU   *
*   STATS OPERATION
RT2A     DC    AL2(RT2E-RT2STAT)  LENGTH OF RETURN AREA 2
RT2STAT  DS    C                  OPERATION STATUS
RT2RCDS  DS    PL8                COUNT OF RECORDS PROCESSED
RT2MIN1  DS    PL16               FIELD 1 - MINIMUM VALUE
RT2MAX1  DS    PL16               FIELD 1 - MAXIMUM VALUE
RT2AVG1  DS    PL16               FIELD 1 - AVERAGE VALUE
RT2TOT1  DS    PL16               FIELD 1 - TOTAL VALUE
RT2MIN2  DS    PL16               FIELD 2 - MINIMUM VALUE
RT2MAX2  DS    PL16               FIELD 2 - MAXIMUM VALUE
RT2AVG2  DS    PL16               FIELD 2 - AVERAGE VALUE
RT2TOT2  DS    PL16               FIELD 2 - TOTAL VALUE
RT2E     EQU   *
*   RANGE OPERATION
RT3A     DC    AL2(RT3E-RT3STAT)  LENGTH OF RETURN AREA 3
RT3STAT  DS    C                  OPERATION STATUS
RT3RCDS  DS    PL8                COUNT OF RECORDS PROCESSED
RT3RNG   DS    PL8                COUNT OF VALUES IN RANGE
RT3E     EQU   *
* VARIABLES/CONSTANTS
WORK     DS    PL6                WORKING VARIABLE
P100     DC    P'100'             CONSTANT 100
P0       DC    P'0'               CONSTANT 0
EMAVGCK  DC    P'17'              ACCEPTABLE AVERAGE EMPLOYEE COUNT
TLAVGCK  DC    P'5000'            ACCEPTABLE AVERAGE TRAVEL EXPENSES
PCTVAL   DS    PL3                PERCENTAGE OF DEPARTMENTS THAT ARE
*                                   OVER/UNDER EMPLOYEE LIMIT
         ...
```

```
//EXAMP JOB A402,PROGRAMMER
//INVOKE EXEC PGM=DEPTVIEW,REGION=1024K
//STEPLIB DD DSN=...  Link library containing DEPTVIEW
//TOOLMSG DD SYSOUT=A
//DFSMSG DD SYSOUT=A
//IN DD DSN=ALL.DEPTS,DISP=SHR
//OUT1 DD DSN=ALL.DEPTS.BACKUP1,DISP=OLD
//OUT2 DD DSN=ALL.DEPTS.BACKUP2,DISP=OLD
//CTL1CNTL DD *
* SELECT ONLY THE DENVER SITE DEPARTMENT RECORDS
  INCLUDE COND=(1,12,CH,EQ,C'DENVER')
/*
```

Figure 33. JCL for Parameter List Interface Program Example

ICETOOL notes and restrictions

- Small REGION values can cause storage problems when ICETOOL calls DFSORT. Large REGION values give DFSORT the flexibility to use the storage it needs for best performance. We recommend that you use a REGION value of at least 1024K for ICETOOL.
- Each ICETOOL operation results in a set of ICETOOL messages in the TOOLMSG data set, and a corresponding set of DFSORT messages in the DFSMSG data set. For a particular call to DFSORT, you can relate the sets of messages in the TOOLMSG and DFSMSG data sets by using the unique identifier for that call. Just match the identifier printed in ICETOOL message ICE606I or ICE627I to the same identifier printed in DFSORT message ICE200I. This is particularly important if an ICETOOL operation fails due to an error detected by DFSORT (return code 16).
- Because ICETOOL calls DFSORT, the installation defaults used for DFSORT are those in effect for the appropriate program-invoked environment, that is, the ICEAM2 or ICEAM4 defaults, or the ICETDx defaults activated for ICEAM2 or ICEAM4. The DFSORT installation defaults apply only to DFSORT, not to ICETOOL. For example, installation option MSGCON=ALL causes DFSORT, but not ICETOOL, to write messages to the master console. The one exception is the SDBMSG installation option; the value in effect from ICEAM2 or ICEAM4 is used for ICETOOL's TOOLMSG data set.
- When ICETOOL calls DFSORT, it passes control statements and options appropriate to the specific operations being performed. You should not override the DFSORT control statements or options passed by ICETOOL unless you understand the ramifications of doing so.

  For example, ICETOOL passes the NOABEND option to DFSORT to ensure that ICETOOL will regain control if DFSORT issues an error message. If you specify:

  ```
  //DFSPARM DD *
    DEBUG ABEND
  ```

  you cause DFSORT to abend when it issues an error message, thus preventing ICETOOL from performing subsequent operators.
- Tape work data sets *cannot* be used with ICETOOL.
- An ON field must not include bytes beyond the fixed part of variable length input records. The entire field specified must be present in every input record, otherwise, DFSORT issues message ICE015A, ICE218A, or ICE027A and terminates.
-

Notes on using JOINKEYS with COPY and SORT

- You can use a JOINKEYS application with a COPY or SORT operator of ICETOOL, but not with any of the other ICETOOL operators. Use the JKFROM operand instead of the FROM operand to indicate you are using a JOINKEYS application.
- See Chapter 4, “Using a JOINKEYS application for joining two files,” on page 439 for complete details on JOINKEYS applications.
- If you want to sort the joined records, use a SORT operator with USING(xxxx) in TOOLIN and a SORT control statement along with the appropriate JOINKEYS, JOIN and REFORMAT statements in xxxxCNTL (main task).
- If you want to copy the joined records, use a COPY operator with USING(yyyy) in TOOLIN and the appropriate JOINKEYS, JOIN and REFORMAT statements in yyyyCNTL (main task).
- For multiple JOINKEYS applications within a single ICETOOL step:
  - You can reference different JOINKEYS F1 and F2 input files for the different SORT and/or COPY operators by using F1=ddname and F2=ddname on the various JOINKEYS statements as appropriate
  - You can reference different subtaskn message and control files for the different SORT and/or COPY operators by using TASKID=id on the various JOINKEYS statements as appropriate.

ICETOOL return codes

ICETOOL sets a return code for each operation it performs in STOP or CONTINUE mode and passes back the highest return code it encounters to the operating system or the invoking program.

For successful completion of all operations, ICETOOL passes back a return code of 0 or 4 to the operating system or the invoking program.

For unsuccessful completion due to an unsupported operating system, ICETOOL passes back a return code of 24 to the operating system or invoking program.

For unsuccessful completion of one or more operations, ICETOOL passes back a return code of 8, 12, 16, or 20 to the operating system or the invoking program.

The meanings of the return codes that ICETOOL passes back (in register 15) are:

0

Successful completion. All operations completed successfully.

4

Successful completion. All operations completed successfully. Either:

- RC4 was specified for an ICETOOL COUNT operator and the record count met the specified criteria, or
- DFSORT passed back a return code of 4 for one or more operations. See "DFSORT messages and return codes" on page 25 for details.

8

Unsuccessful completion. RC8 was specified for an ICETOOL COUNT operator and the record count met the specified criteria.

12

Unsuccessful completion. ICETOOL detected one or more errors that prevented it from completing successfully. Messages for these errors were printed in the TOOLMSG data set.

16

Unsuccessful completion. DFSORT detected one or more errors that prevented ICETOOL from completing successfully. Messages for these errors were printed in the DFSMSG data set.

20

Message data set error. The TOOLMSG DD statement was not present or the TOOLMSG data set was not opened.

24

Unsupported operating system. This operating system is not supported by this release of DFSORT.

Chapter 8. Using symbols for fields and constants

Field and constant symbols overview

This chapter describes DFSORT's simple and flexible method for using symbols in DFSORT and ICETOOL statements. You can define and use a symbol for any field, constant, or output column that is recognized in a DFSORT control statement or ICETOOL operator. You can also use symbols for numbers associated with various keywords in DFSORT and ICETOOL statements. This makes it easy to create and reuse collections of symbols (that is, mappings) representing information associated with various record layouts.

In addition, you can obtain and use collections of DFSORT symbols created specifically for records produced by other products (for example, RACF®, DFSMSrmm and DCOLLECT) or by your site. Visit the DFSORT Home Page (www.ibm.com/storage/dfsort) to obtain information about downloading DFSORT symbol mappings for records produced by other products, and examples that use these symbols.

Symbols can increase your productivity by automatically providing the positions, lengths and formats of the fields, the values of the constants, and the positions of the output columns associated with the particular records you are processing with DFSORT or ICETOOL.

You can even use system symbols (for example, &JOBNAME.) and SET and PROC symbols, in your symbol constants.

To use symbols for DFSORT or ICETOOL, you just:

1. Create or obtain the DFSORT symbol data sets you need. Symbol data sets contain symbols that map the fields in your records, and constants used for comparisons, titles, headings and so on. The symbols are specified in DFSORT's simple but flexible SYMNAMES statement format. Symbols can be easily added or modified using an editor, such as ISPF EDIT.
2. Include a SYMNAMES DD statement in your job. SYMNAMES specifies one or more symbol data sets (sequential, partitioned member, DD *) to be used for your DFSORT or ICETOOL application. SYMNAMES can be used to concatenate as many symbol data sets as you like.
3. Use the symbols from SYMNAMES where appropriate in DFSORT control statements or ICETOOL operators. You can mix symbols (for example, Last_Name) with regular fields (for example, p,m,f) and constants (for example, C'string').

DFSORT or ICETOOL will read SYMNAMES and use the symbols it contains to "transform" your statements by performing symbol substitution. DFSORT or ICETOOL will then use the transformed statements as if you had specified them directly.

If your record layout changes, just make a corresponding change to your DFSORT symbol data set. DFSORT will use the new mapping to "transform" your symbols correctly, even if positions change, so you won't have to change your statements. Be sure that your symbol definitions match your record layout before you attempt to use them.

DFSORT example

The example in this section shows the JCL and control statements for a simple DFSORT job that uses symbols.

Let's say you created a symbols data set named MY.SYMBOLS that contains the following SYMNAMES statements:

```
* Fields
First_Name,6,20,CH
Last_Name,*,=,=
Account_Number,53,3,PD
SKIP,2
Balance,*,6,ZD
Type,*,8,CH
```

```
* Constants
Loan,'LOAN'
Check,'CHECKING'
Level1,50000
Level2,-100
```

Here's the JCL and control statements for the example:

```
//EXAMP JOB A402,PROGRAMMER
//RUNIT EXEC PGM=ICEMAN
//SYMNAMES  DD  DSN=MY.SYMBOLS,DISP=SHR
//SYMNOUT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SORTIN DD ...
//SORTOUT DD ...
//SYSIN    DD    *
  INCLUDE COND=((Type,EQ,Loan,AND,Balance,GT,Level1),OR,
                (Type,EQ,Check,AND,Balance,LE,Level2))
  SORT FIELDS=(Last_Name,A,First_Name,A,
               Type,A,Account_Number,D)
/*
```

This example is only meant to give you a quick overview of how symbols can be used. The rest of this chapter will explain all of the details, but here are a few important things to take note of:

- The SYMNAMES DD indicates you want DFSORT or ICETOOL to do symbol processing. The SYMNAMES data set contains the symbols for fields and constants.
- DFSORT or ICETOOL will print your original symbols and the symbol table constructed from them in the SYMNOUT data set, if you specify it. You might want to use SYMNOUT while debugging a set of symbols and then remove it, or you might want to keep SYMNOUT permanently so you can always see your original symbols and the symbol table.
- The simple, yet flexible, format for SYMNAMES statements is:

  ```
  symbol,value remark
  ```

  where value can represent a field (p,m,f or p,m or p), a parsed field (%nn), or a constant (C'string', c'string', 'string', S'string', s'string', X'string', x'string', B'string', b'string', n, +n or -n). Leading blanks are allowed before symbol so indentation can be used. For example, the following SYMNAMES statements could be specified:

  ```
  Div1_Department,8,1,BI      Division 1 Department
    Research,B'0001....'        Research Departments
    Marketing,B'0010....'       Marketing Departments
    Development,B'0100....'     Development Departments
  ```

- Symbols are case-sensitive: Frank, FRANK and frank are three **different** symbols.
- An asterisk (*) can be used to assign the *next position* to p. For example:

  ```
  Symbola,6,20,CH
  Symbolb,*,5,BI
  Symbolc,*,12,ZD
  ```

  is the same as specifying:

  ```
  Symbola,6,20,CH
  Symbolb,26,5,BI
  Symbolc,31,12,ZD
  ```

  By using * for p, you can map consecutive fields in your records without having to compute their actual positions.
- SKIP,n can be used to advance the *next position* by n bytes so it can be used for *. For example:

  ```
  Symbola,6,20,CH
  SKIP,2
  Symbolb,*,5,BI
  ```

is the same as specifying:

```
Symbola,6,20,CH
Symbolb,28,5,BI
```

SKIP,n gives you an easy way to skip unused bytes. Other mapping aids allow you to reset the *next position* (POSITION,q or POSITION,symbol), or align the *next position* on a halfword (ALIGN,H), fullword (ALIGN,F) or doubleword (ALIGN,D).

- An equal sign (=) can be used for p, m or f to assign the previous position, length or format to p, m, or f, respectively. For example:

```
Symbola,6,20,CH
  Symbola1,=,8,=
  Symbola2,*,12,=
Symbold,*,=,ZD
```

is the same as specifying:

```
Symbola,6,20,CH
  Symbola1,6,8,CH
  Symbola2,14,12,CH
Symbold,26,12,ZD
```

By using = and *, you can easily map fields onto other fields.

- Symbols for fields and constants can be specified in any order. However, the use of * and = imposes order dependencies on symbols for fields.
- Comment statements and blank statements are allowed in SYMNAMES.

SYMNAMES DD statement

A SYMNAMES DD statement indicates you want DFSORT or ICETOOL to do symbol processing. It specifies the SYMNAMES data set (SYMNAMES for short), which can consist of one DFSORT symbol data set or many concatenated symbol data sets.

A symbol data set can be a sequential data set, a partitioned member or a DD * data set; all three types can be concatenated together for the SYMNAMES DD. Each symbol data set must contain SYMNAMES statements describing the symbols for fields and constants to be used for the DFSORT or ICETOOL application. Each symbol data set must have the following attributes: RECFM=F or RECFM=FB and LRECL=80.

For best performance, use a large block size, such as the system determined optimum block size, for all DFSORT symbol data sets.

If a SYMNAMES DD statement is not present, or SYMNAMES is empty, symbol processing is not performed.

SYMNOUT DD statement

A SYMNOUT DD statement specifies a data set in which you want DFSORT or ICETOOL to print your original SYMNAMES statements and the symbol table constructed from them. DFSORT or ICETOOL uses RECFM=FBA, LRECL=121 and the specified BLKSIZE for the SYMNOUT data set (SYMNOUT for short).

If the BLKSIZE you specify is not a multiple of 121, or you do not specify the BLKSIZE:

- the system determined optimum block size is used, if supported
- otherwise, BLKSIZE=121 is used.

For best performance, use a large block size, such as the system determined optimum block size, for the SYMNOUT data set.

SYMNAMES statements

Each symbol in SYMNAMES must be described using a SYMNAMES statement. A SYMNAMES statement can be a symbol statement, keyword statement, comment statement or blank statement.

Comment and blank statements

A statement with an asterisk (*) in column 1 is treated as a comment statement. It is printed in SYMNOUT (if specified), but otherwise not processed. A statement with blanks in columns 1 through 80 is treated as a blank statement. It is printed in SYMNOUT (if specified), but otherwise not processed.

Symbol statements

The general format for a symbol statement is:

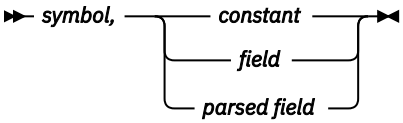
```
symbol,value remark
```

The general coding rules are as follows:

- Columns 1 through 80 are scanned.
- The symbol can start in column 1 or in any column after 1.
- A remark is optional, but if specified, must be separated from the value by at least one blank. A remark is printed in SYMNOUT (if specified), but otherwise not processed.
- A semicolon (;) can be used instead of a comma (,) between the symbol and the value.
- Continuation is not allowed. Each symbol and value must be completely specified on one line.

The specific syntax for symbol statements is:

symbol statements syntax



Symbol: A symbol can be 1 to 50 EBCDIC characters consisting of uppercase letters (A-Z), lowercase letters (a-z), numbers (0-9), the number sign (#), the dollar sign ($), the commercial at sign (@), the underscore(_) and the hyphen(-). The first character of a symbol must not be a number or a hyphen. Symbols are treated as case-sensitive: Frank, FRANK and frank are three **different** symbols.

The following DFSORT/ICETOOL reserved words (uppercase only as shown) are not allowed as symbols: A, AC, ADD, ADDDAYS, ADDMONS, ADDYEARS, ALL, AND, AQ, ASF, ASL, AST, AUF, BI, CH, CLO, COPY, COUNT, COUNT15, CSF, CSL, CST, CTO, D, DATE, DATEDIFF, DATE1, DATE1..., DATE2, DATE2..., DATE3, DATE3..., DATE4, DATE5, DC1, DC2, DC3, DC4, DE1, DE2, DE3, DE4, DIV, DT1, DT2, DT3, D1, D2, E, F, FI, FL, FS, H, HEX, LASTDAYM, LASTDAYQ, LASTDAYW, LASTDAYY, LC, LN, LS, MAX, MC, MIN, MN, MOD, MUL, Mn, Mnn, NEXTDday, NONE, NUM, OL, OR, OT, PAGE, PAGEHEAD, PD, PDC, PDF, PD0, PREVDday, SEQNUM, SFF, SS, SUB, SUBCOUNT, SUBCOUNT15, SUBDAYS, SUBMONS, SUBYEARS, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TIME, TIME1, TIME1P, TIME2, TIME2P, TIME3, TIME3P, TM1, TM2, TM3, TM4, TS, UC, UFF, UN, UTF8, UTF16, UTF32, VALCNT, VLEN, X, Y2x, Y2xx, Y4x, Z, ZD, ZDC, and ZDF, where n is 0-9, day is SUN, MON, TUE, WED, THU, FRI or SAT, and x is any character. Lower case and mixed case forms of these words, such as None and page, can be used as symbols.

POSITION, SKIP and ALIGN (uppercase only) are treated as keywords as discussed in “Keyword statements” on page 697 and thus are not recognized as symbols. However, lowercase and mixed case forms of these words, such as Position and skip, can be used as symbols.

Some examples of valid symbols are: Account_Number, CON12, PHONE#, count, SORT-KEY, and _Invalid.

Some examples of invalid symbols are: 123_Account (starts with a number), COUNT (reserved word), DATE1-20 (reserved word), and -Invalid (starts with a hyphen).

Constant: A constant can be a character string, system symbol string, hexadecimal string, bit string, decimal number, or two-digit year date string.

A symbol for a constant value must be used only where such a constant is allowed and has the desired result. Otherwise, substitution of the constant for the symbol will result in an error message or unintended processing. For example, if the following SYMNAMES statement is specified:

```
SYMB,B'10110001'
```

SYMB can be used in this INCLUDE statement:

```
  INCLUDE COND=(12,1,BI,EQ,SYMB)
```

because a bit string can be compared to a binary field. However, SYMB will result in an error message if used in this INCLUDE statement:

```
  INCLUDE COND=(12,1,CH,EQ,SYMB)
```

because a bit string cannot be compared to a character field.

Make sure the constants that will be substituted for your symbols are appropriate. If in doubt, check the rules for constants given in the description of the relevant operand.

A symbol can represent one of the following types of constants:

- A character string in the format 'xx...x', C'xx...x' or c'xx...x'.

  The value x may be any EBCDIC character. You can specify up to 64 characters for the string. c'xx...x' will be treated like C'xx...x'.

  Note: If you want to use system symbols (for example, &JOBNAME.) in your character strings, use a system symbol string (S'string' or s'string') as described later in this section instead of a character string.

  If you want to include a single apostrophe in the character string, you must specify it as two single apostrophes (each pair of apostrophes counts as two characters towards the 64 character limit for the string). Thus:

  ```
  Required:  O'Neill         Specify:  C'O''Neill'
  ```

  Double-byte data may be used in a character string (each pair of shift-in/shift-out characters and each double-byte character counts as two characters towards the 64 character limit for the string). See “INCLUDE control statement” on page 91 for details on double-byte data.

  Some examples of valid character strings are: '+0.193', c'Title', C'O''Neil', C'J62,J82,M72' and ''.

  Some examples of invalid character strings are: C'AB'' (apostrophes not paired), c'title (ending apostrophe missing) and C'O'NEIL' (one apostrophe after O instead of two).

  You can use C'xx...x' and 'xx...x' interchangeably. 'xx...x' will be substituted for symbols where appropriate even if C'xx...x' is specified in SYMNAMES. Likewise, C'xx...x' will be substituted for symbols where appropriate even if 'xx...x' is specified in SYMNAMES. For example, if these SYMNAMES statements are specified:

  ```
  My_Title,c'My Report'
  My_Heading,C'January'
  DEPT1,'J82'
  DEPT2,c'M72'
  ```

  the ICETOOL operator:

  ```
    DISPLAY TITLE(My_Title) HEADER(My_Heading) ...
  ```

  will be transformed to:

  ```
    DISPLAY TITLE('My Report') HEADER('January') ...
  ```

and the INCLUDE statement:

```
  INCLUDE COND=(5,3,EQ,DEPT1,OR,5,3,EQ,DEPT2),FORMAT=CH
```

will be transformed to:

```
  INCLUDE COND=(5,3,EQ,C'J82',OR,5,3,EQ,C'M72'),FORMAT=CH
```

Although the rules for character strings used as symbols generally follow the rules for INCLUDE/OMIT character strings, keep in mind that the same rules do not apply for character strings in all DFSORT and ICETOOL operands, so use symbols representing character strings appropriately. For example, ICETOOL only allows up to 50 characters for a TITLE string, so TITLE(MYCON) would result in an error message if MYCON is a 64-character string, even though MYCON could be used without error in an INCLUDE statement. As another example, double-byte characters would be recognized in a character string substituted for a symbol in an INCLUDE statement, but would not be recognized in a character string substituted in an OUTREC statement.

- A system symbol string in the format S'original_string' or s'original_string'

  The original_string can contain any combination of EBCDIC characters and system symbols (&SYMBOL or &SYMBOL.) you want to use to form a character string. DFSORT will replace each system symbol in the system symbol string with its substitution text to create a character string in the format C'result_string'.

  For example, you could specify the following symbol statement in SYMNAMES:

```
Rpt_hdr,S'  Job &JOBNAME. was run on Sysplex &SYSPLEX. on &LWDAY'
```

  If you used this symbol statement in a job named TEST2 that you ran on sysplex MAS3 on a Thursday, DFSORT would transform it into the following symbol statement:

```
Rpt_hdr,C'  Job TEST2 was run on Sysplex MAS3 on THU'
```

  &JOBNAME. was replaced with 'TEST2', &SYSPLEX. was replaced with 'MAS3' and &LWDAY was replaced with 'THU'.

  If you used this same symbol statement in a job named BIGTEST that you ran on sysplex MAS2 on a Monday, DFSORT would transform it into the following symbol statement:

```
Rpt_hdr,C'  Job BIGTEST was run on Sysplex MAS2 on MON'
```

  This time &JOBNAME. was replaced with 'BIGTEST', &SYSPLEX. was replaced with 'MAS2' and &LWDAY was replaced with 'MON'.

  You could use the Rpt_hdr symbol in an OUTFIL statement like this:

```
  OUTFIL HEADER2=(Rpt_hdr,5X,'Page ',PAGE=(EDIT=(TTT)),/)
```

  and get the heading you needed based on the setting of the system symbol values where and when the job was run.

  The types of system symbols you can use in a system symbol string are:

  – Dynamic system symbols like &YYMMDD, &LYYMMDD, &HHMMSS, &LHHMMSS, &DAY, &LDAY, &HR, &LHR, &JDAY, &LJDAY, &JOBNAME, &MIN, &LMIN, &MON, &LMON, &SEC, &LSEC, &WDAY, &LWDAY, &YR2, &LYR2, &YR4, and &LYR4.
  – System-defined static system symbols like &SYSCLONE, &SYSNAME, &SYSPLEX, &SYSR1, and &SYSALVL.
  – Installation-defined static system symbols specified by your installation in an IEASYMxx member of SYS1.PARMLIB.

  Note: System symbols must be specified using all uppercase characters. Lowercase characters are not recognized as system symbols. For example, &JOBNAME is recognized as a system symbol, but &jobname and &Jobname are not.

You can use all three types of system symbols separately or in combination within a system symbol string. You can also use all of the system symbol string building facilities such as substring notation, concatenation, embedded blanks, and so on, in a system symbol string. Thus, system symbol strings can be very simple like this one:

```
Sysname,s'&SYSNAME'
```

Or more complex like this one:

```
Dsn1,s'ACCT.&SYSNAME(3:2)..D&LJDAY'
```

For more information on system symbols, see *z/OS MVS Initialization and Tuning Reference*.

Note: JCL symbols and IPCS symbols are not system symbols and will not be recognized or replaced in a system symbol string.

s'original_string' will be treated like S'original_string'.

If you want to include a single apostrophe in the system symbol string, you must specify it as two single apostrophes.

DFSORT uses the ASASYMBM service to transform S'original_string' into C'result_string'. If your system symbol string has errors involving symbol names, substring notation, and so on, ASASYMBM transforms the symbol system string to a character string according to its rules for substitution. For example, if you specify these symbol statements:

```
Ok,S'&YR4(1:2).&SYSNAME.'
Bad,S'&YR4(1.2).&SYSNAM.'
```

where the valid substring notation 1:2 and the known system symbol &SYSNAME. (for example, 'ED53') are used for Ok, but the invalid substring notation 1.2 and the unknown system symbol &SYSNAM. are used for Bad, the resulting symbol statements are:

```
Ok,C'20EDS3'
Bad,C'2006(1.2).&SYSNAM.'
```

See *z/OS MVS Programming: Assembler Services Reference ABE-HSP* for complete details on the ASASYMBM service.

After a symbol statement containing a system symbol string (S'original_string') is transformed into a symbol statement containing the resulting character string (C'result_string'), it will be error checked and processed like any other symbol statement containing a character string. See the description of character string previously in this section for details.

If a SYMNOUT data set is specified, it will show the symbol statement containing the original system symbol string as well as the transformed symbol statement containing the resulting character string. This can be helpful for debugging "errors" or unexpected results in your system symbol strings.

- A hexadecimal string in the format X'yy...yy' or x'yy...yy'.

  The value yy represents any pair of hexadecimal digits. Each hexadecimal digit must be 0-9, A-F or a-f. You can specify up to 32 pairs of hexadecimal digits. x'yy...yy' will be treated like X'yy...yy'. a-f will be treated like A-F.

  Some examples of valid hexadecimal strings are: X'F2C3', x'2fa71e', and X'07'.

  Some examples of invalid hexadecimal strings are: X'F2G301' (G is not a valid hexadecimal digit), x'bf3' (unpaired hexadecimal digits) and X'' (no hexadecimal digits).

- A bit string in the format B'bbbbbbbb...bbbbbbbb' or b'bbbbbbbb...bbbbbbbb'.

  The value bbbbbbbb represents 8 bits that constitute a byte. Each bit must be 1, 0 or . (period). You can specify up to 8 groups of 8 bits. b'bbbbbbbb...bbbbbbbb' will be treated like B'bbbbbbbb...bbbbbbbb'.

  Some examples of valid bit strings are: B'01100100', b'11..00..000..111' and B'11.....'.

Some examples of invalid bit strings are: B'0101' (not a multiple of eight bits), b'00..11....' (not a multiple of eight bits), b'00000002' (2 is not a valid bit) and B'' (no bits).

- A decimal number in the format n, +n or -n. You can specify from 1 to 31 significant digits.

  Some examples of valid decimal numbers are: +270, 270, 000036, +0 and -2000000.

  Some examples of invalid decimal numbers are: ++15 (too many plus signs), 280- (sign in wrong place) and 2.8 (period is not allowed).

- A two-digit year date string in the format Y'string' or y'string'.

  string can be:

  – yy, yyx,yyxx, yyxxx or yyxxxx where y is a hexadecimal year digit (0-9) and x is a hexadecimal non-year digit (0-9).
  – Uppercase, lowercase or mixed case forms of LOW, BLANKS, or HIGH.
  – A two-digit year date string in the format Y'datex', Y'datex'+n or Y'datex'-n.
    - datex can be uppercase, lowercase or mixed case forms of DATE1, DATE2 or DATE3.
    - n is a decimal number representing the number of days for DATE1 and DATE3, or the number of months for DATE2.

  Some examples of valid two-digit year date strings are: Y'99', y'00123', y'date2', Y'DATE1'+20, y'Date3'-1, and Y'Blanks'.

  Some examples of invalid two-digit year date strings are: Y'9', y'AB123', Y'DATE2-3', and Y'blank'.

Field: A field can be specified as p,m,f (position, length and format), p,m (position and length) or p (position only).

A symbol for a field value must be used only where such a field is allowed and has the desired result. Otherwise, substitution of the field for the symbol will result in an error message or unintended processing. For example, if the following SYMNAMES statement is specified:

```
Field1,15,2,CH
```

Field1 can be used in a SORT statement such as:

```
  SORT FIELDS=(Field1,A)
```

because a character field is allowed in a SORT statement. However, Field1 will result in an error message if used in a SUM statement such as:

```
  SUM FIELDS=(Field1)
```

because a character field is not allowed in a SUM statement.

As another example, Field1: can be used in an INREC statement such as:

```
  OUTFIL OVERLAY=(Field1:C'AB')
```

to set the output column to 15, because column notation is allowed in the BUILD, OVERLAY or PUSH operand of an INREC or OUTREC statement and in the BUILD, OVERLAY, PUSH, TRLUPD, HEADERx, or TRAILERx operand of an OUTFIL statement. However, Field1: will result in an error message if used in any other operand, such as WHEN.

Make sure the fields that will be substituted for your symbols are appropriate. If in doubt, check the rules for p, m, f, and c: given in the description of the relevant operand.

You can specify p,m,f for your field symbols and then use them where p,m is required because DFSORT or ICETOOL will substitute just p,m when appropriate. You can specify p,m,f or p,m for your field symbols and then use them where c: is required because DFSORT will substitute just p: when appropriate. For example, if you specify the following in SYMNAMES:

```
First_Field,12,2,BI
Second_Field,18,6,CH
Third_Field,28,5,PD
Fourth_Field,36,3
Fifth_Field,52,4,PD
Max,200000
Outcol2,16
```

These DFSORT control statements:

```
  OMIT COND=(Fifth_Field,GT,Max)
  SORT FIELDS=(First_Field,A,Fourth_Field,A),FORMAT=CH
  SUM FIELDS=(Second_Field,ZD)
  OUTFIL OUTREC=(First_Field:First_Field,
     Outcol2:Third_Field,M11,
     Fourth_Field)
```

will be transformed to:

```
  OMIT COND=(52,4,PD,GT,200000)
  SORT FIELDS=(12,2,A,36,3,A),FORMAT=CH
  SUM FIELDS=(18,6,ZD)
  OUTFIL BUILD=(12:12,2,16:28,5,PD,M11,36,3)
```

Note that DFSORT did the following substitutions:

- OMIT statement: p,m,f for Fifth_Field as required by COND without FORMAT.
- SORT statement: p,m for First_Field and Fourth_Field as required by FIELDS with FORMAT.
- SUM statement: p,m for Second_Field as required for symbol,f (that is, Second_Field,ZD).
- OUTFIL statement: p: for First_Field: and Outcol2: as required by the OUTREC operand for an output column. p,m for First_Field and Fourth_Field as required by the OUTREC operand for an unedited field. p,m,f for Third_Field as required by the OUTREC operand for an edited field (that is, Third_Field,M11).

The general rules for using p, m and f in symbol statements are as follows:

- **p** can be a number, an asterisk (*) or an equal sign (=). A number from 1 to 32752 is allowed in p,m or p,m,f. Because p (position only) cannot be distinguished from the constant n, 1 to 15 significant digits are allowed for p (position only).

  An asterisk (*) can be used to assign the *next position* to p. Each time a symbol for p,m,f or p,m is read, the *next position* is set to p+m. Additionally, the *next position* can be modified by keyword statements (see “Keyword statements” on page 697). When * is specified for p, the *next position* is assigned to p. If the *next position* has not been set when * is used for p (for example, * is used in the first symbol), p is set to 1.

  The symbol table printed in the SYMNOUT data set (if specified) will show you the actual positions assigned when you specify * for p.

  As an example of how * can be used, if you specify the following SYMNAMES statements:

  ```
  Sym1,*,5,ZD
  Con1,27
  Sym2,*,2,BI
  Field1,8,13,CH
  Field2,*,5,PD
  Field3,*,2,FI
  ```

  SYMNOUT will show the following symbol table:

  ```
  Sym1,1,5,ZD
  Con1,27
  Sym2,6,2,BI
  Field1,8,13,CH
  Field2,21,5,PD
  Field3,26,2,FI
  ```

By using * for p, you can map consecutive fields in your records without having to compute their actual positions. You can also map fields added between other fields without having to change the p values for the original or inserted fields. * is also useful for creating mappings of contiguous fields using concatenated symbol data sets. As a simple example, if you specify:

```
//SYMNAMES DD DSN=MY.SYMPDS(RDW),DISP=SHR
//         DD DSN=MY.SYMPDS(SECTION1),DISP=SHR
//         DD DSN=MY.SYMPDS(SECTION2),DISP=SHR
```

and the RDW member contains:

```
RDW,1,4,BI
```

the SECTION1 member contains:

```
Flag_Byte,*,1,BI
  Error1,X'80'
  Error2,X'40'
Count_of_Parts,*,5,ZD
```

and the SECTION2 member contains:

```
New_Parts,*,5,ZD
Old_Parts,*,5,ZD
Variable_Fields,*
```

SYMNOUT will show the following symbol table:

```
RDW,1,4,BI
Flag_Byte,5,1,BI
Error1,X'80'
Error2,X'40'
Count_of_Parts,6,5,ZD
New_Parts,11,5,ZD
Old_Parts,16,5,ZD
Variable_Fields,21
```

You might use these symbols in the following statements:

```
  OPTION COPY
  OUTFIL FNAMES=ERR1,INCLUDE=(Flag_Byte,EQ,Error1),
     OUTREC=(RDW,Count_of_Parts,Variable_Fields)
  OUTFIL FNAMES=ERR2,INCLUDE=(Flag_Byte,EQ,Error2),
     OUTREC=(RDW,New_Parts,Old_Parts,Variable_Fields)
```

An equal sign (=) can be used to assign the *previous position* to p. Each time a symbol for p,m,f or p,m is read, the *previous position* is set to p. Additionally, the *previous position* can be modified by a POSITION keyword statement (see later in this section). When = is specified for p, the *previous position* is assigned to p. If the *previous position* has not been set when = is used for p, an error message is issued.

The symbol table printed in the SYMNOUT data set (if specified) will show you the actual positions assigned when you specify = for p.

As an example of how = can be used for p, if you specify the following SYMNAMES statements:

```
Sym1,5,4,CH
  Sym2,=,2,CH
  Sym3,*,2,CH
```

SYMNOUT will show the following symbol table:

```
Sym1,5,4,CH
Sym2,5,2,CH
Sym3,7,2,CH
```

By using = and * for p, you can easily map fields onto other fields.

Whenever you use = for p, you must ensure that the *previous position* is the one you want. In particular, if you insert a new field symbol with the wrong position before a symbol that uses = for p, you will need to change = to the actual position you want.

- **m** can be an equal sign (=) or a number from 1 to 32752. An equal sign (=) can be used to assign the *previous length* to m. Each time a symbol for p,m,f or p,m is read, the *previous length* is set to m. When = is specified for m, the *previous length* is assigned to m. If the *previous length* has not been set when = is used for m, an error message is issued.

  The symbol table printed in the SYMNOUT data set (if specified) will show you the actual lengths assigned when you specify = for m.

  As an example of how = can be used for m, if you specify the following SYMNAMES statements:

```
Flags1,5,1,BI
  Error1,X'08'
Flags2,15,=,BI
  Error2,X'04'
Flags3,22,=,BI
  Error3,X'23'
```

  SYMNOUT will show the following symbol table:

```
Flags1,5,1,BI
Error1,X'08'
Flags2,15,1,BI
Error2,X'04'
Flags3,22,1,BI
Error3,X'23'
```

  Whenever you use = for m, you must ensure that the *previous length* is the one you want. In particular, if you insert a new field symbol with the wrong length before a symbol that uses = for m, you will need to change = to the actual length you want.

- f can be an equal sign (=) or one of the following formats: AC, AQ, ASL, AST, BI, CH, CLO, CSF, CSL, CST, CTO, DC1, DC2, DC3, DE1, DE2, DE3, DT1, DT2, DT3, D1, D2, FI, FL, FS, LS, OL, OT, PD, PD0, SFF, SS, TC1, TC2, TC3, TC4, TE1, TE2, TE3, TE4, TM1, TM2, TM3, TM4, TS, UFF, Y2B, Y2C, Y2D, Y2DP, Y2P, Y2PP, Y2S, Y2T, Y2TP, Y2U, Y2UP, Y2V, Y2VP, Y2W, Y2WP, Y2X, Y2XP, Y2Y, Y2YP, Y2Z, Y4T, Y4U, Y4V, Y4W, Y4X, Y4Y or ZD.

  You can specify f using uppercase letters (for example, CH), lowercase letters (for example, ch) or mixed case letters (for example, Ch). f specified in any case will be treated like uppercase.

  An equal sign (=) can be used to assign the *previous format* to f. Each time a symbol for p,m,f is read, the *previous format* is set to f. When = is specified for f, the *previous format* is assigned to f. If the *previous format* has not been set when = is used for f, an error message is issued.

  The symbol table printed in the SYMNOUT data set (if specified) will show you the actual formats assigned when you specify = for f.

  As an example of how = can be used for f, if you specify the following SYMNAMES statements:

```
Field1,5,8,CH
Field1a,=,3
Field2,*,12,=
Field3,*,20,=
```

  SYMNOUT will show the following symbol table:

```
Field1,5,8,CH
Field1a,5,3
Field2,8,12,CH
Field3,20,20,CH
```

  Whenever you use = for f, you must ensure that the *previous format* is the one you want. In particular, if you insert a new field symbol with the wrong format before a symbol that uses = for f, you will need to change = to the actual format you want.

Parsed field: A parsed field can be specified as %nnn with nnn as 000-999, %nn with nn as 00-99 or as %n with n as 0-9. %0n will be substituted for %n and %00n. %nn will be substitued for %0nn. A symbol for a parsed field must be used only where such a field is allowed and has the desired result. Otherwise, substitution of %nn for the symbol will result in an error message. For example, if the following SYMNAMES statement is specified:

```
Revenue,%03
```

Revenue can be used in an OUTREC statement such as:

```
  OUTREC PARSE=(Revenue=(STARTAT=C'+',FIXLEN=9)),
         OVERLAY=(31:Revenue,UFF,EDIT=(III,IIT,TTT))
```

because a parsed field is allowed in the PARSE operand and the OVERLAY operand. However, Revenue will result in an error message if used in a SORT statement such as:

```
    SORT FIELDS=(Revenue,UFF,A)
```

because a parsed field is not allowed in a SORT statement.

Make sure the parsed fields that will be substituted for your symbols are appropriate. If in doubt, check the rules for %nn given in the description of the relevant operand.

As an example of how parsed fields can be used, if you specify the following SYMNAMES statements:

```
branch,1,7
tab,X'05'
comma,','
first_name,%00
last_name,%1
date,%02
company,%03
city,%004
state,%305
```

SYMNOUT will show the following symbol table:

```
branch,1,7
tab,X'05'
comma,C','
first_name,%00
last_name,%01
date,%02
company,%03
city,%04
state,%305
```

This DFSORT control statement:

```
  INREC PARSE=(first_name=(ABSPOS=9,FIXLEN=12,ENDBEFR=comma),
            last_name=(FIXLEN=15,ENDBEFR=comma),
            %=(ENDBEFR=tab),
            date=(FIXLEN=10,ENDBEFR=tab),
            company=(FIXLEN=31,ENDBEFR=tab),
            city=(FIXLEN=14,ENDBEFR=tab),
            state=(FIXLEN=12)),
       BUILD=(branch,9:first_name,22:last_name,
              38:date,JFY=(SHIFT=RIGHT),
              49:company,83:city,98:state)
```

will be transformed to:

```
  INREC PARSE=(%00=(ABSPOS=9,FIXLEN=12,ENDBEFR=C','),%01=(FIXLEN=15,ENDB*
               EFR=C','),%=(ENDBEFR=X'05'),%02=(FIXLEN=10,ENDBEFR=X'05'*
               ),%03=(FIXLEN=31,ENDBEFR=X'05'),%04=(FIXLEN=14,ENDBEFR=X*
               '05'),%05=(FIXLEN=12)),BUILD=(1,7,9:%00,22:%01,38:%02,JF*
               Y=(SHIFT=RIGHT),49:%03,83:%04,98:%305)
```

Keyword statements

The general format for a keyword statement is:

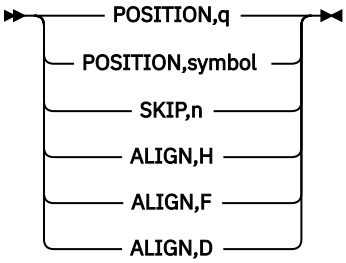
```
keyword,value remark
```

The general coding rules are as follows:

- Columns 1 through 80 are scanned.
- The keyword can start in column 1 or in any column after 1.
- The keyword must be specified in all **uppercase** letters. Otherwise, it will be treated as a symbol.
- A remark is optional, but if specified, must be separated from the value by at least one blank. A remark is printed in SYMNOUT (if specified), but otherwise not processed.
- A semicolon (;) can be used instead of a comma (,) between the keyword and the value.
- Continuation is not allowed. Each keyword and value must be completely specified on one line.

The specific syntax for keyword statements is:

Keyword statements syntax



Keyword statements can help you map the fields in your records by letting you set a starting position, skip unused bytes and align fields on specific boundaries.

- **POSITION,q** can be used to set the *next position* and the *previous position* to q. As discussed under p previously, the *next position* is used when an asterisk (*) is specified for p in a symbol statement, and the *previous position* is used when an equal sign (=) is specified for p in a symbol statement. q can be a number from 1 to 32752. When you use POSITION,q you can use either * or = interchangably for p of the next symbol.

  As an example of how POSITION,q can be used, if you specify the following SYMNAMES statements:

  ```
  POSITION,27
  Account_Balance,*,5,PD
  Account_Id,*,8,CH
  POSITION,84
  New_Balance,=,20
  ```

  SYMNOUT will show the following symbol table:

  ```
  Account_Balance,27,5,PD
  Account_Id,32,8,CH
  New_Balance,84,20
  ```

- **POSITION,symbol** can be used to set the *next position* and the *previous position* to the position established for the indicated symbol. As discussed under p previously, the *next position* is used when an asterisk (*) is specified for p in a symbol statement, and the *previous position* is used when an equal sign (=) is specified for p in a symbol statement. When you use POSITION,symbol you can use either * or = interchangeably for p of the next symbol.

  symbol can be any previously defined field symbol. Thus, POSITION,symbol can be used like the Assembler ORG instruction to map different fields onto the same area.

  As an example of how POSITION,symbol can be used, if you specify the following SYMNAMES statements:

```
Workarea,21,100     Use workarea for volsers
  volser1,=,6,CH
  volser2,*,6,CH
POSITION,Workarea   Reuse workarea for status and dsname
  status,=,1,BI
  dsname,*,44,CH
```

SYMNOUT will show the following symbol table:

```
Workarea,21,100
volser1,21,6,CH
volser2,27,6,CH
status,21,1,BI
dsname,22,44,CH
```

- **SKIP,n** can be used to add n bytes to the *next position*. As discussed under p previously, the *next position* is used when an asterisk (*) is specified for p in a symbol statement. n can be a number from 1 to 32752.

  As an example of how SKIP,n can be used, if you specify the following SYMNAMES statements:

  ```
  Field#1,15,6,FS
   SKIP,4   Unused bytes
  Field#2,*,5,=
   SKIP,2   Unused bytes
  Field#3,*,8,CH
  ```

  SYMNOUT will show the following symbol table:

  ```
  Field#1,15,6,FS
  Field#2,25,5,FS
  Field#3,32,8,CH
  ```

- **ALIGN,H** can be used to align the *next position* on a halfword boundary, that is, 1, 3, 5 and so on. As discussed under p previously, the *next position* is used when an asterisk (*) is specified for p in a symbol statement. ALIGN,h will be treated like ALIGN,H.

  As an example of how ALIGN,H can be used, if you specify the following SYMNAMES statements:

  ```
  A1,7,3,CH
  ALIGN,H
  A2,*,2,BI
  ```

  SYMNOUT will show the following symbol table:

  ```
  A1,7,3,CH
  A2,11,2,BI
  ```

- **ALIGN,F** can be used to align the *next position* on a fullword boundary, that is, 1, 5, 9 and so on. As discussed under p previously, the *next position* is used when an asterisk (*) is specified for p in a symbol statement. ALIGN,f will be treated like ALIGN,F.

  As an example of how ALIGN,F can be used, if you specify the following SYMNAMES statements:

  ```
  B1,7,3,CH
  ALIGN,f
  B2,*,4,BI
  ```

  SYMNOUT will show the following symbol table:

  ```
  B1,7,3,CH
  B2,13,4,BI
  ```

- **ALIGN,D** can be used to align the *next position* on a doubleword boundary, that is, 1, 9, 17 and so on. As discussed under p previously, the *next position* is used when an asterisk (*) is specified for p in a symbol statement. ALIGN,d will be treated like ALIGN,D.

  As an example of how ALIGN,D can be used, if you specify the following SYMNAMES statements:

```
C1,7,3,CH
ALIGN,D
C2,*,8,BI
```

SYMNOUT will show the following symbol table:

```
C1,7,3,CH
C2,17,8,BI
```

Using SYMNOUT to check your SYMNAMES statements

To avoid surprises, it's a good idea to check for errors and incorrect positions, lengths and formats in any SYMNAMES statements you create before you use them in DFSORT or ICETOOL statements.

The following simple job will cause DFSORT to issue messages in SYSOUT for any errors it detects in your SYMNAMES statements, allowing you to correct these errors before proceeding. Once your SYMNAMES statements are free of errors, the job will cause DFSORT to show the Symbol Table in SYMNOUT, allowing you to correct any incorrect positions, lengths or formats for your symbols (for example, those caused by incorrect use of *, =, SKIP, and so on).

```
//CHECK JOB A402,PROGRAMMER
//DOIT EXEC PGM=ICEMAN
//SYMNAMES DD ...        SYMNAMES statements to be checked
//SYMNOUT DD SYSOUT=*
//SORTIN DD *
//SORTOUT DD DUMMY
//SYSIN DD *
  OPTION COPY
/*
```

Once you've "debugged" your SYMNAMES statements, you can use them in DFSORT and ICETOOL statements.

Using symbols in DFSORT statements

You can use symbols in the following DFSORT control statements: INCLUDE, INREC, JOINKEYS, MERGE, OMIT, OPTION, OUTFIL, OUTREC, REFORMAT, SORT and SUM. In general, you can use symbols in these DFSORT statements where you can use constants ('string', C'string', X'string', B'string', n, +n or -n), fields (p,m,f or p,m or p), parsed fields (%nn), and output columns (c:). See the discussion of each control statement in Chapter 3, “Using DFSORT program control statements,” on page 77 for a description of its syntax.

You can use symbols in these control statements in any source (that is, DFSPARM, SYSIN, SORTCNTL, and parameter lists).

When DFSORT transforms these control statements, it removes labels and remarks, and continues statements by placing an asterisk in column 72 and beginning the next line in column 16. DFSORT will list the original control statements as specified (with labels, remarks, comment statements and blank statements) by source, as well as the transformed statements.

Details and examples of the use of symbols for each applicable DFSORT control statement is given later in this section. The examples are meant to illustrate variations in how symbols can be used and how they will be transformed. Therefore, the examples do not necessarily correspond to how symbols would be used in real applications.

The examples use these SYMNAMES statements:

```
C_Field1,6,5,CH
Filler,'*'
Any_Format,12,3
Z_Field1,22,8,ZD
P_Field1,30,4,PD
C_Field2,4,2,ch
SubString,16,3,SS
Start_col,11
Dept_col,55
```

```
LIMIT,+12500
Sysplex,S'&SYSPLEX.'
Depts,'J82,L92,M72'
Code_1,c'86A4Z'
QCON,C'Carrie''s Constant'
Stopper,X'FFFFFF'
Flags,35,1,BI
  Error,B'11010000'
  Empty,B'......01'
  Full,X'FF'
Lookup,52,1,BI
  Entry1,X'05'
  Value1,'Read'
  Entry2,X'20'
  Value2,'Update'
RDW,1,4             Record Descriptor Word
Variable_Fields,451  Variable fields at end of variable-length record
* Constants for report
Div_Title,'Division:  '
  BO_Title,'Branch Office'
BO_Hyphens,'-------------'
 BO_Equals,'============='
  PL_Title,'       Profit/(Loss)'
PL_Hyphens,'--------------------'
 PL_Equals,'===================='
Total,'Total'
Lowest,'Lowest'
* Fields for report
Division,3,10,CH
Branch_Office,16,13,CH
Profit_or_Loss,31,10,ZD
Class_value,%00
Students_value,%01
Class_constant,'CLASS=('
Students_constant,'STUDENTS=('
End_constant,')'
n5,5
n6,6
len80,80
at81,81
len3,3
```

SORT and MERGE

FIELDS operand: You can use symbols where you can use fields (p,m,f and p,m). A symbol for p,m,f results in substitution of p,m if FORMAT=f or symbol,f is specified.

KEYWORD=n operands: You can use a symbol where you can use a number (n) with the SKIPREC and STOPAFT operands.

Example 1

```
  SORT FIELDS=(C_Field1,A,Z_Field1,D,
              C_Field2,ZD,A),EQUALS,SKIPREC=n5
```

will be transformed to:

```
  SORT FIELDS=(6,5,CH,A,22,8,ZD,D,4,2,ZD,A),EQUALS,SKIPREC=5
```

Example 2

```
  MERGE FIELDS=(Any_Format,A,C_Field1,A),FORMAT=CH
```

will be transformed to:

```
  MERGE FIELDS=(12,3,A,6,5,A),FORMAT=CH
```

SUM

FIELDS operand: You can use symbols where you can use fields (p,m,f and p,m). A symbol for p,m,f results in substitution of p,m if FORMAT=f or symbol,f is specified.

Example 1

```
SUM FIELDS=(Z_Field1,C_Field1,ZD)
```

will be transformed to:

```
SUM FIELDS=(22,8,ZD,6,5,ZD)
```

Example 2

```
SUM FORMAT=ZD,FIELDS=(C_Field1,Any_Format)
```

will be transformed to:

```
SUM FORMAT=ZD,FIELDS=(6,5,12,3)
```

INCLUDE and OMIT

COND operand: You can use symbols where you can use fields (p1,m1,f1 and p1,m1 and p2,m2,f2 and p2,m2) and constants (n, +n, -n, C'xx...x', X'yy...yy', Y'yyx...x' and B'bbbbbbbb...bbbbbbbb'). A symbol for p,m,f results in substitution of p,m if FORMAT=f or symbol,f is specified. A symbol for 'string' always results in substitution of C'string'.

Example 1

```
INCLUDE COND=((Z_Field1,GT,LIMIT,AND,Any_Format,CH,EQ,C_Field2),OR,
              (SubString,NE,Depts),OR,
              (Flags,ALL,Error,AND,Flags,NE,Empty))
```

will be transformed to:

```
INCLUDE COND=((22,8,ZD,GT,+12500,AND,12,3,CH,EQ,4,2,CH),OR,(16,3,SS,NE*
              ,C'J82,L92,M72'),OR,(35,1,BI,ALL,B'11010000',AND,35,1,BI*
              ,NE,B'......01'))
```

Example 2

```
 OMIT FORMAT=BI,COND=(C_Field1,EQ,Code_1,OR,
                     Any_Format,EQ,Stopper,OR,
                     Flags,EQ,Full)
```

will be transformed to:

```
OMIT FORMAT=BI,COND=(6,5,EQ,C'86A4Z',OR,12,3,EQ,X'FFFFFF',OR,35,1,EQ,X*
              'FF')
```

Example 3

```
 INCLUDE COND=(25,8,CH,EQ,Sysplex)
```

If the value for the system symbol &SYSPLEX. is 'MAS3', the INCLUDE statement will be transformed to:

```
 INCLUDE COND=(25,8,CH,EQ,C'MAS3')
```

Note: You can use a symbol for Y'DATEx', Y'DATEx'+n or Y'DATEx'-n (where x is 1, 2 or 3) in the COND operand, but you cannot use symbol+n or symbol-n to substitute Y'DATEn'+n or Y'DATEn'-n in the COND operand. For example, if you have the symbols:

```
 YD1,Y'DATE1'
 YD1P5,Y'DATE1'+5
```

The INCLUDE statement:

```
 INCLUDE COND=(1,6,Y2T,EQ,YD1)
```

will be transformed to:

```
INCLUDE COND=(1,6,Y2T,EQ,Y'DATE1')
```

The INCLUDE statement:

```
INCLUDE COND=(1,6,Y2T,EQ,YD1P5)
```

will be transformed to:

```
INCLUDE COND=(1,6,Y2T,EQ,Y'DATE1'+5)
```

The INCLUDE statement:

```
INCLUDE COND=(1,6,Y2T,EQ,YD1+5)
```

will not be transformed because YD1+5 is not valid.

INREC and OUTREC

PARSE operand: You can use symbols where you can use parsed fields (%nn) and constants (C'xx...x' and X'yy...yy').

FIELDS, BUILD, OVERLAY, FINDREP, IFTHEN BUILD, IFTHEN OVERLAY, IFTHEN FINDREP and IFTHEN PUSH operands: You can use symbols where you can use output columns (c:), fields (p,m,f and p,m and p), parsed fields (%nn), non-repeated constants (C'xx...x' and X'yy...yy', but not nC'xx...x' or nX'yy...yy'), and decimal constants (+n and -n, but not n). You cannot use symbols for edit patterns ('pattern').

In the JFY and SQZ suboperands, you can use symbols where you can use constants (C'xx...x' and X'yy...yy').

In the CHANGE and NOMATCH suboperands, you can use symbols where you can use a number (n), fields (q,n), parsed fields (%nn), and constants (C'xx...x', X'yy...yy' and B'bbbbbbbb').

In the RESTART suboperand for SEQNUM, you can use a symbol where you can use a number (n), a field (p,m) or a parsed field (%nn).

A symbol for p or p,m or p,m,f or p,m,Y2x or p,m,Y2xP or p,m,Y4x results in substitution of p: for symbol: (output column).

A symbol for p,m,f results in substitution of p,m,f if (symbol),fo or symbol,TO=fo or symbol,ZDF or symbol,ZDC or symbol,PDC or symbol,PDF is specified, but results in substitution of p,m if symbol,fo is specified (when fo is not ZDF, ZDC, PDC, or PDF) because fo cannot be distinguished from f (except for ZDF, ZDC, PDC, or PDF). For example, if SYM1 is defined as 5,4,ZD, (SYM1),PD is transformed to (5,4,ZD),PD and SYM1,TO=PD is transformed to 5,4,ZD,TO=PD, whereas SYM1,PD is transformed to 5,4,PD. Thus, you should always use (symbol),fo or symbol,TO=fo rather than symbol,fo

A symbol for p,m,f results in substitution of p,m unless symbol,edit or symbol,to or (symbol) is specified or the symbol is part of an arithmetic expression. For example, if SYM1 is defined as 5,4,ZD, SYM1,X is transformed to 5,4,X, whereas SYM1,M12 is transformed to 5,4,ZD,M12 and SYM1,ADD,+1 is transformed to 5,4,ZD,ADD,+1.

A symbol for p,m,Y2x results in substitution of p,m,Y2x if symbol,TO=fo or symbol,todate or symbol,dateop or symbol,ZDF or symbol,ZDC or symbol,PDC or symbol,PDF is specified, but results in substitution of p,m if symbol,fo is specified (when fo is not ZDF, ZDC, PDC, or PDF) because fo cannot be distinguished from f (except for ZDF, ZDC, PDC, or PDF). For example, if SYM1 is defined as 5,4,Y2T, SYM1,TO=PD is transformed to 5,4,Y2T,TO=PD, whereas SYM1,PD is transformed to 5,4,PD. Thus, you should always use symbol,TO=fo rather than symbol,fo when dealing with symbols for p,m,Y2x fields

A symbol for p,m,Y2x results in substitution of p,m,Y2x unless symbol,f or symbol,HEX is specified. A symbol for p,m,Y2xP results in substitution of p,m,Y2xP unless symbol,f or symbol,HEX is specified.

A symbol for p,m,Y4x results in substitution of p,m,Y4x if symbol,TO=fo or symbol,todate or symbol,dateop or symbol,ZDF or symbol,ZDC or symbol,PDC or symbol,PDF is specified, but results in

substitution of p,m if symbol,fo is specified (when fo is not ZDF, ZDC, PDC, or PDF) because fo cannot be distinguished from f (except for ZDF, ZDC, PDC, or PDF). For example, if SYM1 is defined as 5,8,Y4T, SYM1,TO=PD is transformed to 5,8,Y4T,TO=PD, whereas SYM1,PD is transformed to 5,4,PD. Thus, you should always use symbol,TO=fo rather than symbol,fo when dealing with symbols for p,m,Y4x fields.

A symbol for p,m,Y4x results in substitution of p,m,Y4x unless symbol,f or symbol,HEX is specified.

A symbol for 'string' always results in substitution of C'string'.

Example 1

```
    INREC FIELDS=(Start_col:C_Field2,2X,C_Field1,F,Stopper,5C'*',
     Z_Field1,Dept_col:Depts,X,P_Field1,TO=FS,X,Z_Field1,M10,
     Z_Field1,MUL,(P_Field1,SUB,LIMIT),EDIT=(SIIIIIT.TT),SIGNS=(+,-))
```

will be transformed to:

```
  INREC FIELDS=(11:4,2,2X,6,5,F,X'FFFFFF',5C'*',22,8,55:C'J82,L92,M72',X*
              ,30,4,PD,TO=FS,X,22,8,ZD,M10,22,8,ZD,MUL,(30,4,PD,SUB,+1*
              2500),EDIT=(SIIIIIT.TT),SIGNS=(+,-))
```

Example 2

```
  OUTREC FIELDS=(RDW,  ** Record Descriptor Word **
                 Z_Field1,2Z,
                 3C'Symbol cannot be used for a repeated constant',
                 Code_1,Flags,
                 Variable_Fields)  ** Variable part of input record
```

will be transformed to:

```
  OUTREC FIELDS=(1,4,22,8,2Z,3C'Symbol cannot be used for a repeated con*
               stant',C'86A4Z',35,1,451)
```

Example 3

```
  INREC PARSE=(Class_value=(STARTAFT=Class_constant,
               ENDBEFR=End_constant,FIXLEN=8),
        Students_value=(STARTAFT=Students_constant,
               ENDBEFR=End_constant,FIXLEN=n5)),
    BUILD=(Class_value,3X,Students_value,UFF,M11,LENGTH=n5)
```

will be transformed to:

```
  INREC PARSE=(%00=(STARTAFT=C'CLASS=(',ENDBEFR=C')',FIXLEN=8),%01=(STAR*
               TAFT=C'STUDENTS=(',ENDBEFR=C')',FIXLEN=5)),BUILD=(%00,3X*
               ,%01,UFF,M11,LENGTH=5)
```

IFTHEN WHEN=(logexp), IFTHEN BEGIN=(logexp) and IFTHEN END=(logexp) operands: You can use symbols where you can use fields (p1,m1,f1 and p1,m1 and p2,m2,f2 and p2,m2) and constants (n, +n, -n, C'xx...x', X'yy...yy', Y'yyx...x' and B'bbbbbbbb...bbbbbbbb'). A symbol for p,m,f results in substitution of p,m if symbol,f is specified. A symbol for 'string' always results in substitution of C'string'.

Example 1

```
  INREC IFTHEN=(WHEN=(Lookup,EQ,Entry1),OVERLAY=(75:Value1)),
        IFTHEN=(WHEN=(Lookup,EQ,Entry2),OVERLAY=(75:Value2))
```

will be transformed to:

```
  INREC IFTHEN=(WHEN=(52,1,BI,EQ,X'05'),OVERLAY=(75:C'Read')),IFTHEN=(WH*
               EN=(52,1,BI,EQ,X'20'),OVERLAY=(75:C'Update'))
```

Note: You can use a symbol for Y'DATEx', Y'DATEx'+n or Y'DATEx'-n (where x is 1, 2 or 3) in the WHEN, BEGIN or END operand, but you cannot use symbol+n or symbol-n to substitute Y'DATEn'+n or Y'DATEn'-n in the WHEN, BEGIN or END operand. See the discussion of "INCLUDE and OMIT" for further details.

KEYWORD=n operands: You can use a symbol where you can use a number (n) with the IFOUTLEN, RECORDS, ABSPOS, SUBPOS, ADDPOS, FIXLEN, REPEAT, STARTPOS, ENDPOS, MAXLEN, DO, ID, SEQ, START, INCR and LENGTH operands.

Example 1

```
OUTREC IFOUTLEN=len80,
  IFTHEN=(WHEN=GROUP,RECORDS=n5,PUSH=(at81:ID=n6))
```

will be transformed to:

```
OUTREC IFOUTLEN=80,IFTHEN=(WHEN=GROUP,RECORDS=5,PUSH=(81:ID=6))
```

OUTFIL

INCLUDE, OMIT, IFTRAIL TRLID=(logexp), IFTHEN WHEN=(logexp), IFTHEN BEGIN=(logexp) and IFTHEN END=(logexp) operands: You can use symbols where you can use fields (p1,m1,f1 and p1,m1 and p2,m2,f2 and p2,m2) and constants (n, +n, -n, C'xx...x', X'yy...yy', Y'yyx...x' and B'bbbbbbbb...bbbbbbbb'). A symbol for p,m,f results in substitution of p,m if symbol,f is specified. A symbol for 'string' always results in substitution of C'string'.

Note: You can use a symbol for Y'DATEx', Y'DATEx'+n or Y'DATEx'-n (where x is 1, 2 or 3) in the INCLUDE, OMIT, TRLID, WHEN, BEGIN or END operand, but you cannot use symbol+n or symbol-n to substitute Y'DATEn'+n or Y'DATEn'-n in the INCLUDE, OMIT, TRLID, WHEN, BEGIN or END operand. See the discussion of "INCLUDE and OMIT" for further details.

PARSE operand: You can use symbols where you can use parsed fields (%nn) and constants (C'xx...x' and X'yy...yy').

OUTREC, BUILD, OVERLAY, FINDREP, IFTRAIL TRLUPD, IFTHEN BUILD, IFTHEN OVERLAY, IFTHEN FINDREP and IFTHEN PUSH operands: You can use symbols where you can use output columns (c:), fields (p,m,f and p,m and p), parsed fields (%nn), non-repeated constants (C'xx...x' and X'yy...yy', but not nC'xx...x' or nX'yy...yy'), and decimal constants (+n and -n, but not n). You cannot use symbols for edit patterns ('pattern').

In the JFY and SQZ suboperands, you can use symbols where you can use constants (C'xx...x' and X'yy...yy').

In the CHANGE and NOMATCH suboperands, you can use symbols where you can use fields (q,n), parsed fields (%nn), and constants (C'xx...x', X'yy...yy' and B'bbbbbbbb').

In the RESTART suboperand for SEQNUM, you can use a symbol where you can use a field (p,m) or a parsed field (%nn).

In the IFTHEN KEYBEGIN operand, you can use a symbol where you can use a field (p,m).

A symbol for p or p,m or p,m,f or p,m,Y2x or p,m,Y2xP results in substitution of p: for symbol: (output column).

A symbol for p,m,f results in substitution of p,m,f if (symbol),fo or symbol,TO=fo or symbol,ZDF or symbol,ZDC or symbol,PDC or symbol,PDF is specified, but results in substitution of p,m if symbol,fo is specified (when fo is not ZDF or ZDC ZDF, ZDC, PDC, or PDF) because fo cannot be distinguished from f (except for ZDF, ZDC, PDC, and PDF). For example, if SYM1 is defined as 5,4,ZD, (SYM1),PD is transformed to (5,4,ZD),PD and SYM1,TO=PD is transformed to 5,4,ZD,TO=PD, whereas SYM1,PD is transformed to 5,4,PD. Thus, you should always use (symbol),fo or symbol,TO=fo rather than symbol,fo.

A symbol for p,m,f results in substitution of p,m unless symbol,edit or symbol,to or (symbol) is specified or the symbol is part of an arithmetic expression. For example, if SYM1 is defined as 5,4,ZD, SYM1,X is transformed to 5,4,X, whereas SYM1,M12 is transformed to 5,4,ZD,M12 and SYM1,ADD,+1 is transformed to 5,4,ZD,ADD,+1.

A symbol for p,m,Y2x results in substitution of p,m,Y2x if symbol,TO=fo or symbol,todate or symbol, dateop or symbol,ZDF or symbol,ZDC or symbol,PDC or symbol,PDF is specified, but results in substitution of p,m if symbol,fo is specified (when fo is not ZDF, ZDC, PDC, or PDF) because fo cannot

be distinguished from f (except for ZDF, ZDC, PDC, and PDF). For example, if SYM1 is defined as 5,4,Y2T, SYM1,TO=PD is transformed to 5,4,Y2T,TO=PD, whereas SYM1,PD is transformed to 5,4,PD. Thus, you should always use symbol,TO=fo rather than symbol,fo when dealing with symbols for p,m,Y2x fields

A symbol for p,m,Y2x results in substitution of p,m,Y2x unless symbol,f or symbol,HEX is specified. A symbol for p,m,Y2xP results in substitution of p,m,Y2xP unless symbol,f or symbol,HEX is specified.

A symbol for p,m,Y4x results in substitution of p,m,Y4x if symbol,TO=fo or symbol,todate or symbol, dateop or symbol,ZDF or symbol,ZDC or symbol,PDC or symbol,PDF is specified, but results in substitution of p,m if symbol,fo is specified (when fo is not ZDF, ZDC, PDC, or PDF) because fo cannot be distinguished from f (except for ZDF, ZDC, PDC, or PDF). For example, if SYM1 is defined as 5,8,Y4T, SYM1,TO=PD is transformed to 5,8,Y4T,TO=PD, whereas SYM1,PD is transformed to 5,4,PD. Thus, you should always use symbol,TO=fo rather than symbol,fo when dealing with symbols for p,m,Y4x fields.

A symbol for p,m,Y4x results in substitution of p,m,Y4x unless symbol,f or symbol,HEX is specified.

A symbol for 'string' always results in substitution of C'string'.

VLFILL operand: You can use symbols where you can use constants (C'x' and X'yy'). A symbol for 'string' always results in substitution of C'string'.

VLTRIM operand: You can use symbols where you can use constants (C'x' and X'yy'). A symbol for 'string' always results in substitution of C'string'.

VLTRAIL operand: You can use symbols where you can use constants (C'xx...x' and X'yy...yy'). A symbol for 'string' always results in substitution of C'string').

HEADER1 and HEADER2 operands: You can use symbols where you can use output columns (c:), fields (p,m) and non-repeated constants ('xx...x', C'xx...x', and X'yy...yy', but not n'xx...x', nC'xx...x', or nX'yy...yy'). A symbol for p or p,m or p,m,f results in substitution of p: for symbol: (output column).A symbol for p,m,f always results in substitution of p,m. A symbol for 'string' always results in substitution of C'string'.

TRAILER1 and TRAILER2 operands: Outside of the suboperands TOTAL, TOT, MIN, MAX, AVG, SUBTOTAL, SUBTOT, SUB, SUBMIN, SUBMAX and SUBAVG: You can use symbols where you can use output columns (c:), fields (p,m) and non-repeated constants ('xx...x', C'xx...x' and X'yy...yy', but not n'xx...x', nC'xx...x' or nX'yy...yy'). A symbol for p or p,m or p,m,f results in substitution of p: for symbol: (output column). A symbol for p,m,f always results in substitution of p,m. A symbol for 'string' always results in substitution of C'string'.

Inside of the suboperands TOTAL, TOT, MIN, MAX, AVG, SUBTOTAL, SUBTOT, SUB, SUBMIN, SUBMAX and SUBAVG: You can use symbols where you can use fields (p,m,f). A symbol for p,m,f results in substitution of p,m,f if symbol,TO=fo or symbol,ZDF or symbol,ZDC or symbol,PDC or symbol,PDF is specified, but results in substitution of p,m if symbol,fo is specified (when fo is not ZDF, ZDC, PDC, or PDF) because fo cannot be distinguished from f (except for ZDF, ZDC, PDC, and PDF). For example, if SYM1 is defined as 5,4,ZD, MIN=(SYM1,TO=PD) is transformed to MIN=(5,4,ZD,TO=PD) whereas MIN=(SYM1,PD) is transformed to MIN=(5,4,PD). Thus, you should always use symbol,TO=fo rather than symbol,fo.

SECTIONS operand: The "HEADER1 and HEADER2 operands" discussion shown previously also applies to the HEADER3 suboperand of SECTIONS. The "TRAILER1 and TRAILER2 operands" discussion also applies to the TRAILER3 suboperand of SECTIONS.

Outside of the HEADER3 and TRAILER3 suboperands, you can use symbols where you can use fields (p,m). A symbol for p,m,f always results in substitution of p,m.

KEYWORD=n operands: You can use a symbol where you can use a number (n) with the IFOUTLEN, RECORDS, ABSPOS, SUBPOS, ADDPOS, FIXLEN, REPEAT, STARTPOS, ENDPOS, MAXLEN, DO, ID, SEQ, START, INCR, LENGTH, STARTREC, ENDREC, SAMPLE, ACCEPT, SPLITBY, SPLIT1R and LINES operands.

Example 1

```
 OUTFIL FNAMES=OUT1,
       INCLUDE=(Z_Field1,GT,LIMIT,AND,Any_Format,CH,EQ,C_Field2),
       OUTREC=(Any_Format:P_Field1,M0,2X,Any_Format,BI,LENGTH=len3,2X,
               QCON,2X,
               C_Field2,HEX,2X,Z_Field1,EDIT=('I III IIT.T'),2X,
*  Lookup Table
```

```
               Lookup,CHANGE=(n6,Entry1,Value1,Entry2,Value2),
                    NOMATCH=(Lookup))
```

will be transformed to:

```
OUTFIL FNAMES=OUT1,INCLUDE=(22,8,ZD,GT,+12500,AND,12,3,CH,EQ,4,2,CH),O*
              UTREC=(12:30,4,PD,M0,2X,12,3,BI,LENGTH=3,2X,C'Carrie''s *
              Constant',2X,4,2,HEX,2X,22,8,ZD,EDIT=('I III IIT.T'),2X,*
              52,1,CHANGE=(6,X'05',C'Read',X'20',C'Update'),NOMATCH=(5*
              2,1))
```

Example 2

```
OUTFIL FNAMES=REPORT,
   OUTREC=(6:Branch_Office,24:Profit_or_Loss,M5,LENGTH=20,75:X),
   SECTIONS=(Division,SKIP=P,
      HEADER3=(2:Div_Title,Division,5X,'Page:',&PAGE,2/,
                  6:BO_Title,24:PL_Title,/,
                  6:BO_Hyphens,24:PL_Hyphens),
     TRAILER3=(6:BO_Equals,24:PL_Equals,/,
                  6:Total,24:TOTAL=(Profit_or_Loss,M5,LENGTH=20),/,
                  6:Lowest,24:MIN=(Profit_or_Loss,M5,LENGTH=20)))
```

will be transformed to:

```
OUTFIL FNAMES=REPORT,OUTREC=(6:16,13,24:31,10,ZD,M5,LENGTH=20,75:X),SE*
              CTIONS=(3,10,SKIP=P,HEADER3=(2:C'Division:  ',3,10,5X,'P*
              age:',&PAGE,2/,6:C'Branch Office',24:C'       Profit/(Lo*
              ss)',/,6:C'-------------',24:C'--------------------'),TR*
              AILER3=(6:C'=============',24:C'====================',/,*
              6:C'Total',24:TOTAL=(31,10,ZD,M5,LENGTH=20),/,6:C'Lowest*
              ',24:MIN=(31,10,ZD,M5,LENGTH=20)))
```

Example 3

```
 OUTFIL IFTHEN=(WHEN=INIT,
         PARSE=(Class_value=(STARTAFT=Class_constant,
              ENDBEFR=End_constant,FIXLEN=8),
          Students_value=(STARTAFT=Students_constant,
              ENDBEFR=End_constant,FIXLEN=n5)),
          BUILD=(1:Class_value)),
        IFTHEN=(WHEN=(1,8,SS,EQ,C'Biology ,Algebra ,Geometry'),
          BUILD=(C'There are ',Students_value,UFF,M10,X,
               Class_value,C' Students')),
        IFTHEN=(WHEN=NONE,
          BUILD=(C'*Not relevant*'))
```

will be transformed to:

```
 OUTFIL IFTHEN=(WHEN=INIT,PARSE=(%00=(STARTAFT=C'CLASS=(',ENDBEFR=C')',*
               FIXLEN=8),%01=(STARTAFT=C'STUDENTS=(',ENDBEFR=C')',FIXLE*
               N=5)),BUILD=(1:%00)),IFTHEN=(WHEN=(1,8,SS,EQ,C'Biology ,*
               Algebra ,Geometry'),BUILD=(C'There are ',%01,UFF,M10,X,%*
               00,C' Students')),IFTHEN=(WHEN=NONE,BUILD=(C'*Not releva*
               nt*'))
```

JOINKEYS

FIELDS operand: You can use symbols where you can use fields (p,m). A symbol for p,m,f results in substitution of p,m.

INCLUDE and OMIT operands: You can use symbols where you can use fields (p1,m1,f1 and p1,m1 and p2,m2,f2 and p2,m2) and constants (n, +n, -n, C'xx...x', X'yy...yy', Y'yyx...x' and B'bbbbbbbb...bbbbbbbb'). A symbol for p,m,f results in substitution of p,m if symbol,f is specified. A symbol for 'string' always results in substitution of C'string'.

Note: You can use a symbol for Y'DATEx', Y'DATEx'+n or Y'DATEx'-n (where x is 1, 2 or 3) in the INCLUDE or OMIT operand, but you cannot use symbol+n or symbol-n to substitute Y'DATEn'+n or Y'DATEn'-n in the INCLUDE or OMIT operand. See the discussion of "INCLUDE and OMIT" for further details.

STOPAFT=n operand: You can use a symbol where you can use a number (n) with the STOPAFT operand.

Example 1

```
  JOINKEYS FILE=F1,FIELDS=(Division,A),
    INCLUDE=(Substring,EQ,Depts)
```

will be transformed to:

```
JOINKEYS FILE=F1,FIELDS=(3,10,A),INCLUDE=(16,3,SS,EQ,C'J82,L92,M72')
```

REFORMAT

FIELDS operand: You can use symbols where you can use fields (p,m and p). A symbol for p,m,f results in substitution of p,m.

FILL operand: You can use symbols where you can use constants (C'x' and X'yy'). A symbol for 'string' always results in substitution of C'string'.

Example 1

```
  REFORMAT FIELDS=(F1:RDW,Profit_or_Loss,F2:Any_Format,
      F1:Variable_Fields),FILL=filler
```

will be transformed to:

```
REFORMAT FIELDS=(F1:1,4,31,10,F2:12,3,F1:451),FILL=C'*'
```

OPTION

KEYWORD=n operands: You can use a symbol where you can use a number (n) with the SKIPREC and STOPAFT operands.

Example 1

```
  OPTION SKIPREC=n5,MAINSIZE=MAX,STOPAFT=n6
```

will be transformed to:

```
OPTION SKIPREC=5,MAINSIZE=MAX,STOPAFT=6
```

Using symbols in ICETOOL operators

You can use symbols in the following ICETOOL operators: COUNT, DATASORT, DISPLAY, OCCUR, RANGE, SELECT, SPLICE, STATS, SUBSET, UNIQUE, and VERIFY. In general, you can use symbols in these ICETOOL operators where you can use constants ('string', n, +n or -n) and fields (p,m,f or p,m). See the discussion of each operator in Chapter 7, “Using ICETOOL,” on page 537 for a description of its syntax.

ICETOOL reads the SYMNAMES data set once and uses it for all operators and DFSORT control statements for the run. You can use symbols in operators from the TOOLIN data set or your parameter list. You can also use symbols in DFSORT control statements in xxxxCNTL data sets or in the DFSPARM data set (see “Using symbols in DFSORT statements” on page 699 for details).

ICETOOL will list the original operator statements as well as the transformed operator statements.

Details of the use of symbols for each applicable ICETOOL operator is given later in this section followed by a complete ICETOOL example. The example is meant to illustrate variations in how symbols can be used and how they will be transformed. Therefore, the example does not necessarily correspond to how symbols would be used in real applications.

COUNT

HIGHER, LOWER, EQUAL and NOTEQUAL operands: You can use symbols where you can use constants (x, y, v, and w).

SUB and ADD operands: You can use symbols where you can use constants (q and r).

TEXT operand: You can use symbols where you can use constants ('string'). A symbol for C'string' always results in substitution of 'string'.

DATASORT

HEADER, FIRST, TRAILER and LAST operands: You can use symbols where you can use constants (u and v).

DISPLAY

ON operand: You can use symbols where you can use fields (p,m,f and p,m). A symbol for p,m,f results in substitution of p,m if symbol,f or symbol,HEX is specified.

BREAK operand: You can use symbols where you can use fields (p,m,f and p,m). A symbol for p,m,f results in substitution of p,m if symbol,f is specified.

TITLE, HEADER, TOTAL, MAXIMUM, MINIMUM, AVERAGE, COUNT, BTITLE, BTOTAL, BMAXIMUM, BMINIMUM, BAVERAGE and BCOUNT operands: You can use symbols where you can use constants ('string'). A symbol for C'string' always results in substitution of 'string'.

OCCUR

ON operand: You can use symbols where you can use fields (p,m,f and p,m). A symbol for p,m,f results in substitution of p,m if symbol,f or symbol,HEX is specified.

TITLE and HEADER operands: You can use symbols where you can use constants ('string'). A symbol for C'string' always results in substitution of 'string'.

HIGHER, LOWER and EQUAL operands: You can use symbols where you can use constants (x, y and v).

RANGE

ON operand: You can use symbols where you can use fields (p,m,f and p,m). A symbol for p,m,f results in substitution of p,m if symbol,f is specified.

HIGHER, LOWER, EQUAL and NOTEQUAL operands: You can use symbols where you can use constants (x, y, v and w).

SELECT

ON operand: You can use symbols where you can use fields (p,m,f and p,m). A symbol for p,m,f results in substitution of p,m if symbol,f is specified.

FIRST, FIRSTDUP, HIGHER, LOWER and EQUAL operands: You can use symbols where you can use constants (u, v, w, x, and y).

SPLICE

ON operand: You can use symbols where you can use fields (p,m,f and p,m). A symbol for p,m,f results in substitution of p,m if symbol,f is specified.

WITH operand: You can use symbols where you can use fields (p,m). A symbol for p,m,f results in substitution of p,m.

STATS, UNIQUE and VERIFY

ON operand: You can use symbols where you can use fields (p,m,f and p,m). A symbol for p,m,f results in substitution of p,m if symbol,f is specified.

SUBSET

HEADER, FIRST, RRN, TRAILER and LAST operands: You can use symbols where you can use constants (q, r, u and w).

ICETOOL Example

```
//TOOLSYM JOB A402,PROGRAMMER
//DOIT  EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//SYMNOUT DD SYSOUT=*
//SYMNAMES  DD  *
Rdw,1,4,BI
Account_Code,12,1,CH
Dept_Code,*,=,=
Customer_Name,*,20,CH
SKIP,2
Customer_Balance,*,10,ZD
Customer_Flags,*,1,BI
* Department Codes
Research,'R'
Marketing,'M'
* Balance Cutoffs
Cancel,+10000       100.00
Gift,+1000000    10,000.00
Stop_Check,-500      -5.00
* Headings and Titles
Title,S'Customer Report for &LWDAY.'
Head1,'Customer Name'
Head2,'Customer Balance'
Head3,'Customer Flags'
/*
//IN DD DSN=MY.CUSTOMER.INPUT,DISP=SHR
//OUT DD DSN=&&O,UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE),
// DISP=(,PASS)
//LIST1 DD SYSOUT=*
//TOOLIN DD *
  RANGE FROM(IN) ON(Customer_Balance) LOWER(Stop_Check)
  SORT FROM(IN) TO(OUT) USING(CTL1)
  DISPLAY FROM(OUT) LIST(LIST1) BLANK  WIDTH(133) -
    TITLE(Title) DATE(4MD/) PAGE -
    HEADER(Head1) ON(Customer_Name) -
    HEADER(Head2) ON(Customer_Balance,C1) -
    HEADER(Head3) ON(Customer_Flags,HEX)
/*
//CTL1CNTL DD *
  SORT FIELDS=(Customer_Balance,D,Customer_Name,A)
  INCLUDE COND=((Dept_Code,EQ,Research,OR,Dept_Code,EQ,Marketing),
    AND,Customer_Balance,GT,Gift)
/*
```

If the value for the system symbol &LWDAY. is 'FRI', SYMNOUT will show the following:

```
------- ORIGINAL STATEMENTS FROM SYMNAMES -------
Rdw,1,4,BI
Account_Code,12,1,CH
Dept_Code,*,=,=
Customer_Name,*,20,CH
SKIP,2
Customer_Balance,*,10,ZD
Customer_Flags,*,1,BI
* Department Codes
Research,'R'
Marketing,'M'
* Balance Cutoffs
Cancel,+10000       100.00
Gift,+1000000    10,000.00
Stop_Check,-500      -5.00
* Headings and Titles
Title,S'Customer Report for &LWDAY.'
Head1,'Customer Name'
Head2,'Customer Balance'
Head3,'Customer Flags'

------------------ SYMBOL TABLE -----------------
Rdw,1,4,BI
```

```
Account_Code,12,1,CH
Dept_Code,13,1,CH
Customer_Name,14,20,CH
Customer_Balance,36,10,ZD
Customer_Flags,46,1,BI
Research,C'R'
Marketing,C'M'
Cancel,+10000
Gift,+1000000
Stop_Check,-500
Title,C'Customer Report for FRI'
Head1,C'Customer Name'
Head2,C'Customer Balance'
Head3,C'Customer Flags'
```

The ICETOOL operators will be transformed to:

```
RANGE FROM(IN) ON(36,10,ZD) LOWER(-500)

  SORT FROM(IN) TO(OUT) USING(CTL1)

DISPLAY FROM(OUT) LIST(LIST1) BLANK WIDTH(133)-
TITLE('Customer Report for FRI') DATE(4MD/) PAGE-
HEADER('Customer Name') ON(14,20,CH)-
HEADER('Customer Balance') ON(36,10,ZD,C1)-
HEADER('Customer Flags') ON(46,1,HEX)
```

The DFSORT control statements in CTL1CNTL will be transformed to:

```
 SORT FIELDS=(36,10,ZD,D,14,20,CH,A)
 INCLUDE COND=((13,1,CH,EQ,C'R',OR,13,1,CH,EQ,C'M'),AND,36,10,ZD,GT,+10*
               00000)
```

Using SET and PROC symbols in DFSORT and ICETOOL statements

For jobs that directly invoke DFSORT (PGM=SORT or PGM=ICEMAN) or ICETOOL (PGM=ICETOOL), you can construct DFSORT symbols that incorporate JCL PROC or SET symbols as well as text and system symbols. You can then use those constructed DFSORT symbols in DFSORT and ICETOOL control statements in the same way you use regular DFSORT symbols.

As a simple example, if you wanted to use JCL SET symbols named XDSN and YDSN in a DFSORT statement, you could code the following:

```
// SET XDSN='X.DATA',YDSN='Y.DATA'
//S1 EXEC PGM=SORT,PARM='MSGDDN=MYOUT,JP1"&XDSN",JP2"&YDSN",LIST'
//MYOUT DD SYSOUT=*
...
//SYSIN DD *
  OPTION COPY
  OMIT COND=(1,44,CH,EQ,JP1,OR,1,44,CH,EQ,JP2)
/*
```

For each JPn"string" parameter found in EXEC PARM, a DFSORT symbol in the following form is constructed, and treated as if it was specified in the SYMNAMES data set (even if a SYMNAMES data set is not actually present):

```
JPn,S'string'
```

Up to ten symbols, JP0-JP9, can be set up this way.

For the previous example, the following DFSORT symbols are constructed and treated as if they were specified in SYMNAMES:

```
JP1,S'X.DATA'
JP2,S'Y.DATA'
```

These symbols are placed in the Symbol Table as:

```
JP1,C'X.DATA'
JP2,C'Y.DATA'
```

When JP1 and JP2 are used in the OMIT statement, the statement is transformed to:

```
 OMIT COND=(1,44,CH,EQ,C'X.DATA',OR,1,44,CH,EQ,C'Y.DATA')
```

Here is another example using ICETOOL:

```
// SET VLSR1='339001'
// SET VLSR2='339002'
//S2 EXEC PGM=ICETOOL,
// PARM='JP0"Report for disks &VLSR1 and &VLSR2 on &WDAY"'
...
//TOOLIN DD *
DISPLAY TITLE(JP0) -
  FROM(IN) LIST(RPT) ON(5,4,CH)
```

The following DFSORT symbol is constructed:

```
JP0,S'Report for disks 339001 and 339002 on &WDAY'
```

If the job is run on Friday, 'FRI' is substituted for the &WDAY system symbol and the following is placed in the Symbol Table:

```
JP0,C'Report for disks 339001 and 339002 on FRI'
```

When JP0 is used in the DISPLAY statement, the statement is transformed to:

```
DISPLAY TITLE('Report for disks 339001 and 339002 on FRI')-
FROM(IN) LIST(OUT) ON(1,5,CH)
```

Note: For a JOINKEYS application, you can use JCL SET and PROC symbols (JPn) in DFSORT control statements for the main task, but you cannot use JCL SET and PROC symbols (JPn) in DFSORT control statements for the subtasks.

Using JPn parameters in EXEC PARM for DFSORT

```
//stepname EXEC PGM=SORT,PARM='option<,option>...'
```

option can be JPn"string" or one of the other valid DFSORT options (for example, VLSCMP or DSA=8). JPn"string" options are used to construct DFSORT symbols. Other DFSORT options are passed to DFSORT as EXEC PARM options in the normal way.

PGM=ICEMAN or one of the other aliases can be used instead of PGM=SORT.

Normal system JCL rules for the EXEC PARM operand apply. In addition:

- JPn"string" parameters can only be used when DFSORT is invoked directly (for example, with PGM=SORT), not when DFSORT is invoked from a program (for example, with LINK EP=SORT).
- If the length of the EXEC PARM options is greater than the JCL limit of 100 bytes, JPn"string" options will not be processed.

Using JPn parameters in EXEC PARM for ICETOOL

```
//stepname EXEC PGM=ICETOOL,PARM='option<,option>...'
```

option can be JPn"string". JPn"string" options are used to construct DFSORT symbols. Other options are ignored by ICETOOL.

Normal system JCL rules for the EXEC PARM operand apply. In addition:

- JPn"string" parameters can only be used when ICETOOL is invoked directly (for example, with PGM=ICETOOL), not when ICETOOL is invoked from a program (for example, with LINK EP=ICETOOL).

- If the length of the EXEC PARM options is greater than the JCL limit of 100 bytes, JPn"string" options will not be processed.

Description of JPn"string"

Only JP0 through JP9 can be used for a symbol name where 'JP' must be uppercase EBCDIC (X'D1D7') and n must be '0'-'9' (X'F0'-X'F9').

A quote must appear before and after the string. A quote must not appear within the string (it would be interpreted as the ending quote). PARM='...,JPn"string' (a missing ending quote) will be interpreted as PARM='...,JPn"string"'.

If the keyword does not appear as JPn"string", it will not be used as a JPn symbol. For example, JP1"ABC" would be used as a JP1 symbol, but JP1="ABC", JP1("ABC"), JP1=("ABC") or jp1"ABC" would not.

Any or all of the following can appear in the string:

- any EBCDIC character (except quote). If you want to include a single apostrophe in the character string, you must specify it as two single apostrophes
- a JCL SET or PROC symbol (&SYMBOL). Normal system rules for SET and PROC symbols apply.
- a system symbol (for example, &JOBNAME, &SYSPLEX, etc). Normal system rules for system symbols apply.

The symbol will be constructed as:

```
JPn,S'string'
```

and must conform to the rules for a system symbol string as documented earlier in this Chapter.

If you specify:

```
//SYMNOUT DD SYSOUT=*
```

the JPn symbols will be listed along with any other DFSORT symbols you specify in SYMNAMES.

If you specify a SYMNAMES data set, the JPn symbols will be processed **before** the first SYMNAMES symbol.

Note: Here is an example of one suggested way of specifying multiple options in EXEC PARM:

```
//  SET X1='ANY-STRING'
//  SET X2='ANOTHER-STRING'
//S1 EXEC PGM=SORT,
//  PARM=('RESALL=12K',
//   'VLLONG',
//   'JP1"&X1"',
//   'JP2"&X2"')
```

Example 1

MYPROC procedure

```
//MYPROC PROC DAY=
//STEP01 EXEC PGM=SORT,PARM='JP1"&DAY"'
//SYSOUT  DD SYSOUT=*
//SYMNOUT DD SYSOUT=*
//SORTIN DD DSN=MYCNTL.PDS(SEL),DISP=SHR
//SORTOUT DD SYSOUT=*
//SYSIN DD DSN=MYCNTL.PDS(OVLY),DISP=SHR
//  PEND
```

Execution of MYPROC

```
//S1 EXEC MYPROC,DAY=2
```

Record in SEL member

```
SELECT * FROM MYTABLE WHERE TM_RECEIPT >= (CURRENT DATE - ? DAYS)
```

Records in OVLY member

```
  OPTION COPY
  INREC OVERLAY=(59:JP1)
```

This example illustrates how you can use a JCL PROC symbol in a DFSORT control statement.

The JCL PROC symbol is DAY which will be set to a 1 character numeric value (2 in this example), and used to overlay the number of days in the SELECT statement (in the SEL member).

PARM='JP1"&DAY"' is used to create a DFSORT symbol named JP1 containing a string with the DAY value.

The SYMNOUT listing will show the following:

```
------ SYMNAMES STATEMENTS FROM EXEC PARM -------
JP1,S'2'

------------------ SYMBOL TABLE -----------------
JP1,C'2'
```

The DFSORT INREC statement in the OVLY member uses JP1. It will be transformed to:

```
  INREC OVERLAY=(59:C'2')
```

and used to change the SELECT statement to the following in SORTOUT:

```
SELECT * FROM MYTABLE WHERE TM_RECEIPT >= (CURRENT DATE - 2 DAYS)
```

Example 2

```
//  SET JOBNM='FRANK2'
//S1    EXEC  PGM=ICETOOL,
//  PARM=('JP1"&JOBNM"',
//        'JP2"&JOBNM STEP"',
//        'JP3"&JOBNM EXCPS"')
//TOOLMSG DD SYSOUT=*
//DFSMSG  DD SYSOUT=*
//SYMNAMES DD *
JOBN,5,8,CH
STEPN,*,8,CH
EXCPS,*,5,ZD
//SYMNOUT DD SYSOUT=*
//IN DD *
    FRANK2  S1      00008
    FRANK2  S2      00123
    FRANK2  S3      00023
    FRANK3  S1      00016
    FRANK3  S2      00152
/*
//T1 DD DSN=&&T1,UNIT=SYSDA,SPACE=(CYL,(5,5)),DISP=(,PASS)
//RPT DD SYSOUT=*
//TOOLIN DD *
COPY FROM(IN) TO(T1) USING(CTL1)
DISPLAY FROM(T1) LIST(RPT) BLANK -
 HEADER(JP2) ON(STEPN) -
 HEADER(JP3) ON(EXCPS)
/*
//CTL1CNTL DD *
  INCLUDE COND=(JOBN,EQ,JP1)
/*
```

This example illustrates how you can use a JCL SET symbol in ICETOOL and DFSORT control statements.

The JCL SET symbol is JOBNM which is set to a constant ('FRANK2' for this example).

```
//  PARM=('JP1"&JOBNM"',
//        'JP2"&JOBNM STEP"',
//        'JP3"&JOBNM EXCPS"')
```

is used to create DFSORT symbols named JP1, JP2 and JP3 containing strings including the JOBNM value. ICETOOL treats these JPn symbols as if they were specified before the Symbols in SYMNAMES.

The SYMNOUT listing will show the following:

```
------ SYMNAMES STATEMENTS FROM EXEC PARM -------
JP1,S'FRANK2'
JP2,S'FRANK2 STEP'
JP3,S'FRANK2 EXCPS'

------- ORIGINAL STATEMENTS FROM SYMNAMES -------
JOBN,5,8,CH
STEPN,*,8,CH
EXCPS,*,5,ZD

------------------ SYMBOL TABLE -----------------
JP1,C'FRANK2'
JP2,C'FRANK2 STEP'
JP3,C'FRANK2 EXCPS'
JOBN,5,8,CH
STEPN,13,8,CH
EXCPS,21,5,ZD
```

The DFSORT INCLUDE statement in CTL1CNTL uses the JOBN and JP1 symbols. It will be transformed to:

```
 INCLUDE COND=(5,8,CH,EQ,C'FRANK2')
```

The ICETOOL DISPLAY statement uses the JP2, JP3, STEPN and EXCPS symbols. It will be transformed to:

```
DISPLAY FROM(T1) LIST(RPT) BLANK-
HEADER('FRANK2 STEP') ON(13,8,CH)-
HEADER('FRANK2 EXCPS') ON(21,5,ZD)
```

RPT will contain the following report:

```
FRANK2 STEP   FRANK2 EXCPS
-----------   ------------
S1                       8
S2                     123
S3                      23
```

Notes for symbols

- EFS programs cannot be used with symbol processing.
- DFSORT or ICETOOL scans each SYMNAMES statement for errors, and prints an error message for the first error detected. A marker ($) is printed directly below the SYMNAMES statement near the error, if appropriate.

  Scanning stops at the first error, and then continues with the next SYMNAMES statement. However, once an error is detected, positions generated by using an asterisk (*) for p or POSITION,symbol in subsequent statements will not be checked for errors. DFSORT and ICETOOL terminate after all SYMNAMES statements are scanned if an error is detected in any statement.
- If DFSORT or ICETOOL detects an error in a control statement or operator statement during the substitution phase (that is, while attempting to substitute values for symbols), it may either:
  - print the original statement in error followed by a $ marker (if appropriate) and an error message, continue the substitution phase with the next statement and terminate when the substitution phase is complete, or
  - stop performing substitution for the statement in error, continue with the next statement and let processing after the substitution phase handle the error. It is possible in this case for a symbol, rather than a substituted value, to appear in a transformed statement.

- If the substitution phase is successful, DFSORT and ICETOOL will substitute values for symbols wherever symbols are allowed. Substituted values which are invalid for a particular statement or operand will be detected after the substitution phase. This makes it easier to determine the cause of the error. For example, if SYMNAMES contains:

```
Sym1,5,4,ZD
Con1,'1234'
Con2,1234
```

  the statement:

```
 INCLUDE COND=(Sym1,EQ,Con1)
```

  will be transformed to the following during the substitution phase:

```
 INCLUDE COND=(5,4,ZD,EQ,C'1234')
```

  An ICE114A message with a $ marker under C'1234' will then be issued for the statement because a ZD field cannot be compared to a character constant. In this example, the error could be fixed by using Con2 (a decimal constant) in the statement instead of Con1 or by redefining Con1 as a decimal constant.
- If you use a temporary or permanent message data set, it is best to specify a disposition of MOD to ensure you see all messages and control statements in the message data set. In particular, if you use symbols processing and do not use MOD, you will not see the original control statements unless Blockset is selected.
- If you rearrange your records in any way (for example, using E15, E35, INREC, OUTREC or OUTFIL) and want to use symbols for the rearranged records, be sure to use symbols that map to the new positions of your fields. For example, if you use a SYMNAMES data set with the following statements:

```
Field1,1,5,ZD
Field2,*,6,ZD
Field3,*,3,ZD
Field4,*,4,ZD
```

  for this INREC statement:

```
 INREC FIELDS=(Field2,Field4)
```

  the resulting records will only contain Field2 and Field4. If you want to use symbols for the rearranged records (for example, in a SORT statement), you will need to use a SYMNAMES data set with symbols that map to the rearranged records, such as:

```
New_Field2,1,6,ZD
New_Field4,*,4,ZD
```

  If you use unique symbols for the rearranged fields, as in the previous example, you can concatenate the old and new symbol data sets together and use the old and new symbols where appropriate, as in this example:

```
 INREC FIELDS=(Field2,Field4)
 SORT FIELDS=(New_Field2,A,New_Field4,A)
```

Chapter 9. Using extended function support

Using EFS

Like the user exits described in Chapter 5, “Using your own user exit routines,” on page 467, the DFSORT Extended Function Support (EFS) interface is a means by which you can pass run-time control to an EFS program you write yourself. An EFS program is essential if you want to process double-byte character sets (such as Japanese characters) with DFSORT.

To process Japanese data types with DFSORT, you can use the IBM Double Byte Character Set Ordering Support Program (DBCS Ordering), Licensed Program 5665-360, Release 2.0, or you can use locale processing with the appropriate locale.

Using an EFS program and EFS program exit routines, you can:

- Sort or merge user-defined data types (such as double-byte character sets) with user-defined collating sequences
- Include or omit records based on the user-defined data types
- Provide user-written messages to DFSORT for printing to the message data set
- Examine, alter, or ignore control statements or EXEC PARM options prior to processing by DFSORT.

The EFS program can also perform routine tasks, such as opening and initializing data sets, terminating DFSORT, and closing data sets.

You can write your EFS program in any language that uses standard register and linkage conventions, and can:

- Pass a parameter list and a record (if you provide the EFS01 and EFS02 exit routines in the EFS program) in register 1
- Pass a return code in general register 15.

Note:

1. DFSORT does not support EFS programs for Conventional merge or tape work data set sort applications.
2. VLSHRT is not allowed if EFS processing is in effect and an EFS01 or EFS02 exit routine is provided by the EFS program.
3. If you use locale processing for SORT, MERGE, INCLUDE, or OMIT fields, you must not use an EFS program. DFSORT's locale processing may eliminate the need for an EFS program. See “OPTION control statement” on page 173 for information related to locale processing.
4. If you use symbol processing, you must not use an EFS program.

The DFSORT target library, SICEUSER, contains a mapping macro called ICEDEFS, which provides a separate Assembler DSECT for the EFS parameter list.

Addressing and residence mode of the EFS program

You can design the EFS program to reside and run above or below 16MB virtual. Residency and addressing mode can be any valid combination of 24-bit, 31-bit, or ANY. If your EFS program is designed to reside and run below 16MB virtual, the EFS program must determine the proper return mode.

How EFS works

The EFS interface consists of a variable-length parameter list used to communicate between DFSORT and your EFS program. DFSORT activates the EFS program you write at specific points during run-time, and communicates information back and forth across the interface as your EFS program runs.

You can activate your EFS program with the installation or run-time option EFS=name (name is the name of your EFS program):

See Appendix B, “Specification/override of DFSORT options,” on page 805 for override information. Figure 34 on page 718 illustrates the role of the EFS interface in linking DFSORT's processing capabilities to the EFS program you write.

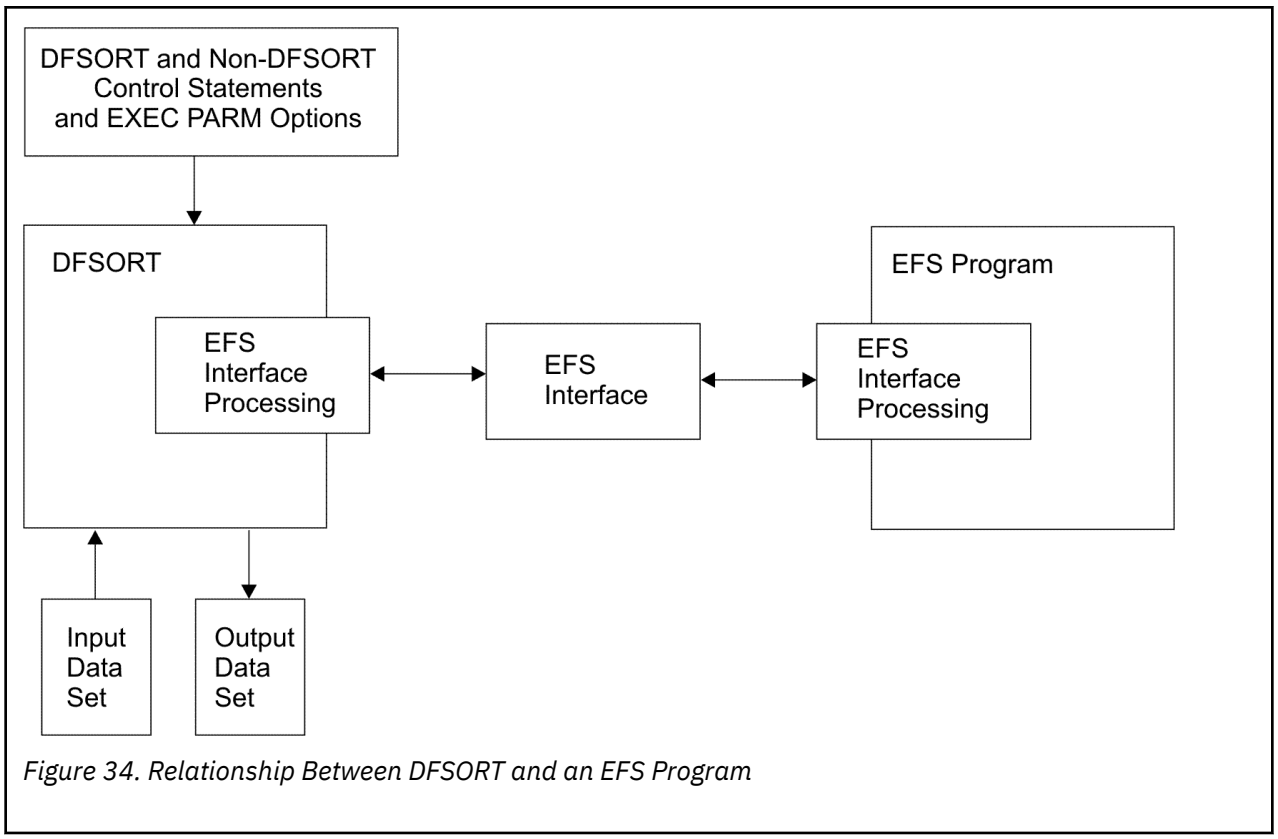


Figure 34. Relationship Between DFSORT and an EFS Program

DFSORT program phases

A DFSORT program phase is a large DFSORT component designed to perform a specific task such as writing the output file. An EFS program is called at various points during run-time of DFSORT program phases in performing the variety of tasks capable with an EFS program. When the termination phase is completed, DFSORT returns control to the operating system or invoking program.

EFS processing can be invoked during the initialization, input, and termination phases of DFSORT. DFSORT always calls the EFS program during the initialization phase.

During the input phase, DFSORT reads input records, and performs any INCLUDE or OMIT statement logic on the records. If the EFS program generates exit routines (EFS01 and EFS02), DFSORT calls them during the input phase.

During the termination phase, DFSORT closes data sets, releases storage, and returns control to the calling program or system. DFSORT always calls the EFS program from the termination phase.

DFSORT calls to your EFS program

DFSORT makes five functional calls (Major Calls 1 through 5) at various phases to transfer information across the EFS interface, between DFSORT and your EFS program. DFSORT can make multiple calls at Major Calls 2 and 3. Refer to Figure 35 on page 719 and Figure 36 on page 720 as you read this section for illustrations of the relationships between program phases and calls during run-time.

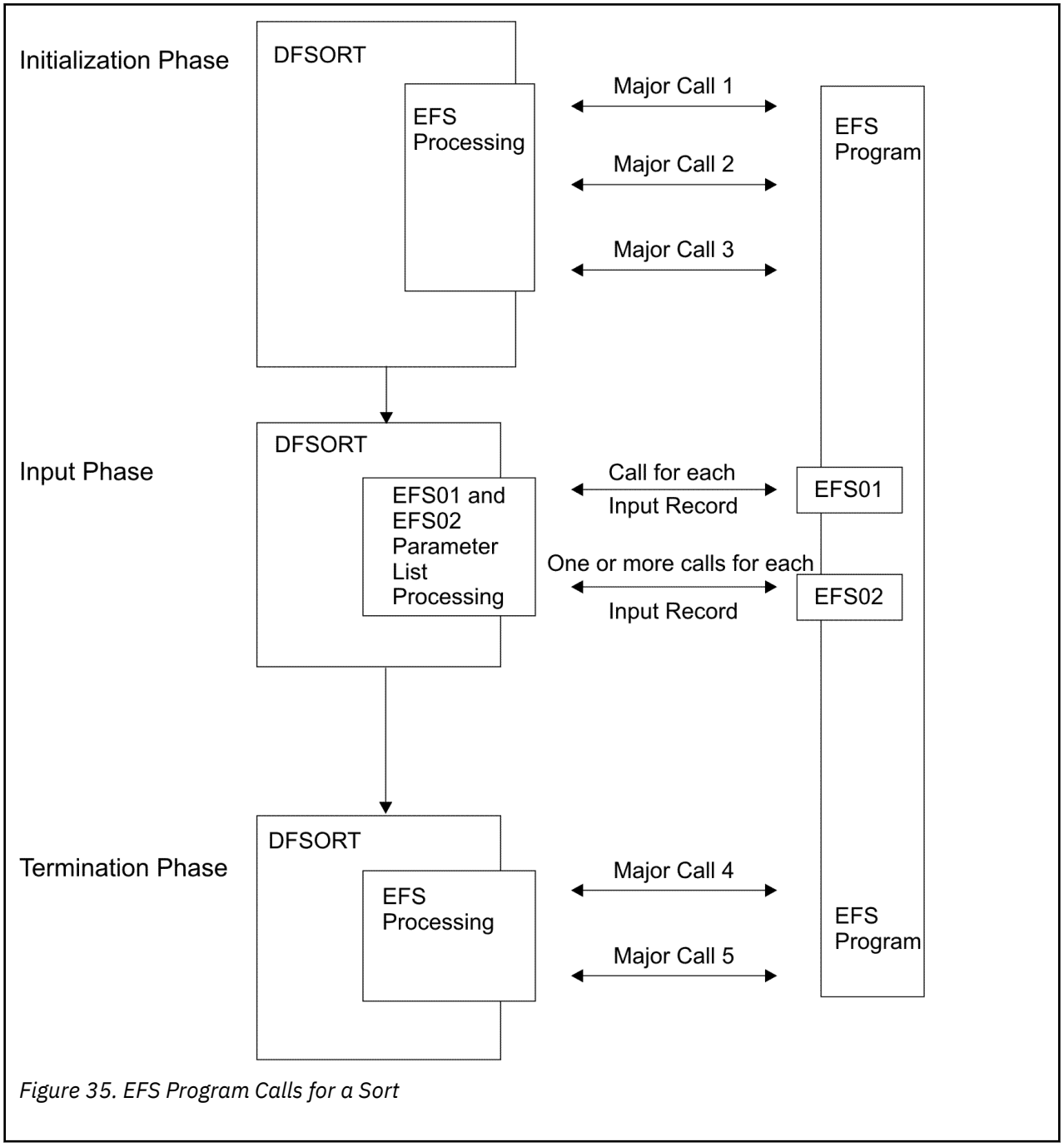


Figure 35. EFS Program Calls for a Sort

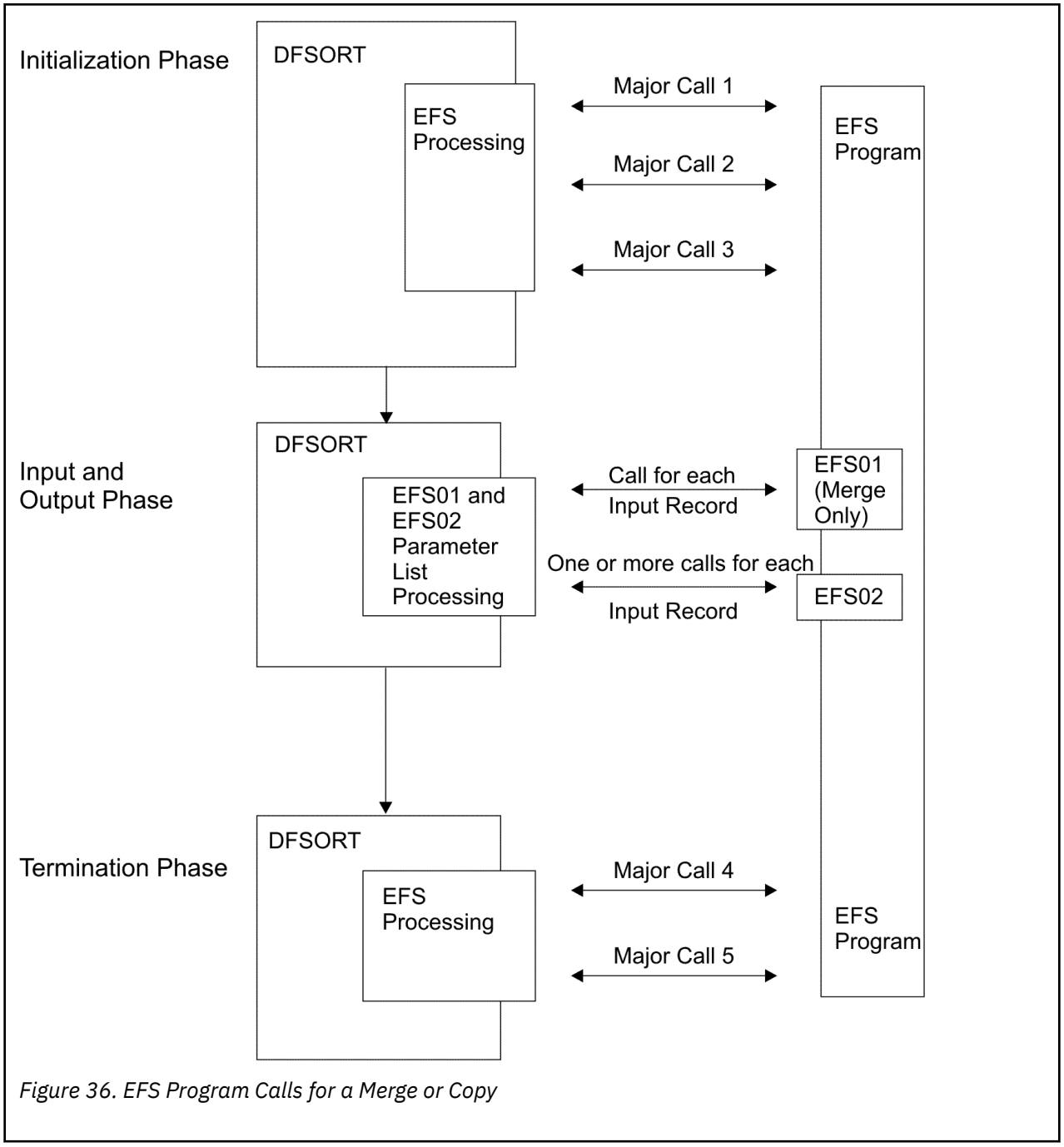


Figure 36. EFS Program Calls for a Merge or Copy

Initialization phase

DFSORT runs Major Calls 1 through 3 during the initialization phase.

Major call 1

The EFS program can perform initialization processing such as opening data sets and obtaining storage.

Information is passed in both directions between DFSORT and the EFS program across the EFS interface.

At Major Call 1, DFSORT supplies your EFS program with fields in the EFS interface containing:

- An action code indicating that Major Call 1 is in effect
- Informational flags that describe current processing.

When the EFS program returns control to DFSORT, it can supply fields in the EFS interface containing:

- A control statement request list, with a list of DFSORT and non-DFSORT control statement operation definers, or EXEC PARM options

  Note: OUTFIL statements cannot be requested by an EFS program.
- An EFS Program Context area (a private communication area for the EFS program)
- A list containing messages for printing to the message data set
- A return code (in general register 15).

Major call 2

At this call, your EFS program can examine, alter, or ignore control statements before DFSORT processes them, and provide user-written messages to the message data set. DFSORT calls your EFS program once for each control statement or EXEC PARM you request.

At Major Call 2, DFSORT supplies your EFS program with fields in the EFS interface containing:

- An action code indicating that Major Call 2 is in effect
- The original control statement or EXEC PARM option requested by the EFS program
- The length of the original control statement or EXEC PARM option
- Informational flags that describe current processing
- An EFS Program Context area (a private communication area for the EFS program).

When the EFS program returns control to DFSORT, it can supply fields in the EFS interface containing:

- A modified version of the control statement or EXEC PARM option sent by DFSORT to the EFS program. If you plan to sort or merge user-defined data types, or include or omit user-defined data types, your EFS program must return new formats for the SORT/MERGE or INCLUDE/OMIT control statements. These new formats (D1 and D2) signal DFSORT to call the EFS01 and EFS02 exit routines you included with your EFS program.

  Note: OUTFIL statements cannot be passed to an EFS program or returned from an EFS program to be parsed.
- The length of the altered control statement or EXEC PARM option.
- Informational flags signaling DFSORT whether to parse or ignore the control statement or EXEC PARM option.
- A list of messages for DFSORT to print to the message data set.
- A return code (in general register 15).

Major call 3

At Major Call 3, your EFS program can provide DFSORT with user-written messages to print to the message data set. DFSORT can call the EFS program once for the Blockset technique and once for the Peerage/Vale techniques. DFSORT obtains more information at this call from the EFS program to process the EFS01 and EFS02 exit routines.

At Major Call 3, DFSORT supplies your EFS program with fields in the EFS interface containing:

- An action code indicating that Major Call 3 is in effect
- An extract buffer offsets list needed by the EFS01 exit routine
- A record lengths list of input and output records
- Informational flags that describe current processing
- An EFS Program Context area (a private communication area for the EFS program).

When the EFS program returns control to DFSORT, it can supply fields in the EFS interface containing:

- An EFS01 exit routine address
- An EFS02 exit routine address
- A list of messages for printing to the message data set

- A return code in general register 15.

Input phase

DFSORT runs the two exit routines, EFS01 and EFS02, during the input phase. The EFS01 routine supports sorting or merging user-defined data types with user-defined collating sequences and is called once for each record. The EFS02 routine provides logic to include or omit records on user-defined data types and is called one or more times for each record, according to the logic.

Information is passed in both directions between DFSORT and the exit routines across the EFS01 and EFS02 parameter lists.

DFSORT supplies the EFS01 routine with fields in the parameter list containing:

- An Extract Buffer Area to which the EFS01 routine must move all EFS control fields. See "EFS01 user exit routine" on page 737 for more information.
- The input data record.
- An EFS Program Context Area (a private communication area for the EFS program).

When the EFS01 routine returns control to DFSORT, it must return a return code in general register 15.

DFSORT supplies the EFS02 routine with fields in the parameter list containing:

- A Correlator Identifier, which identifies a relational condition containing EFS fields. See "EFS02 user exit routine" on page 738 for more information.
- The input data record.

When the EFS02 routine returns control to DFSORT, it must return a return code in general register 15.

Termination phase

DFSORT runs Major Calls 4 and 5 during the termination phase. Only one call is made at each of these Major Calls.

Note: If a system abend occurs while DFSORT's ESTAE recovery routine is in effect, and Major Calls 4 and 5 have not already been run, the ESTAE routine runs them. If an EFS abend occurs during Major Call 1, the ESTAE routine does not run Major Calls 4 and 5. See Appendix E, "DFSORT abend processing," on page 847 for more information about ESTAE.

Major call 4

The EFS program provides any final user-written messages for printing to the message data set.

At Major Call 4, DFSORT supplies your EFS program with fields in the EFS interface containing:

- An action code indicating that Major Call 4 is in effect.
- An EFS Program Context Area (a private communication area for the EFS program).

When the EFS program returns control to DFSORT, it can supply fields in the EFS interface containing:

- A message list containing messages for printing to the message data set.
- A return code (in general register 15).

Major call 5

The EFS program performs any termination processing, such as closing data sets and releasing storage.

At Major Call 5, DFSORT supplies your EFS program with fields in the EFS interface containing:

- An action code indicating that Major Call 5 is in effect.
- An EFS Program Context Area (a private communication area for the EFS program).

When the EFS program returns control to DFSORT, it must supply a return code in general register 15.

What you can do with EFS

You can design your EFS program to perform seven basic tasks at the initialization, input, and termination phases of DFSORT. Some of the tasks require using the EFS program-generated user exit routines EFS01 and EFS02.

Table 93. Functions of an Extended Function Support (EFS) Program

Functions of an Extended Function Support (EFS) Program EFS Program Functions	Initialization Phase	Input Phase	Termination Phase
Opening and initializing	EFS Program		
Examining, altering, or ignoring DFSORT and non-DFSORT control statements prior to processing by DFSORT	EFS Program		
Sorting or merging user-defined data types with user-defined collating sequences		EFS01	
Providing the logic to include or omit records based on user-defined data types		EFS02	
Supplying messages to DFSORT for printing to the message data set	EFS Program		EFS Program
Terminating DFSORT	EFS Program	EFS01, EFS02	EFS Program
Closing data sets and housekeeping			EFS Program

Opening and initializing data sets

Your EFS program can open data sets, obtain necessary storage, and perform other forms of initialization needed during a run.

Examining, altering, or ignoring control statements

At Major Call 1, your EFS program can send a control statement request list to indicate the control statements and EXEC PARM options you want DFSORT to send to your EFS program at Major Call 2. OUTFIL statements cannot be requested by an EFS program.

At Major Call 2, your EFS program can examine, alter, or ignore control statements and EXEC PARM options that DFSORT reads from the EXEC statement, SYSIN, SORTCNTL, DFSPARM, or a parameter list passed from an invoking program. OUTFIL statements cannot be passed to an EFS program or returned from an EFS program to be parsed.

Refer to Figure 37 on page 724 for an illustration of the control statement processing sequence used when an EFS program is activated.

The same override rules apply to control statements and parameters returned from an EFS program as apply to the original control statements and parameters.

For example, a STOPAFT parameter added to the SORT statement by an EFS program is overridden by a STOPAFT parameter in an OPTION statement in the same way as if the SORT statement originally contained the STOPAFT parameter.

See Appendix B, “Specification/override of DFSORT options,” on page 805 for full override details.

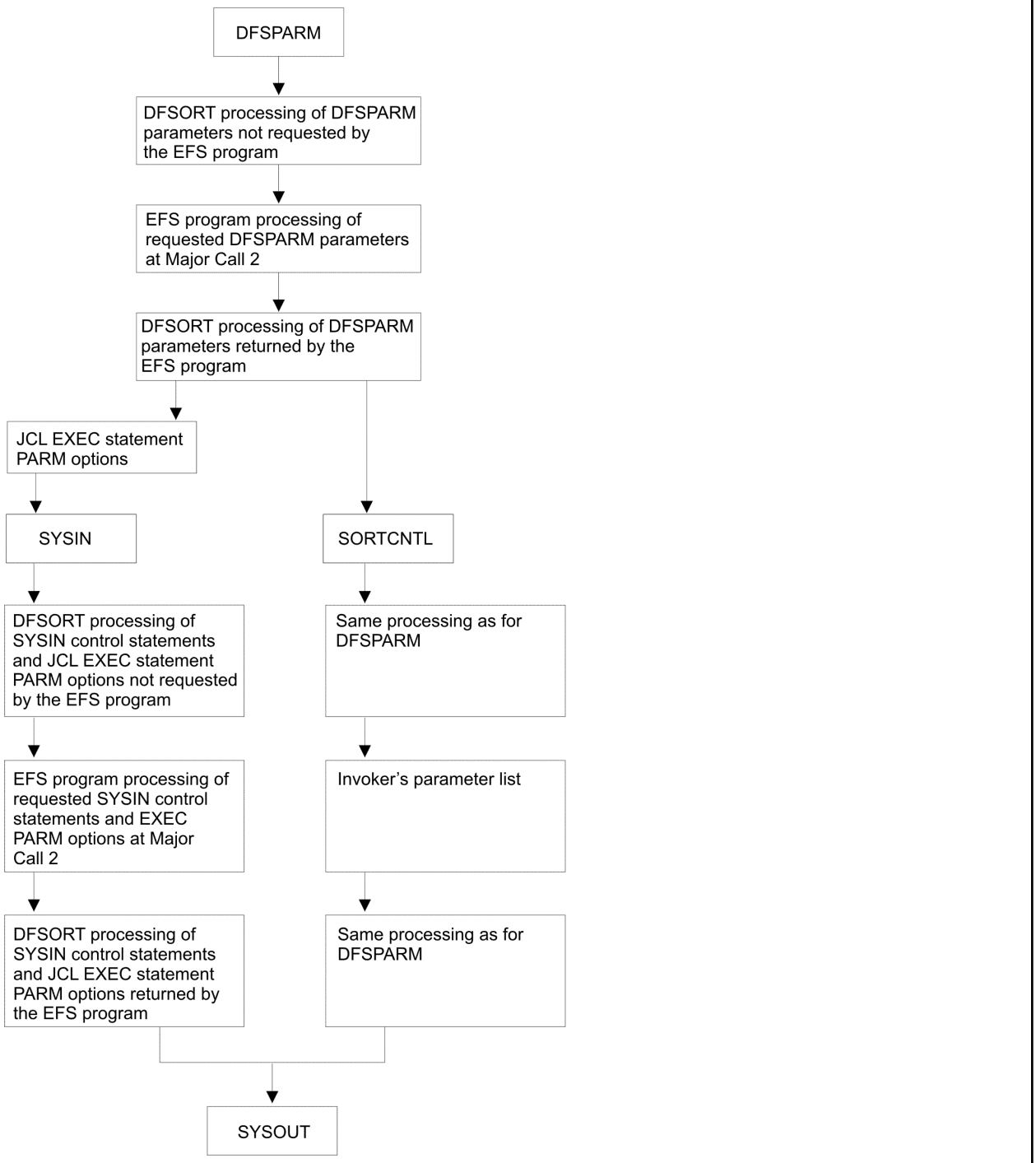


Figure 37. Control Statement Processing Sequence

Processing user-defined data types with EFS program user exit routines

You can write your EFS program to provide two user exit routines to perform various tasks during run-time.

Your EFS program user exit routines can:

- Process user-defined data types. Your EFS program can provide an EFS01 routine to alter any control field of an input record.
- Include or omit records based on user-defined data types. Your EFS program can provide an exit routine to examine any input field of an input record to determine whether or not to include that record for processing.

Supplying messages for printing to the message data set

You can use an EFS program to tailor messages for several purposes:

- To describe new types of operations
- To describe extended field parameters
- To customize the message data set to your site
- To display statistical information about control statements or EXEC PARM options.

You can control whether to print the control statements returned by an EFS program to the message data set with:

-
- The LISTX installation option (see “Installation defaults” on page 17)
- The LISTX or NOLISTX operators in the PARM field of the JCL EXEC statement (see “Specifying EXEC/DFSPARM PARM options” on page 32)
- The LIST or NOLIST operators of the OPTION program control statement.

Terminating DFSORT

Your EFS program can terminate DFSORT at any of the five Major Calls and also from either of the two EFS program exit routines during the input phase.

Closing data sets and housekeeping

At Major Call 5, your EFS program can close data sets, free storage and perform any other necessary housekeeping.

Structure of the EFS interface parameter list

The EFS interface consists of a variable-length parameter list and is used to communicate between DFSORT and your EFS program. DFSORT initializes the parameter list to zeros during the initialization phase, except that the list end indicator is set to X'FFFFFFFF'.

The parameter list resides below 16MB virtual, and remains accessible while the EFS program is active, although DFSORT might change its storage location during run-time to optimize use of storage. The actual address in register 1 (used to pass the interface parameter list address) can therefore change while DFSORT is running.

Figure 38 on page 726 and Figure 39 on page 727 illustrate the structure of the EFS interface parameter list. The illustrated portions of the list are explained in order in the following pages. EXEC PARMs are not described in the figure, but are included in processing.

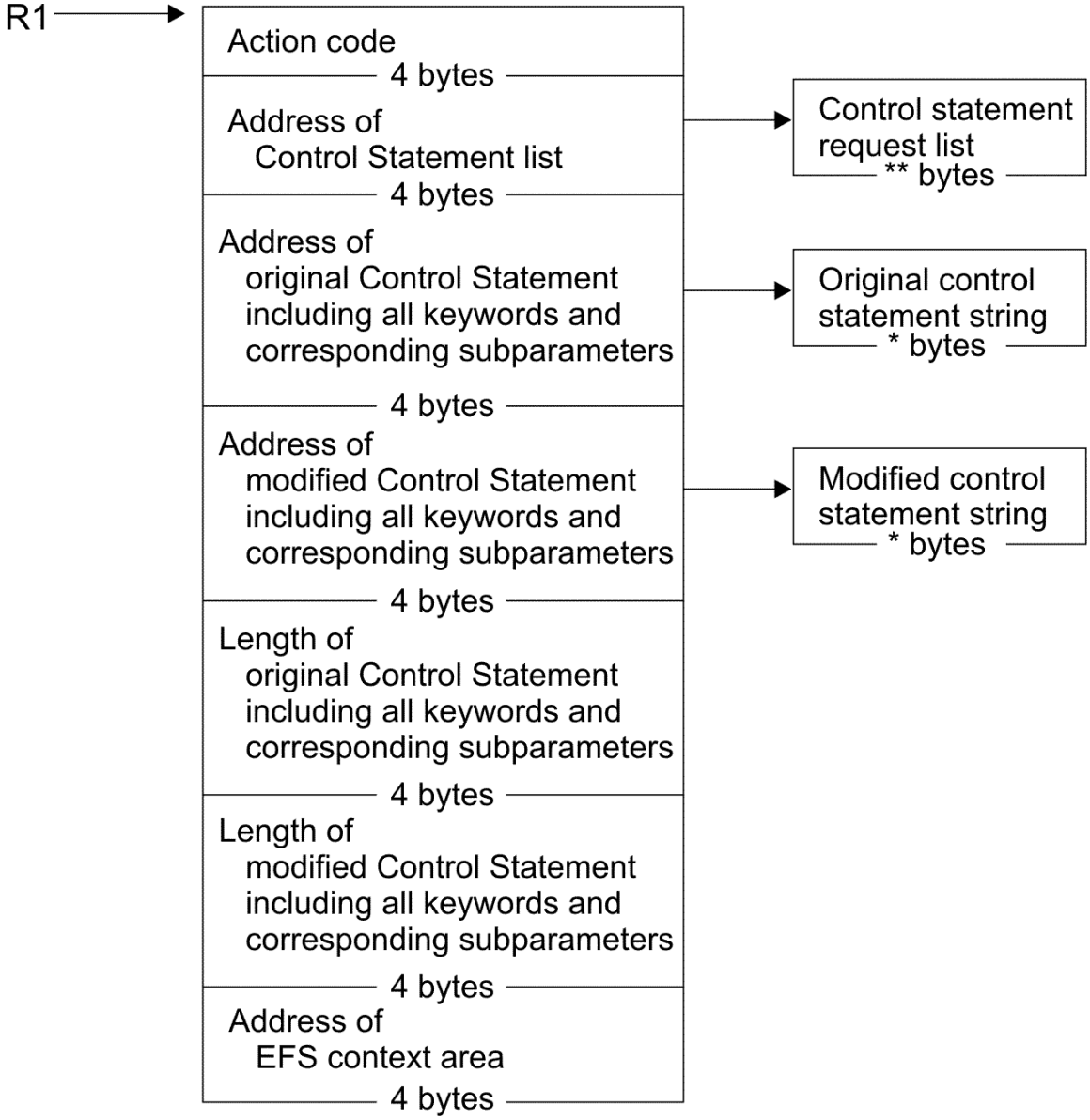


Figure 38. EFS Interface Parameter List - Part 1 of 2

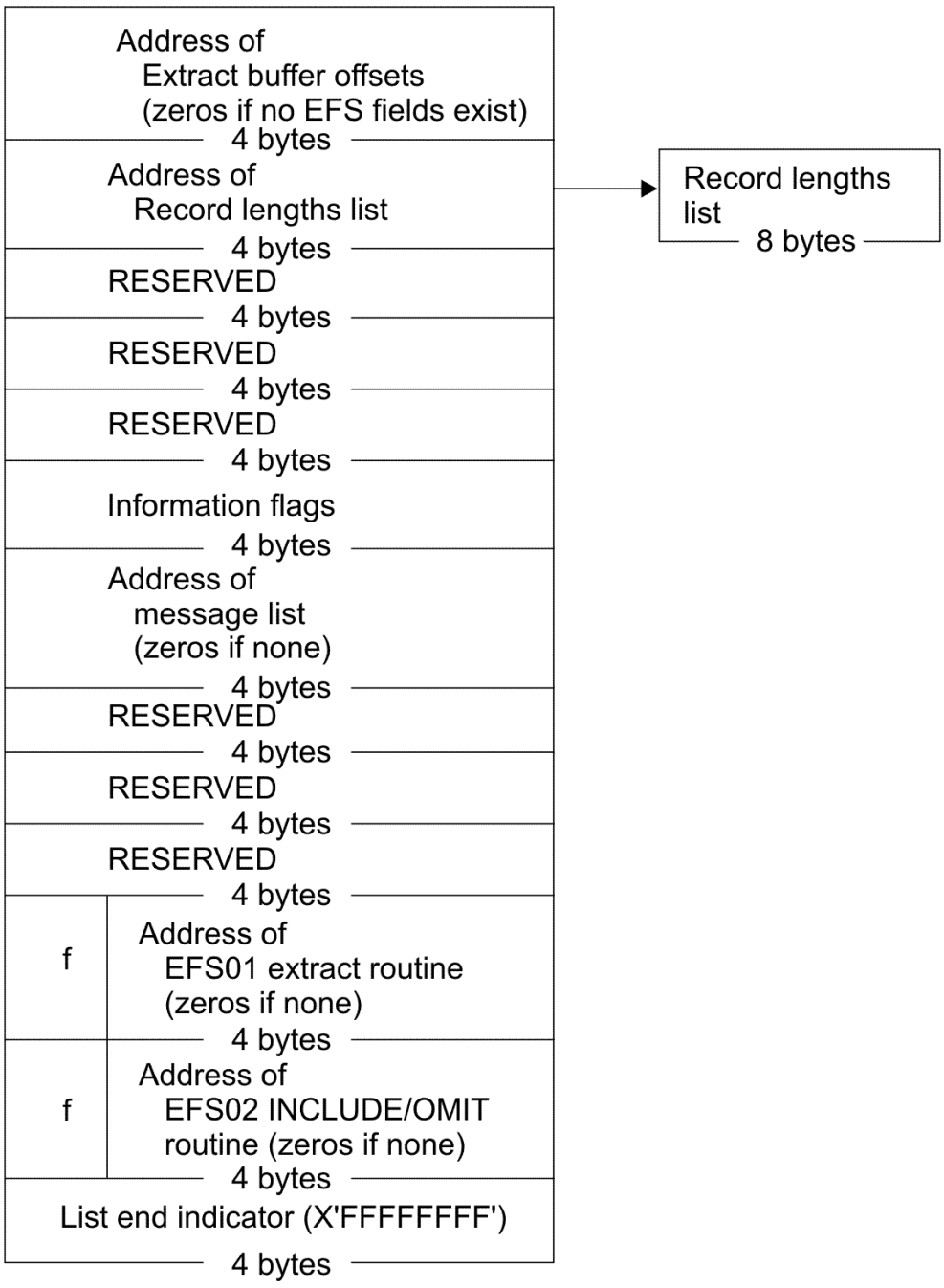


Figure 39. EFS Interface Parameter List - Part 2 of 2

Action codes

DFSORT sets one of five action codes before a call to the EFS program:

0
Indicates Major Call 1 to the EFS program. DFSORT sends this action code once.

4
Indicates Major Call 2 to the EFS program. DFSORT might send this action code several times at Major Call 2 depending on how many control statements are requested and found. For example, if the SORT, MERGE, and INCLUDE control statements are all supplied in SYSIN and are requested, the EFS program is called twice: once for the SORT control statement (because SORT and MERGE are mutually exclusive, and assuming the SORT statement is specified first, only the SORT statement is taken) and once for the INCLUDE control statement.

8
Indicates Major Call 3 to the EFS program. DFSORT can send this action code once for the Blockset technique and once for the Peerage/Vale technique.

12
Indicates Major Call 4 to the EFS program. DFSORT sends this action code once.

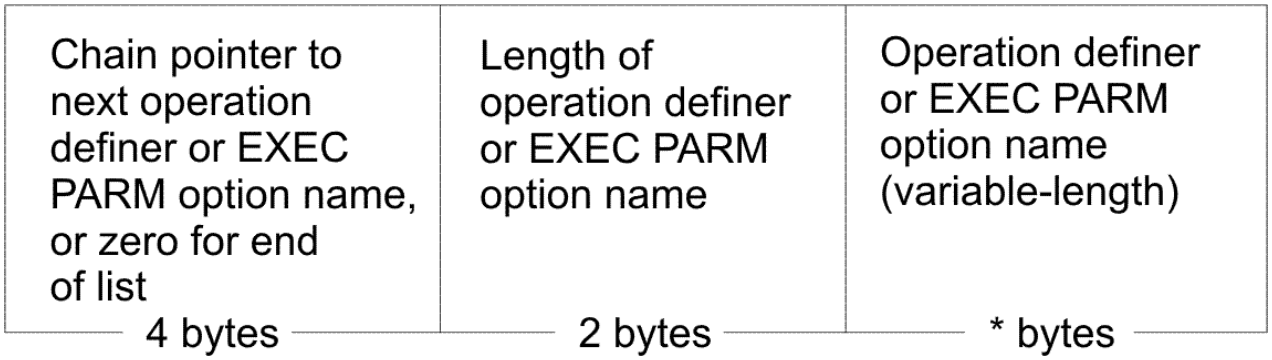
16
Indicates Major Call 5 to the EFS program. DFSORT sends this action code once.

Control statement request list

The control statement request list describes the control statements and the PARM options to be sent to the EFS program by DFSORT. The control statement request list consists of control statement operation definers and PARM option names. The maximum length allowed for an operation definer or PARM option name is eight bytes. If the operation definer or PARM option name is longer, DFSORT will use only the first eight bytes. Length field values must not include their own length.

OUTFIL statements cannot be requested by an EFS program.

Non-DFSORT operation definers and PARM options must be in EBCDIC format, and the first character must be non-numeric. The format of the control statement request list is:



The asterisk (*) indicates that the length is determined by the corresponding length field (maximum of 8 bytes).

Control statement string sent to the EFS program

DFSORT scans for the requested control statement from SYSIN, SORTCNTL, DFSPARM, or the invoker's parameter list to create a contiguous control statement string; DFSORT will handle any necessary continuation requirements for control statements from SYSIN, SORTCNTL, or DFSPARM. DFSORT scans for the requested PARM option to create a contiguous PARM option string.

DFSORT places a copy of the requested control statement or PARM option string in a contiguous storage area for the EFS program. No labels are supplied with the control statement; the address of the string always points to the first byte of the appropriate operation definer or PARM option.

DFSORT will send the requested control statement(s) or PARM option(s) to the EFS program as found by DFSORT; DFSORT will provide limited syntax checking of control statements or PARM option(s) before sending them to the EFS program.

In addition to following the rules in “General coding rules” on page 79, you must observe the following rules for non-DFSORT control statements:

- DFSORT will recognize a control statement with no operand(s) provided the operation definer (1) is supplied in SYSIN, SORTCNTL, or DFSPARM and (2) is the only operation definer contained on a line.
- Operation definers supplied through SYSIN, SORTCNTL, DFSPARM, or the extended parameter list and requested by the EFS program will not be recognized if they are longer than eight bytes.

In addition to observing the rules in *z/OS MVS JCL User's Guide* and *z/OS MVS JCL Reference* you must observe the following rule for non-DFSORT PARM options:

- PARM options requested by the EFS program will not be recognized if they are longer than eight bytes.

DFSORT will send the requested DFSORT or non-DFSORT control statements or PARM options that remain after DFSORT override rules have been applied.

- If duplicate DFSORT or non-DFSORT control statements or PARM options are supplied through the same source (such as SYSIN), then DFSORT will send the first occurrence of the control statement. The second occurrence of the DFSORT or non-DFSORT control statement or PARM option will be ignored by DFSORT.

  If duplicate DFSORT or non-DFSORT control statements are supplied through different sources (such as extended parameter list, SORTCNTL, and DFSPARM), then DFSORT will send the control statement remaining after different source override rules have been applied, except for the DFSORT OPTION and DEBUG control statements (see “Special handling of OPTION and DEBUG control statements ” on page 729).

  If mutually exclusive DFSORT control statements (such as SORT/MERGE) are supplied through the same source (such as SYSIN), then DFSORT will send the first occurrence of the control statement. The second occurrence of the DFSORT control statement will be ignored by DFSORT.

  If mutually exclusive DFSORT control statements (such as SORT/MERGE) are supplied through different sources (such as extended parameter list, SORTCNTL, and DFSPARM), then DFSORT will send the control statement remaining after different source override rules have been applied. The DFSORT control statement not sent will be ignored by DFSORT.

Thus the EFS program will not be sent duplicate DFSORT or non-DFSORT control statements (except for the DFSORT OPTION and DEBUG control statements as explained in “Special handling of OPTION and DEBUG control statements ” on page 729), or duplicate PARM options.

If the EFS program supplies non-DFSORT operands on the DFSORT OPTION control statement and the OPTION control statement is supplied in the extended parameter list, the EFS program must specify the non-DFSORT operands after all DFSORT operands.

DFSORT will free any storage it acquired for the control statement or PARM string.

Note: Blanks and quotes are very important to DFSORT in determining the control statement to send to an EFS program. Do not supply unpaired quotes in the INCLUDE/OMIT control statements, because DFSORT treats data within quotes as a constant, and treats blanks outside of quotes as the major delimiter.

Special handling of OPTION and DEBUG control statements

The override features of both the DFSORT OPTION and DEBUG control statements, when supplied through different sources, require special handling when EFS processing is in effect and either or both control statements are requested by the EFS program.

For example, DFSORT handles override for the OPTION and DEBUG control statements as follows:

- The OPTION control statement supplied in SORTCNTL will selectively override corresponding options on the OPTION control statement supplied in the extended parameter list.
- The DEBUG control statement supplied in SORTCNTL will selectively override corresponding options on the DEBUG control statement supplied in the 24-bit parameter list or the extended parameter list.

Because of these override features, DFSORT cannot simply send the OPTION control statement supplied in SORTCNTL and not send the OPTION control statement supplied in the extended parameter list. For the EFS program to process all possible operands on the OPTION control statements, DFSORT must send the OPTION control statements supplied in both SORTCNTL and the extended parameter list. DFSORT will

send both the OPTION and DEBUG control statements supplied through different sources. If duplicate OPTION or DEBUG control statements are supplied in the same source and the OPTION or DEBUG control statements are also supplied in different sources, DFSORT will send the first occurrence of both the OPTION and DEBUG control statements supplied through different sources.

Control statement string returned by the EFS program

Your EFS program can alter the control statement or PARM option string and replace it in the original contiguous storage area. If the area is too small, your program must allocate a new contiguous area. If the string is returned in a new storage area, your EFS program will be responsible for freeing the acquired storage.

Your EFS program must set an Informational flag to indicate whether the control statement or PARM option in the string should be parsed or ignored by DFSORT (see "Information flags" on page 734 for further details).

OUTFIL statements cannot be returned from an EFS program to be parsed.

Rules for parsing

The content and format of the altered control statement to be parsed must correspond to valid DFSORT values as described in Chapter 3, "Using DFSORT program control statements," on page 77, except when using the FIELDS operand with SORT or MERGE, or the COND operand with INCLUDE or OMIT (see "EFS formats for SORT, MERGE, INCLUDE, and OMIT control statements" on page 731).

You must observe the following rules for control statements to be returned to DFSORT for parsing:

-
- The operation definer and corresponding operands must be in uppercase EBCDIC format.
- At least one blank must follow the operation definer (SORT, MERGE, RECORD, and so on). A control statement can start with one or more blanks and can end with one or more blanks. No other blanks are allowed unless the blanks are part of a constant.
- Labels are not allowed; a leading blank, or blanks, before the control statement name is optional.
- No continuation character is allowed.
- Comment statements, blank statements, and remarks are not allowed.

The content and format of the altered EXEC PARM option to be parsed must correspond to valid DFSORT values as described in "Specifying EXEC/DFSPARM PARM options" on page 32.

The following operands will be ignored by DFSORT if returned by an EFS program on the OPTION control statement:

- EFS
- LIST
- NOLIST
- LISTX
- NOLISTX
- LOCALE
- MSGDDN
- MSGDD
- MSGPRT
- SMF
- SORTDD
- SORTIN
- SORTOUT

- USEWKDD

The following EXEC PARM options will be ignored by DFSORT if returned by an EFS program:

- EFS
- LIST
- NOLIST
- LISTX
- NOLISTX
- LOCALE
- MSGDDN
- MSGDD
- MSGPRT

EFS formats for SORT, MERGE, INCLUDE, and OMIT control statements

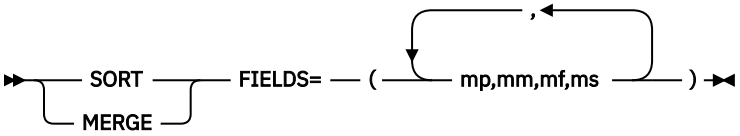
In addition to using the SORT, MERGE, INCLUDE, and OMIT control statements as explained in "Program Control Statements", you can also use two additional formats on the FIELDS and COND parameters. The formats are termed D1 and D2 and apply as follows:

- D1 with the FIELDS parameter of the SORT or MERGE statement
- D2 with the COND parameter of the INCLUDE or OMIT statement.

Use D1 and D2 to reflect data types that require special processing by EFS program exit routines EFS01 and EFS02, respectively. You cannot specify D2 format with the INCLUDE or OMIT parameters of the OUTFIL statement.

D1 format on FIELDS operand

The syntax for the SORT and MERGE statements using the D1 format on the FIELDS operand is as follows.



Where

Represents

mp
field position within the input record

mm
field length

mf
the D1 format that designates this field as an EFS control field

ms
must be either ascending (A) or descending (D); modification by an E61 exit (E) is not allowed.

Table 94 on page 732 gives an example of using the D1 format for a SORT control statement returned to DFSORT by the EFS program.

You must adhere to the following requirements for the D1 format:

- The mp, mm, and ms values returned must be valid SORT or MERGE control statement values, except:
  - The combined value of mp and mm may exceed the record length.
  - CHALT will have no effect on EFS fields and will not limit the length to 256.
  - Value E for ms will not be allowed; EFS fields may not be altered by an E61.

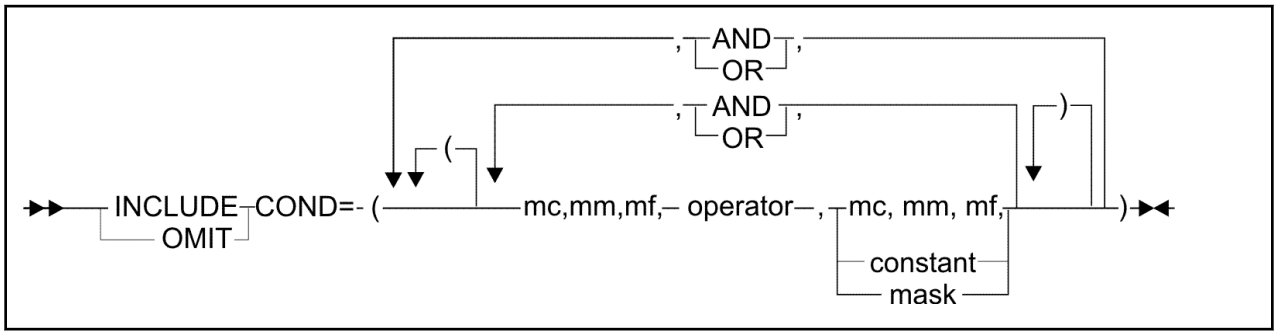
– FORMAT=D1 will not be allowed.

Table 94. D1 Format Returned by an EFS Program
D1 Format Returned by an EFS ProgramFormat

Original SORT control statement sent to EFSPGM
`SORT FIELDS=(15,4,FF,A,20,4,CH,A,40,7,FF,D)`
Altered SORT control statement returned by EFSPGM
`SORT FIELDS=(15,4,D1,A,20,4,CH,A,40,7,D1,D)`
where:
FF is a user-defined format that is modified to D1 by the EFS program before returning to DFSORT

D2 format on COND operand

Following is the syntax for the INCLUDE or OMIT statements using the D2 format on the COND operand.



Where

Represents

mc
the correlator identifier, a numeric value used to identify each relational condition

mm
field length

mf
the D2 format designating an EFS field within the relational condition

operator
a valid DFSORT comparison or bit logic operator

constant
a valid DFSORT decimal, character, hexadecimal or bit constant.

mask
a valid DFSORT hexadecimal or bit string

Table 95 on page 733 gives an example of using a correlator identifier and the D2 format for an INCLUDE control statement returned to DFSORT by the EFS program.

Note: The values for the correlator identifiers assigned to each relational condition by the EFS program can be in any chosen order. The example in Table 95 on page 733 shows a sequential ordering for the correlator identifiers.

You must adhere to the following requirements for the D2 format:

- The mc, mm, or constant values returned must be valid INCLUDE or OMIT control statement values, except:
  - The combined value of mc and mm might exceed the record length.

- Any valid DFSORT constant or mask is allowed.
- If COND=(mc1,mm1,mf1,operator,mc2,mm2,mf2) is used, both mf1 and mf2 must be D2.
- CHALT has no effect on EFS fields.
- FORMAT=D2 is not allowed.

Table 95. Correlator Identifier and D2 Format Returned by an EFS Program

Correlator Identifier and D2 Format Returned by an EFS ProgramFormat

Original INCLUDE control statement sent to EFSPGM
`INCLUDE COND=(15,4,FF,EQ,20,4,FF,AND,40,7,FF,NE,50,7,FF,OR,`
`30,2,FF,NE,35,2,FF)`
Altered INCLUDE control statement returned by EFSPGM
`INCLUDE COND=(1,4,D2,EQ,1,4,D2,AND,2,7,D2,NE,2,7,D2,OR,3,2,D2,NE,3,2,D2)`
Where:
• FF is a user-defined format and modified to D2 by the EFS program before returning to DFSORT.
• The first relational condition specified, (1,4,D2,EQ,1,4,D2), uses correlator identifier value 1 to identify this relational condition.
• The second relational condition specified, (2,7,D2,NE,2,7,D2), uses correlator identifier value 2 to identify this relational condition.
• The third relational condition specified, (3,2,D2,NE,3,2,D2), uses correlator identifier value 3 to identify this relational condition.

Length of original control statement

The control statement includes the first byte of the control statement through the last operand of the control statement or, if only an operation definer is supplied, the length of the operation definer. DFSORT does not send labels supplied with the control statement.

Length of the altered control statement

The length includes the first byte of the control statement through the last operand of the control statement. If leading blanks are provided, the length includes the first leading blank.

EFS program context area

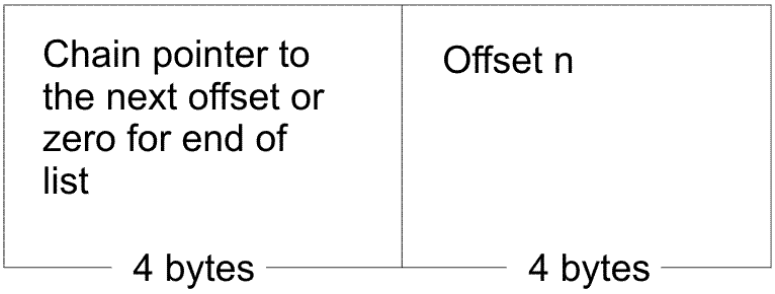
The EFS program context area is a private communication area that can be set up and used by the EFS program as appropriate. DFSORT sends the context area address to the EFS program at each Major Call and at each call to EFS01 and EFS02.

The EFS program is responsible for obtaining (at Major Call 1) and releasing (at Major Call 5) the necessary storage for the EFS program context area.

Extract buffer offsets list

A linked list of offsets into the extract buffer will be passed to your EFS program. The offsets show the starting positions into the buffer area of any EFS control fields specified on the SORT or MERGE FIELDS operand. The offsets are sent only for EFS control fields and are sent in the same order as specified on the FIELDS operand. If no EFS control fields exist, the address to the offsets is zero.

DFSORT frees any storage it acquired for the extract buffer offsets list. The format of the extract buffer offsets list is:

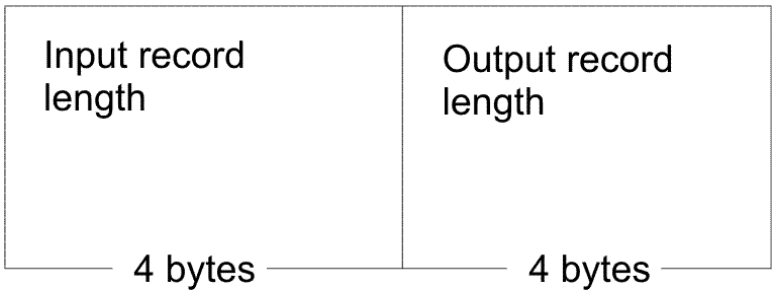


Record lengths list

The record lengths list is a linked list containing the input and output record lengths. You must be aware of the possible change in record size during run-time (for example, with an E15 user exit).

The input and output record lengths are sent to the EFS program for informational use only. DFSORT ignores any changes to the values made to the record lengths list returned by the EFS program.

DFSORT frees any storage it acquired for the record lengths list. The format of the record lengths list is:



Information flags

The information flags are defined in the figure that follows:

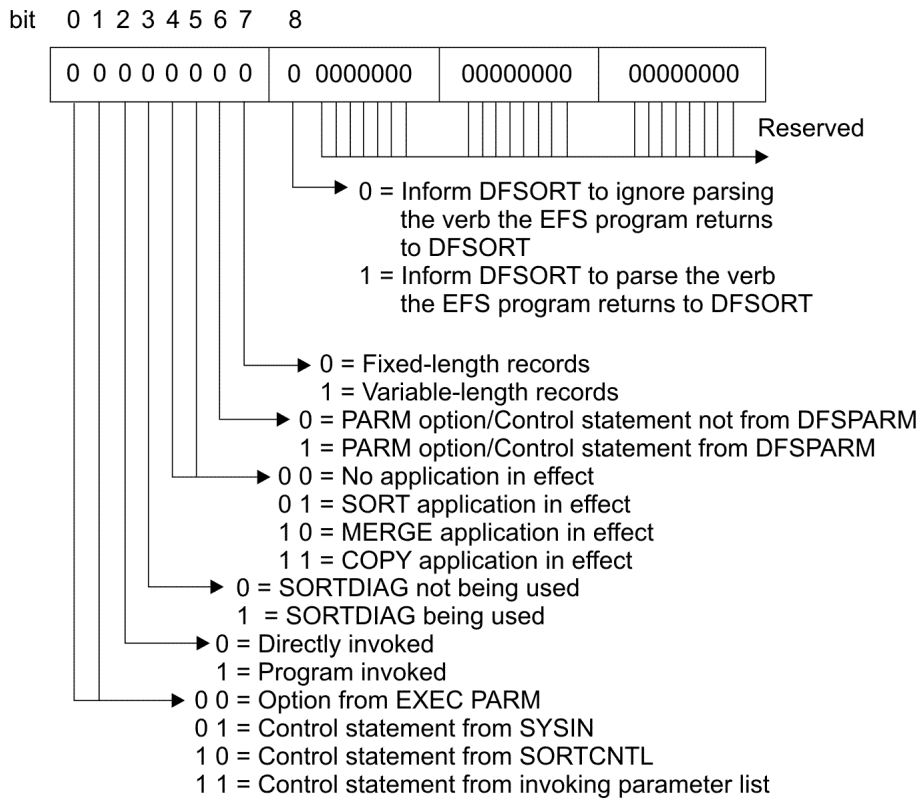


Figure 40. Information Flags

Bit

Description

Bits 0 and 1

Indicate the source of the control statement being processed. Information flags 0 and 1 are set by DFSORT before a call to the EFS program at Major Call 2 (multiple calls are possible at Major Call 2).

Bit 2

Indicates how DFSORT was invoked. Information flag 2 is set by DFSORT before Major Call 1 to the EFS program.

Bit 3

Indicates whether diagnostic messages are to be printed. Information flag 3 is set by DFSORT before Major Call 1 to the EFS program.

Bits 4 and 5

Indicate the DFSORT function being run. Information flags 4 and 5 are set by DFSORT before each call at Major Call 2 and Major Call 3 to the EFS program (multiple calls are possible at Major Call 2 and Major Call 3).

Bit 6

Indicates the source of PARM options and control statements from DFSPARM. Information flag 6 is set by DFSORT before each call at Major Call 2 to the EFS program (multiple calls are possible at Major Call 2).

Bit 7

Indicates whether fixed-length records or variable-length records are to be processed. Information flag 7 is set by DFSORT before each call at Major Call 3 to the EFS program (multiple calls are possible at Major Call 3).

Bit 8
Set by the EFS program to inform DFSORT whether to parse or ignore the control statement returned by the EFS program. Printing of the control statement is managed by the LISTX/NOLISTX parameters (see "OPTION control statement" on page 173 for further details). Information flag 8 is set by the EFS program before returning to DFSORT from each call at Major Call 2 (multiple calls are possible at Major Call 2).

Message list

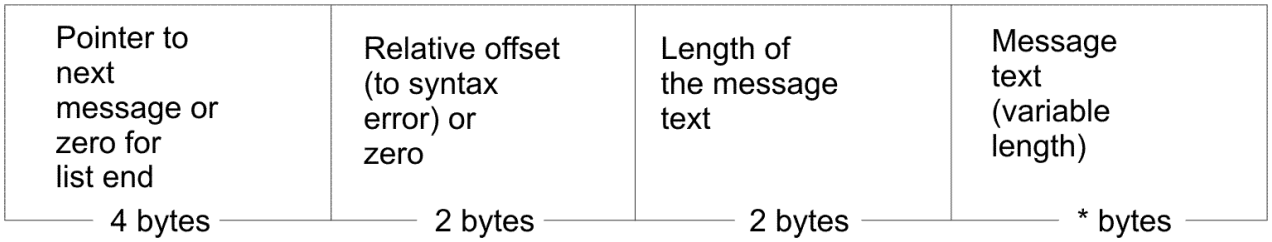
Your EFS program can return informational or critical messages. A return code of 0 in general register 15 indicates an informational message while a return code of 16 indicates a critical message. If the EFS program has no messages to send after a Major Call, it must zero the message list address in the EFS interface parameter list.

At Major Call 2, if the EFS program finds a syntax error in a control statement, it can return an offset relative to the start of the string to indicate the location of the error. DFSORT first prints the control statement in error and then prints another line containing a dollar symbol ($) at the location indicated by the offset.

Because DFSORT associates the relative offset with a critical message, the EFS program must return with a return code of 16 in general register 15. If a relative offset is returned for an EXEC PARM, the relative offset will be ignored. The EFS program must free any storage it acquired for its messages.

The length field values must not include their own length.

The message list format follows:



An asterisk (*) indicates that the length is determined by the corresponding length field.

DFSORT imposes no restrictions on the format of the messages returned by an EFS program. If you wish, you can use the DFSORT message format so that messages in the message data set are consistent in appearance. For a description of the message format used by DFSORT, see *z/OS DFSORT Messages, Codes and Diagnosis Guide*

EFS program exit routines

If you specify EFS control fields (D1 format) or EFS fields (D2 format), DFSORT calls the EFS01 or EFS02 exit routines, respectively, to process those fields. The routines are generated by your EFS program, which can return the following information about them at Major Call 3:

- The address of an extract routine, EFS01, which is used to extract the control fields of an input record to a buffer area before a sort or merge takes place; EFS01 is not applicable to a copy application.
- The address of an INCLUDE or OMIT routine, EFS02, which is used to process comparison logic for including or omitting records.

During the termination phase, the EFS program must free any storage used by these routines.

EFS01 and EFS02 function description

Each DFSORT control statement describes to DFSORT the type of operation to be performed on input data. Through the EFS interface, DFSORT enables an EFS program to provide user exit routines to perform functions beyond the capabilities of DFSORT control statements.

The EFS program can provide user exit routine EFS01 to supplement the function of the DFSORT SORT/MERGE control statements and can provide user exit routine EFS02 to perform the function of the DFSORT INCLUDE/OMIT control statements.

When preparing your EFS program exit routines, remember:

- The routines must follow standard linkage conventions.
- The general registers used by DFSORT for linkage and communication of parameters follow operating system conventions (see Figure 41 on page 737).
- The routines must use the described interfaces (see "EFS01 parameter list" on page 738 and "EFS02 parameter list" on page 739).

Register	Use
1	DFSORT places the address of a parameter list in this register.
13	DFSORT places the address of a standard save area in this register. The area can be used to save contents of registers used by the EFS program exit routine. The first word of the area contains the characters SM1 in its three low-order bytes.
14	Contains the address of DFSORT return point.
15	Contains the address of the EFS program exit routine. This register can be used as the base register for EFS program exit routine. This register is also used by the EFS program exit routine to pass return codes to DFSORT.

Figure 41. DFSORT Register Convention

EFS01 user exit routine

Processing of user-defined data types with the EFS01 exit routine requires using the function that alters control statements. EFS program requirements at Major Calls 1 and 2 are:

- At Major Call 1, the EFS program must provide the control statement request list with the SORT or MERGE operation definer. See "Control statement request list " on page 728 or further details.
- At Major Call 2, the EFS program must return a new format, D1, on the SORT or MERGE control statement that informs DFSORT to call the EFS01 routine, (the control fields defined with the D1 format are also known as EFS control fields). See "EFS formats for SORT, MERGE, INCLUDE, and OMIT control statements" on page 731 for further details. The EFS program must also return the final position, length, and sequence order. DFSORT uses the final position and length to create a list of offsets.

At Major Call 3, DFSORT sends the EFS program a list of offsets into a buffer. These offsets indicate where in the buffer the EFS program must have the EFS01 routine move the data indicated by the EFS control fields. See "Extract buffer offsets list" on page 733 for further details. At Major Call 3, the EFS program must return the address of the EFS01 routine to DFSORT.

During the input phase, DFSORT calls the EFS01 routine for each input record. The EFS01 exit routine must move all data indicated by the EFS control fields, specified in the SORT or MERGE FIELDS operand, from the input record to the extract buffer area as specified by the offsets in the extract buffer offsets list. For each EFS control field, the total number of bytes moved by EFS01 into the buffer area is equal to the total number of bytes specified in the *mm* parameter of the altered SORT or MERGE operand. Records are ordered according to the altered *ms* parameter.

The EFS01 routine is called to extract all EFS control fields to the extract buffer area each time a new record is brought into the input phase.

DFSORT will do sort or merge processing using the data in the extract buffer, and will treat the data as binary data.

EFS01 parameter list

DFSORT sends three words to the EFS01 user exit routine each time it is entered:

- The address of the extract buffer area
- The address of the input record
- The address of the EFS program context area.

DFSORT places the address of a parameter list in register 1. The list begins on a fullword boundary and is three fullwords long. The format of the parameter list is:

The format of the parameter list is:Bytes 1 through 4
Address of the extract buffer area
Address of the input record
Address of the EFS program context area

The EFS01 routine must return one of the following return codes in general register 15:

0
The extraction of the EFS control field was successful.

16
The extraction of the EFS control field was unsuccessful; terminate DFSORT.

EFS02 user exit routine

Including or omitting records based on user-defined data types with the EFS02 user exit routine requires using the function of altering control statements. EFS program requirements at Major Calls 1 and 2 are:

- At Major Call 1, the EFS program must provide the control statement request list with the INCLUDE or OMIT operation definer. See “Control statement request list ” on page 728 for further details.
- At Major Call 2, the EFS program must return a new format, D2, on the INCLUDE or OMIT control statement that informs DFSORT to call the EFS02 routine (the fields defined with the D2 format are also known as EFS compare fields). See “EFS formats for SORT, MERGE, INCLUDE, and OMIT control statements” on page 731 for further details. The EFS program must also return the final length and, in place of the position value, must send an identifier (known as a correlator identifier) that identifies a specific relational condition. For each relational condition containing EFS fields, there must be a unique correlator identifier to identify the particular relational condition. See “EFS formats for SORT, MERGE, INCLUDE, and OMIT control statements” on page 731 for further details.

At Major Call 3, the EFS program must return the address of the EFS02 routine to DFSORT.

The EFS02 routine is called to perform the INCLUDE or OMIT comparison logic for each relational condition containing an EFS field. During the input phase, DFSORT will call the EFS02 exit routine one or more times for each input record according to the evaluation defined by the AND, OR, or parentheses. The EFS02 exit routine must use the correlator identifier to determine the current relational condition being performed. EFS02 must perform the comparison logic for the current relational condition as identified by the correlator identifier. Figure 42 on page 739 repeats Table 95 on page 733 to illustrate an example of the calling sequences to an EFS02 by DFSORT.

Original INCLUDE control statement sent to EFSPGM

```
INCLUDE COND=(15,4,FF,EQ,20,4,FF,AND,40,7,FF,NE,50,7,FF,OR, 30,2,FF,NE,35,2,FF)
```

Altered INCLUDE control statement returned by EFSPGM

```
INCLUDE COND=(1,4,D2,EQ,1,4,D2,AND,2,7,D2,NE,2,7,D2,OR, 3,2,D2,NE,3,2,D2)
```

Where: the calling sequence to EFS02 may be summarized with the following tables:

Relational condition with	EFS02 returns a return code of 0=True or 4=False	DFSORT action when the next logical operator is:
		AND
Correlator Identifier 1	True	Call EFS02 with Correlator Id 2
	False	Call EFS02 with Correlator Id 3

Relational condition with	EFS02 returns a return code of 0=True or 4=False	DFSORT action when the next logical operator is:
		OR
Correlator Identifier 2	True	Include the record
	False	Call EFS02 with Correlator Id 3

Relational condition with	EFS02 returns a return code of 0=True or 4=False	DFSORT action when the next logical operator is:
		NONE
Correlator Identifier 3	True	Include the record
	False	Omit the record

Figure 42. Calling Sequence to EFS02 by DFSORT

EFS02 parameter list

DFSORT sends three words to the EFS02 exit routine each time it is entered:

- The address of the correlator identifier
- The address of the input record
- The address of the EFS program context area.

DFSORT places the address of a parameter list in register 1. The list begins on a fullword boundary and is three fullwords long. The format of the parameter list is:

Byte 1	Byte 2	Byte 3	Byte 4
00	00	00	Correlator identifier
Address of the input record			
Address of the EFS program context area			

The EFS02 exit routine must return one of the following return codes in general register 15:

0
True

The record passed the INCLUDE or OMIT test for the relational condition of an EFS field. If applicable, processing continues with the next relational condition. Otherwise, DFSORT accepts the record if INCLUDE is specified or omits the record if OMIT is specified.

4
False

The record did not pass the INCLUDE or OMIT test for the relational condition of an EFS field. If applicable, processing continues with the next relational condition. Otherwise, DFSORT omits the record if INCLUDE is specified or includes the record if OMIT is specified.

16
Terminate

An error occurred in processing the INCLUDE or OMIT logic; terminate DFSORT.

Addressing and residence mode of EFS program user exit routines

DFSORT supplies the following features to allow residence above or below 16MB virtual and use of either 24-bit or 31-bit addressing:

f (bit 0 of EFS program exit routine address)

0
Enter the EFS program exit routine with 24-bit addressing in effect.

1
Enter the EFS program exit routine with 31-bit addressing in effect.

The EFS program user exit routine can return to DFSORT with either 24-bit or 31-bit addressing in effect. The return address that DFSORT placed in register 14 must be used.

Except for the EFS program context area address (which DFSORT sends to the EFS program user exit routine unchanged), DFSORT handles the EFS program exit routine parameter list addresses (that is, the pointer to the EFS program exit routine parameter list and the addresses in the parameter list) as follows:

- If the EFS program exit routine is entered with 24-bit addressing in effect, DFSORT can pass clean (zeros in the first 8 bits) 24-bit addresses or 31-bit addresses to the EFS program exit routine. The EFS program exit routine must return clean 24-bit addresses if the EFS program exit routine returns to DFSORT with 31-bit addressing in effect.
- If the EFS program exit routine is entered with 31-bit addressing in effect, DFSORT can pass clean 24-bit addresses or 31-bit addresses to the EFS program exit routine. The EFS program exit routine must return 31-bit addresses or clean 24-bit addresses.

EFS program return codes you must supply

Your EFS program must pass one of two return codes to DFSORT:

0
Continue Processing

If you want DFSORT to continue processing for this Major Call, return with a return code of zero in general register 15.

16
Terminate DFSORT

If you want DFSORT to terminate processing for this Major Call, return with a return code of 16 in general register 15.

If the EFS program returns a return code of 16 from a Major Call prior to Major Call 4 or one of its generated user exit routines returns a return code of 16, DFSORT will skip interim Major Calls, where applicable, to the EFS program or user exit routine, and will call the EFS program at Major Call 4 and at Major Call 5.

Multiple calls are possible at Major Call 2 and Major Call 3. If the EFS program returns with a return code of 16 from one of the multiple calls at Major Call 2, subsequent calls at Major Call 2, if applicable, will be completed. If the EFS program returns with a return code of 16 from one of the multiple calls at Major Call 3, subsequent calls at Major Call 3, if applicable, will not be completed.

If the EFS program returns a return code of 16 at Major Call 4, DFSORT will still call the EFS program at Major Call 5.

Record processing order

The order of record processing when using EFS is similar to processing without it. Figure 43 on page 742 illustrates the record processing sequence for a sort or merge and Figure 44 on page 743 illustrates the record processing sequence for a copy when EFS processing is in effect.

The figures illustrate the same points as described in Figure 2 on page 10 with the following exceptions:

- When record processing is done for an INCLUDE or OMIT control statement, an EFS02 user exit routine is called to perform the comparison logic for the relational conditions with EFS fields.
- When record processing is done for a SORT or MERGE control statement, an EFS01 user exit routine is called to perform the extraction process for EFS control fields.

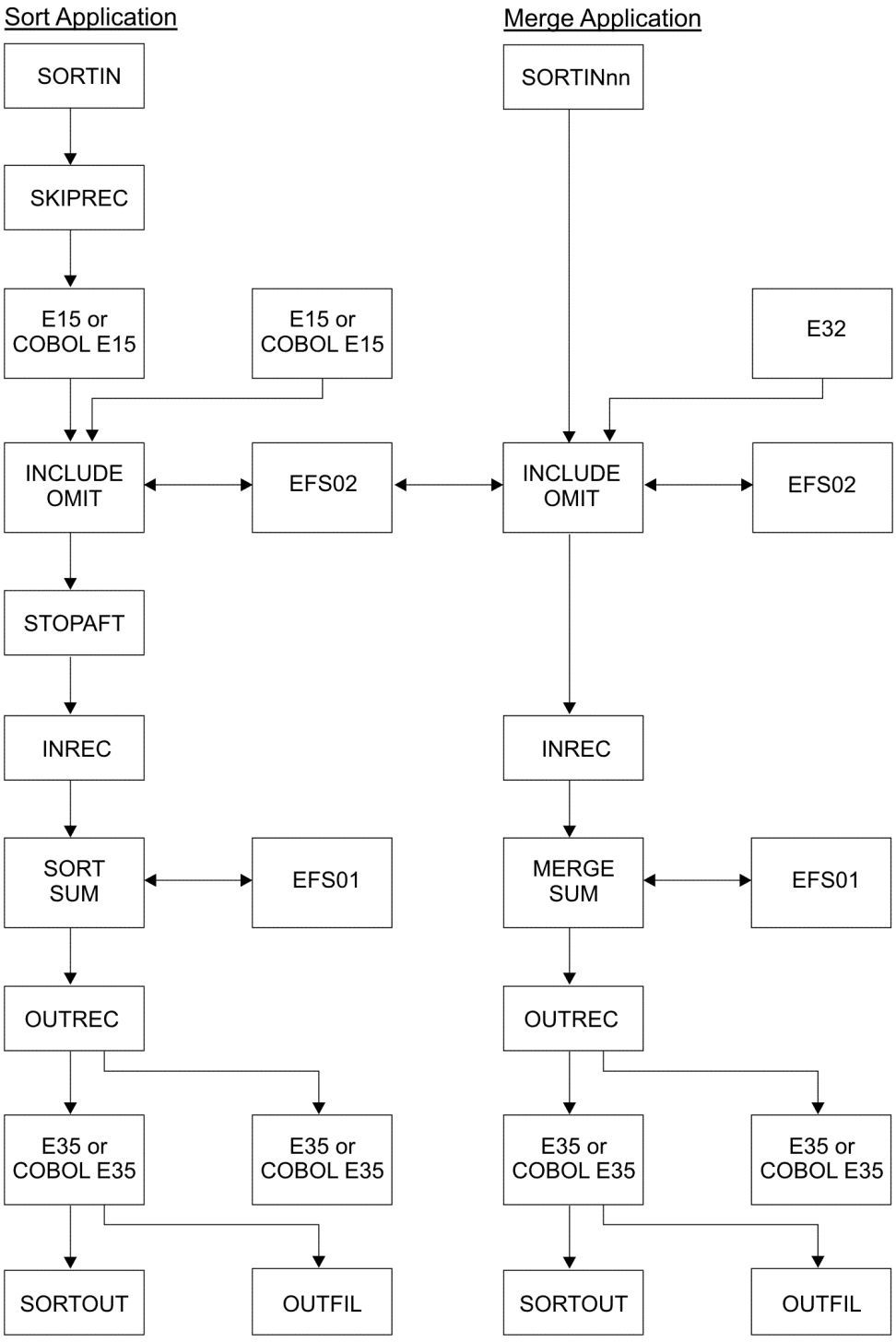


Figure 43. EFS Record Processing Sequence for a Sort or Merge

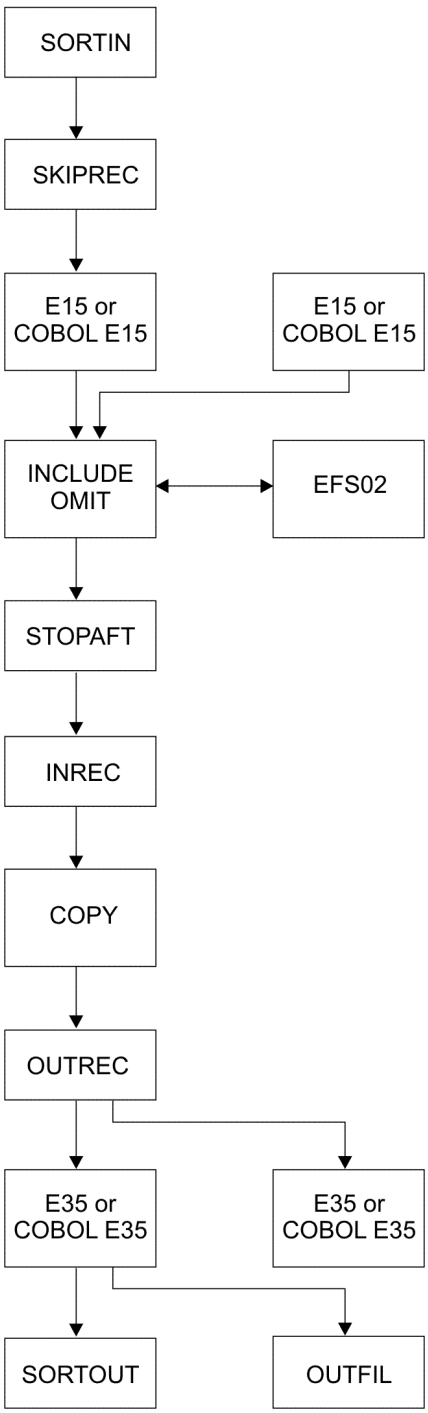


Figure 44. EFS Record Processing Sequence for a Copy

How to request a SNAP dump

You can request a SNAP dump for diagnostic purposes before or after any Major Call except Major Call 1. Use either the EFSDPBFR parameter or the EFSDPAFT parameter on the DEBUG statement.

See "DEBUG control statement" on page 86 for the correct syntax.

EFS program example

The following example shows how an EFS program can be used to change control statements at run-time.

The following information is assumed for this example DFSORT run:

- The EFS program "EFSPGM" resides in the same library as the DFSORT modules.
- The JCL statements for the application are:

```
//EXAMPLE1 JOB A12345,'J. SMITH'
//S1 EXEC  PGM=SORT,PARM='EFS=EFSPGM'
//SYSOUT   DD  SYSOUT=A
//SORTIN   DD  DSNAME=SMITH.INPUT,DISP=SHR,
//     UNIT=3380,SPACE=(TRK,(15,2)),VOL=SER=XYZ003,
//     DCB=(LRECL=80,BLKSIZE=80,RECFM=F)
//SORTOUT  DD  DSNAME=SMITH.OUTPUT,DISP=(NEW,KEEP),
//     UNIT=3380,SPACE=(TRK,(15,2)),VOL=SER=XYZ003
//SYSIN    DD  *
    SORT FIELDS=(5,20,CH,A,13,5,BI,D)
    OPTION STOPAFT=30,DYNALLOC=3390
/*
```

DFSORT initialization phase:

Major call 1

Prior to Major Call 1, DFSORT sets the following fields in the EFS interface parameter list:

- Action code=0

  Major Call 1 is in effect.
- Informational bit flag 2=0

  The DFSORT run is JCL-invoked.
- Informational bit flag 3=0

  SORTDIAG is not in effect.

DFSORT calls EFS program EFSPGM at Major Call 1, and EFSPGM sets the following fields in the EFS interface parameter list:

- Control statement request list

  Contains the OPTION operation definer, which indicates to DFSORT that the OPTION control statement is requested by EFSPGM.
- EFSPGM program context area

  EFSPGM will be using the context area.
- Message list=0

  EFSPGM has no messages for DFSORT to print to the message data set. General register 15 is set to zero.

Major call 2

Prior to Major Call 2, DFSORT sets the following fields in the EFS interface parameter list:

- Action code=4

  Major Call 2 is in effect.
- Informational bit flag 4=0 and informational bit flag 5=0

  No application is in effect.

EFSPGM requested the OPTION control statement. DFSORT makes a call to EFS program EFSPGM for each control statement requested; in this case, one. DFSORT also sets the following fields in the EFS interface parameter list:

- Informational bit flag 0=0 and informational bit flag 1=1

  The requested control statement came from SYSIN.
- The ***original*** OPTION control statement, including all operands and corresponding subparameters

  OPTION STOPAFT=30,DYNALLOC=3390
- The length of the *original* OPTION control statement, including all operands and corresponding subparameters

  The original control statement string is 31 bytes long.

DFSORT calls EFS program EFSPGM at Major Call 2, and EFSPGM sets the following fields in the EFS interface parameter list:

- Informational bit flag 8=1

  DFSORT must parse the control statement returned by EFSPGM.
- The *altered* OPTION control statement, including all operands and subparameters

  OPTION STOPAFT=30,DYNALLOC=3380,EQUALS
- The length of the *altered* OPTION control statement, including all operands and subparameters

  The altered control statement string is 38 bytes long.
- Message list=0

  EFSPGM has no messages for DFSORT to print to the message data set. General register 15 is set to zero.

Table 96 on page 745 shows the original control statement sent to EFS program EFSPGM and the altered control statement returned by EFS program EFSPGM.

Table 96. Original and Altered Control Statements

Original and Altered Control StatementsControl statement format
Original OPTION control statement sent to EFSPGM OPTION STOPAFT=30,DYNALLOC=3390 **Altered** OPTION control statement returned by EFSPGM OPTION STOPAFT=30,DYNALLOC=3380,EQUALS **Where:** STOPAFT=30 is the original operand and value DYNALLOC=3380 is the original operand with a new value EQUALS option has been added

Major call 3

Prior to Major Call 3, DFSORT sets the following fields in the EFS interface parameter list:

- Action code=8

  Major Call 3 is in effect.
- Informational bit flag 4=0 and informational bit flag 5=1

  A sort application is in effect.

- Informational bit flag 7=0

  Fixed-length records are being processed.
- Record lengths list values=80

  The LRECL of the input and output data sets is 80. Because the SORTOUT LRECL was not specified, DFSORT defaulted to the SORTIN LRECL for the SORTOUT LRECL.
- Extract buffer offsets list=0

  No EFS control fields were specified on the SORT control statement.

DFSORT calls EFS program EFSPGM at Major Call 3, and EFSPGM sets the following fields in the EFS interface parameter list:

- EFS01 address=0

  Because the SORT control statement has no EFS control fields, the EFS01 user exit routine is not used.

  Because no INCLUDE control statement was supplied (with EFS fields), the EFS02 user exit routine is not used.
- Message list=0

  EFSPGM has no messages for DFSORT to print to the message data set. General register 15 is set to zero.

DFSORT termination phase

Major call 4

Prior to Major Call 4, DFSORT sets the following fields in the EFS interface parameter list:

- Action code=12

  Major Call 4 is in effect.

DFSORT calls EFS program EFSPGM at Major Call 4, and EFSPGM sets the following fields in the EFS interface parameter list:

- Message list=0

  EFSPGM has no messages for DFSORT to print to the message data set.

And general register 15 is set to zero.

Major call 5

Prior to Major Call 5, DFSORT sets the following fields in the EFS interface parameter list:

- Action Code=16

  Major Call 5 is in effect.

DFSORT calls EFS program EFSPGM at Major Call 5, and EFSPGM does not set any fields in the EFS interface parameter list but sets general register 15 to zero.

Chapter 10. Improving efficiency

Improving performance

DFSORT is designed to optimize performance automatically. It sets optimization variables (such as buffer sizes) and selects the most efficient of several sorting and merging techniques.

You can improve DFSORT performance in several ways:

- Design your applications to maximize performance:
  - Directly invoke DFSORT processing
  - Plan ahead when designing new applications
  - Specify efficient sort/merge techniques
  - Specify input/output data set characteristics accurately
  - Use extended format data sets
  - Use DFSMSrmm-managed tapes, or ICETPEX
  - Specify devices that improve elapsed time
  - Use options that enhance performance
  - Use DFSORT's fast, efficient productivity features
  - Avoid options that degrade performance.
- Use main storage efficiently
- Allocate temporary work space efficiently
- Use Hipersorting
- Sort with data space
- Use memory object sorting
- Use ICEGENER instead of IEBGENER
- Use DFSORT's Performance Booster for The SAS System
- Use DFSORT's BLDINDEX support.

The DFSORT *z/OS DFSORT Tuning Guide* provides additional information related to many of the topics covered in this chapter.

Design your applications to maximize performance

Even though DFSORT automatically optimizes performance when your application is run, you can still improve efficiency by using specifications and options that permit DFSORT to make the best possible use of available resources.

Directly invoke DFSORT processing

You can enhance performance by invoking DFSORT with JCL instead of invoking it from a COBOL or a PL/I program. Generally, COBOL or PL/I is used for convenience. However, the trade-off can be degraded performance. You can improve efficiency by taking advantage of the way DFSORT installation defaults and run-time options can be fine-tuned for optimum performance, especially to make use of control statements that "work together," such as INCLUDE/OMIT, INREC/OUTREC, SUM, and OUTFIL. You can eliminate records from input files, reformat records to eliminate unwanted fields, combine records arithmetically, and create reports, without requiring routines from other programs.

Plan ahead when designing new applications

You should consider several factors when designing new applications. Some of these factors are discussed in this section.

Whenever possible:

- Use either EBCDIC character or binary control fields
- Place binary control fields so they start and end on byte boundaries
- Avoid using the alternative collating sequence character translation
- If you know that a fixed-point control field always contains positive values, specify it as a binary field.
- If you know that a packed decimal or zoned decimal control field always contains positive values with the same sign (for example, X'C'), specify it as a binary field.
- Use packed decimal format rather than zoned decimal
- If several contiguous character or binary control fields in the correct order of significance are to be sorted or merged in the same order (ascending or descending), specify them as one control field
- Avoid overlapping control fields.
- Avoid using locale processing if your SORT, MERGE, INCLUDE, or OMIT character fields can be processed using the binary encoding of the data.

Efficient blocking

You can improve the performance of DFSORT significantly by blocking your input and output records efficiently. Whenever possible, use system-determined optimum block sizes for your data sets.

For more information about letting DFSORT select system-determined optimum block sizes for your output data sets, see the discussion of the SDB option in “OPTION control statement” on page 173.

Specify efficient sort/merge techniques

Depending on various conditions, DFSORT selects different techniques for sorting and merging. Message ICE143I informs you which technique has been selected.

For copy applications, Blockset is the only technique used. If your program cannot use Blockset, DFSORT issues error message ICE160A and stops processing.

Sorting techniques

One condition that affects which sorting technique DFSORT selects is the type of device used for intermediate storage. If you use a tape device, the Conventional technique is used, which is less efficient. For more information on using tape devices for intermediate storage, see “Tape work storage devices” on page 758.

The Blockset and Peerage/Vale techniques can be used only with disk work data sets. These techniques are discussed later in this section.

Blockset sorting techniques

DFSORT's most efficient techniques, FLR-Blockset (for fixed-length records) and VLR-Blockset (for variable-length records), will be used for most sorting applications.

Notes

- The Blockset technique might require more intermediate work space than Peerage/Vale. For more information, see “Allocate temporary work space efficiently” on page 757.
- If Blockset is not selected, you can use a SORTDIAG DD statement to force message ICE800I, which gives a code indicating why Blockset cannot be used.

Peerage/vale sorting techniques

When the conditions for use of the Blockset sorting technique are not met, DFSORT uses Peerage/Vale.

Merging techniques

For merging applications, DFSORT uses the Blockset and Conventional techniques.

Blockset merging techniques

DFSORT's most efficient techniques, FLR-Blockset (for fixed-length records) and VLR-Blockset (for variable-length records), will be used for most merging applications.

Note: If Blockset is not selected, you can use a SORTDIAG DD statement to force message ICE800I, which gives a code indicating why Blockset cannot be used.

Conventional merging technique

When the conditions for use of the Blockset merging technique are not met, DFSORT uses the Conventional merge technique, which is less efficient.

Specify input/output data set characteristics accurately

DFSORT uses the information given it (about the operation it is to perform) to optimize for highest efficiency. When you supply incorrect information or do not supply information such as data set size and record format, the program makes assumptions that, if incorrect, can lead to inefficiency or program termination.

Input file size

When DFSORT has accurate information about the input file size, it can make the most efficient use of both main storage and intermediate work storage. See "File size and dynamic allocation" on page 800 for information about when and how to specify the input file size.

Variable-length records

When the input data set consists of variable-length records and dynamic allocation of intermediate data sets is used, specify the average record length as accurately as possible using AVGRLEN=n in the OPTION statement.

Disk devices

System performance is improved if storage is specified in cylinders rather than tracks or blocks. Storage on sort work data sets will be readjusted to cylinders if possible. The number of tracks per cylinder for disk devices is shown in Table 97 on page 749.

Table 97. Number of Tracks per Cylinder for Disk Devices

Number of Tracks per Cylinder for Disk DevicesDevice	Tracks per Cylinder
3380	15
3390	15
9345	15

If WRKSEC is in effect and the work data set is not allocated to virtual I/O, DFSORT allocates secondary extents as required, even if not requested in the JCL.

Allocating twice the space used by the input data sets is usually adequate for the work data sets. Certain conditions can cause additional space requirements. These include:

- Long control words (more than 150 bytes)
- Using different device types or work data sets
- Using an alternative collating sequence
- Low ratio of available storage to input file size.

Care should be taken to ensure that the LRECL parameter of the DCB corresponds to the actual maximum record length contained in your data set.

Use extended format data sets

Extended format data sets have features that can:

- significantly reduce the elapsed time DFSORT spends reading and writing data
- assist in managing the space requirements of very large data sets.

The Sequential Data Striping feature can reduce the time required to read and write your DFSORT input and output data sets. We recommend using sequential data striping for your very large DFSORT input and output data sets as a way to improve elapsed time.

Features such as compression, extended addressability (for VSAM), increased extent and volume support, and space constraint relief options assist in managing the space requirements of very large data sets.

If you decide to use compression, realize that there is a CPU cost which could outweigh the I/O performance benefits. Although compression can be used to improve disk storage capacity, the impact to CPU usage and elapsed time should be evaluated in your environment.

Consult with your storage administrator to learn how these features can be exploited in your environment.

Use DFSMSrmm-managed tapes, or ICETPEX

The use of tapes managed by DFSMSrmm, or a tape management system that uses ICETPEX, allows DFSORT to obtain accurate information about input file size and data set characteristics. This can result in improved performance and more efficient use of both main storage and intermediate work storage.

Specify devices that improve elapsed time

To get the best elapsed time improvement when using disk with DFSORT, you should use high speed devices, such as IBM's Enterprise Storage Server (ESS) and TotalStorage RS8000 Series subsystems for SORTIN, SORTOUT, SORTWKdd, SORTOUT, and OUTFIL data sets. Throughput can be further improved by leveraging high speed channels, such as IBM's FICON connectivity solutions.

Use options that enhance performance

To obtain optimum performance, you can fine-tune your installation and run-time options. Several options that can enhance performance are described later in this section.

CFW

Blockset sorting performance may be improved by using the Cache Fast Write (CFW) capability of IBM's storage controllers. Because IBM's latest storage subsystems have large cache sizes and high speed disk arrays, the use of CFW may not produce a significant performance gain.

The CFW parameter specifies whether DFSORT can use CFW when processing SORTWKdd data sets. The default is CFW=YES.

If you use the Hyperswap function in an environment that has a high availability configuration, you should specify CFW=NO to prevent DFSORT from abending when a Hyperswap is initiated.

DSA

Performance can be improved for Blockset sort applications by using Dynamic Storage Adjustment (DSA).

The DSA parameter sets the maximum amount of storage available to DFSORT for dynamic storage adjustment of a Blockset sort application when SIZE/MAINSIZE=MAX is in effect. If you specify a DSA value greater than the TMAXLIM value, you allow DFSORT to use more storage than the TMAXLIM value if doing so should improve performance. DFSORT only tries to obtain as much storage as needed to improve performance up to the DSA value.

DSPSIZE

Performance can be improved for sort applications that use the Blockset technique by using dataspace sorting.

The DSPSIZE parameter sets the maximum size of a data space to be used during a run. Specifying DSPSIZE=MAX allows DFSORT to optimize the maximum size of a data space to be used during a run, subject to other system and concurrent Hipersorting, memory object sorting and dataspace sorting activity throughout the run. Total dataspace sorting activity on a system can be further limited by the EXPMAX, EXPOLD, and EXPRES installation options. See the description of DSPSIZE in "OPTION control statement" on page 173 for more information.

FASTSRT

By specifying the COBOL FASTSRT compiler option, you can significantly reduce DFSORT processor time, EXCPs, and elapsed time. With FASTSRT, DFSORT input/output operations are more efficient because DFSORT rather than COBOL does the input/output (see Figure 45 on page 752). For more details, see the COBOL publications.

The FASTSRT option does not take effect for input and output if input and output procedures are used in the SORT statement. Many of the functions usually performed in an input or output procedure are the same as those done by DFSORT INREC, OUTFIL, OUTREC, INCLUDE or OMIT, STOPAFT, SKIPREC, and SUM functions. You might be able to eliminate your input and output procedures by coding the appropriate DFSORT program control statements and placing them in either the DFSPARM (DFSORT), SORTCNTL (DFSORT), or IGZSRTCD (COBOL) data set, thereby allowing your SORT statement to qualify for FASTSRT.

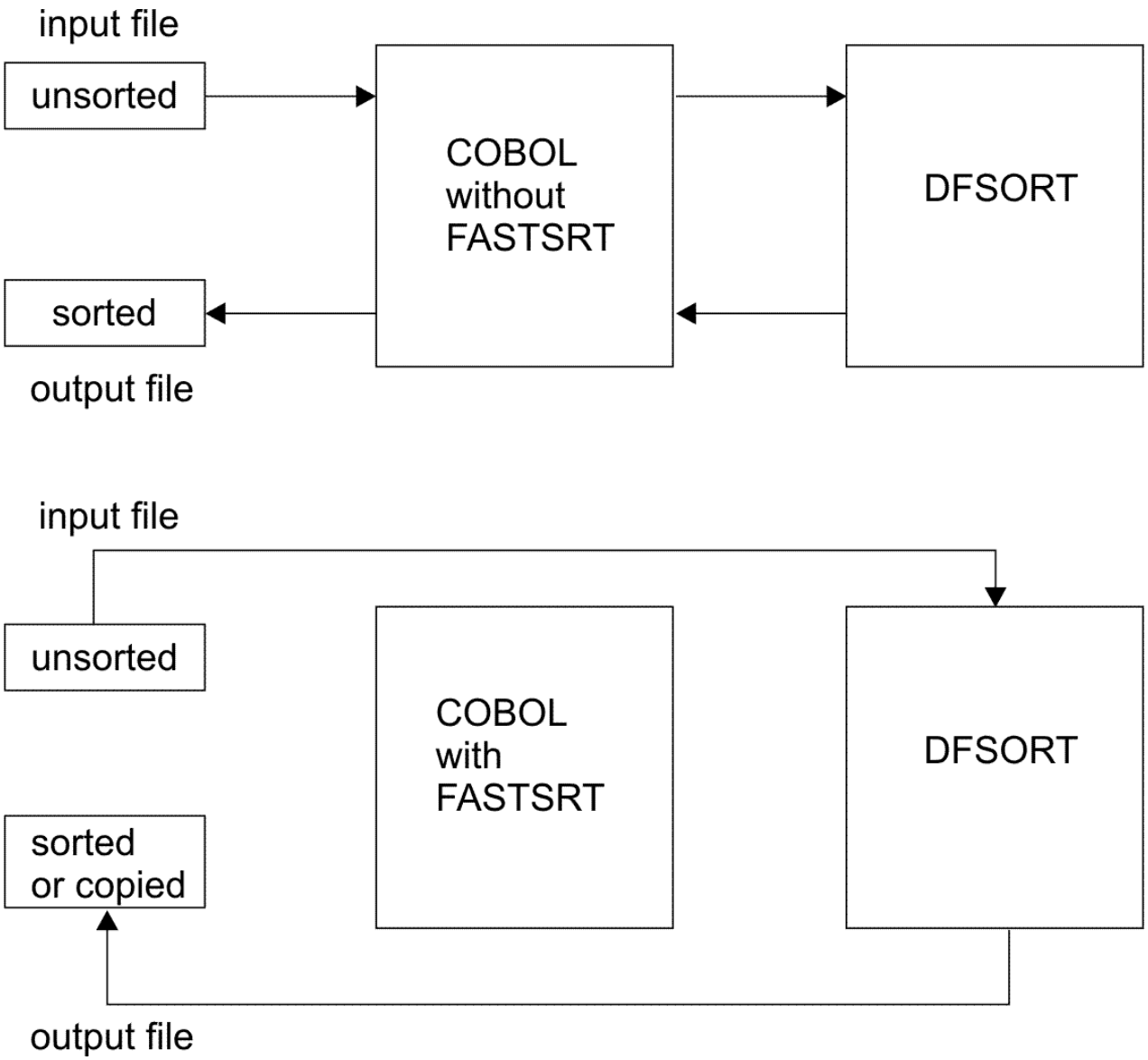


Figure 45. Faster Sorting with COBOL

SDB

To improve Blockset elapsed time, and disk and tape utilization, specify installation option SDB=LARGE as your site's installation (SDB=INPUT is the IBM-supplied default). SDB=LARGE allows DFSORT to select the system-determined optimum block size for your disk and tape output data sets, when appropriate.

HIPRMAX

Blockset sorting performance can be improved by using Hiperspace along with disk for temporary storage.

The HIPRMAX parameter sets the maximum amount of Hiperspace to be used during a run. Specifying HIPRMAX=OPTIMAL allows DFSORT to optimize the maximum amount of Hiperspace to be used during a run, subject to other system and concurrent Hipersorting, memory object sorting and dataspace sorting activity throughout the run. Total Hipersorting activity on a system can be further limited by the EXPMAX, EXPOLD, and EXPRES installation options. See the description of HIPRMAX in "OPTION control statement" on page 173 for more information.

MOSIZE

Blockset sorting performance can be improved by using memory object sorting.

The MOSIZE parameter sets the maximum amount of memory object storage to be used during a run. Specifying MOSIZE=MAX allows DFSORT to optimize the maximum size of a memory object to be used during a run, subject to other system and concurrent Hipersorting, memory object sorting and dataspace sorting activity throughout the run. Total memory object sorting activity on a system can be further limited by the EXPMAX, EXPOLD, and EXPRES installation options. See the description of MOSIZE in "OPTION control statement" on page 173 for more information.

Use DFSORT's fast, efficient productivity features

DFSORT offers a rich set of fast, efficient productivity features. These features can eliminate the up-front costs of writing and debugging your own code to perform various tasks, and will perform those tasks more efficiently. The functional capabilities of each of the features listed later in this section is described in detail elsewhere in this document. This section highlights the performance aspects of these features.

INCLUDE or OMIT, STOPAFT, and SKIPREC

You can use the INCLUDE or OMIT statement and the STOPAFT and SKIPREC options to reduce the number of records to be processed, which can reduce processor and data transfer time.

- The INCLUDE and OMIT statements allow you to select records by comparing fields with constants or other fields.
- The STOPAFT option allows you to specify the maximum number of records to be accepted for sorting or copying.
- The SKIPREC option allows you to skip records at the beginning of the input file and sort or copy only the remaining records.

OUTFIL

If you need to create multiple output data sets from the same input data set, you can use OUTFIL to read the input data set only once, thus improving performance. OUTFIL can be used for sort, merge, and copy applications to provide sophisticated filtering, editing, conversion, lookup and replace, and report features.

If you are creating only a single output data set and do not need the features of OUTFIL, use SORTOUT rather than OUTFIL for best performance.

LOCALE

You can use the LOCALE option to sort, merge, and compare character data based on collating rules in an active locale; this enables you to obtain results with DFSORT that were previously possible only through pre-processing or post-processing of your data. By eliminating such costly processing, you can save time and processing resources.

SUM

You can improve performance by using SUM to add the contents of fields. The SUM statement adds the contents of specified SUM fields in records with identical control fields. The result is placed in one record while the other record is deleted, thus reducing the number of records to be output by DFSORT.

You can use installation option ZDPRINT=YES or run-time option ZDPRINT to specify that positive zoned decimal fields that result from summing are to be printable. That is, you can tell DFSORT to change the last digit of the zone from hex C to hex F.

Eliminating duplicate records

You can eliminate records with duplicate keys by specifying

```
    SUM FIELDS=NONE
```

when using the SUM control statement.

For a diagram of the processing sequence for record handling statements, user exits, and options, see Figure 2 on page 10.

ICETOOL

ICETOOL is a multi-purpose utility that allows you to use DFSORT's highly efficient I/O and processing to perform multiple operations on one or more data sets in a single job step. ICETOOL's 17 operators allow you to perform sort, copy, statistical, and report operations quickly and efficiently.

Avoid options that degrade performance

Certain options can adversely affect performance, and should be used only when necessary. For example, the CKPT option, which activates checkpoint/restart, prevents use of the efficient Blockset techniques.

CKPT

The CKPT option might preclude the use of the more efficient Blockset technique.

Note: If installation option IGNCKPT=YES has been selected, DFSORT ignores the checkpoint/restart request and selects the Blockset technique.

EQUALS

The EQUALS option increases the time needed for comparison of records and for data transfer.

EQUCOUNT

The EQUCOUNT option takes additional time to count the number of records with equal keys.

LOCALE

The LOCALE option may increase the time required to run an application.

NOCINV

The NOCINV option precludes the use of control interval access for more efficient VSAM processing.

NOBLKSET

The NOBLKSET option precludes the use of the more efficient Blockset technique.

VERIFY

The VERIFY option degrades performance, because it involves extra processing.

Tape work data sets

Use of tape work data for intermediate storage precludes the use of the much more efficient disk techniques.

User exit routines

When user exit routines are included in an application, the time required to run the application is usually increased.

The run time required by most user exit routines is generally small, but the routines at user exits E15, E32, and E35 are entered for each record of the data sets. For large input data sets, the total run time of these routines can be relatively large.

Dynamic binding or link-editing

Dynamic binding or link-editing of user exit routines degrades performance.

EFS programs

When EFS programs are included in an application, the time required to run the application might increase.

Use main storage efficiently

In general, the more main storage you make available to DFSORT, the better the performance for large applications. To prevent excessive paging, ensure that sufficient real storage is available to back up the amount of main storage used. This is especially important with main storage sizes greater than 32MB. The default amount of main storage that will be made available to DFSORT is defined when it is installed.

Although it is possible to run a "very minimal" sort application in 88KB of storage, the actual minimum amount of storage required is generally 128KB to 440KB depending on the application. However, 4MB or more of storage is recommended for better performance. Improved performance will be most noticeable with large input files. With MAINSIZE=MAX, the DSA value can be used to automatically increase the amount of storage DFSORT uses for a sort application when doing so should improve performance.

Note: When Blockset is selected, DFSORT can place selected buffers above 16MB virtual. This frees more storage for DFSORT without having to increase the REGION size. A REGION size of at least 440KB must be available to allow DFSORT to use storage effectively.

Tuning main storage

Either the REGION value or the MAINSIZE/SIZE value can determine how much storage is available to DFSORT. See *z/OS DFSORT Installation and Customization* for details.

Generally, the most efficient way to allocate (virtual) main storage is to use MAINSIZE/SIZE=MAX explicitly or by default. However, problems can arise if the values for the TMAXLIM or MAXLIM installation options have been set excessively high (or low). Guidelines for setting these values are given in *z/OS DFSORT Installation and Customization*.

Note: Do not use SIZE/MAINSIZE=MAX with password-protected data sets if passwords are to be entered through a routine at a user exit, because DFSORT cannot then open the data sets during the initialization phase to make the necessary calculations.

If you specify MAINSIZE/SIZE=n and give n a value less than that specified for the MINLIM installation option, MINLIM is used.

When SIZE/MAINSIZE=MAX is in effect, DFSORT will use its Dynamic Storage Adjustment (DSA) feature, when appropriate, to improve performance.

If the MINLIM value is greater than that specified for REGION on the EXEC statement, DFSORT attempts to use the value specified for MINLIM; if it fails to get the amount specified by MINLIM, DFSORT still tries to run, provided at least 88KB (below 16MB virtual) are available to DFSORT.

Storage used for OUTFIL processing will be adjusted automatically, depending upon several factors, including:

- Total available storage
- Non-OUTFIL processing storage requirements
- Number of OUTFIL data sets and their attributes (for example, block size).

OUTFIL processing is subject to the ODMAXBF limit and your system storage limits (for example, IEFUSI) but not to DFSORT storage limits, that is, SIZE/MAINSIZE, MAXLIM, and TMAXLIM. DFSORT attempts to use storage above 16MB virtual for OUTFIL processing whenever possible.

Note:

1. In some cases, this release of DFSORT may use more storage than prior releases for comparable applications. This might affect the operation of some applications. For example, some applications that run as in-storage sorts (with no SORTWKdd data sets) in previous releases might not run in-storage when using this release. The amount of storage allocated is normally controlled by TMAXLIM. A REGION size of at least 440KB must be available if DFSORT is to achieve acceptable performance. The allocation of storage can be adversely affected if you have a smaller region value or if DFSORT needs to allocate buffers below 16MB virtual.
2. For extremely large sorts (for example, 2GB or more of data), make sure that Hipersorting, memory object sorting and dataspace sorting are enabled, and that 32MB or more of main storage is available to DFSORT.

The relationship between TMAXLIM, MAXLIM, MINLIM, and REGION might be described as a series of checks and balances.

Your system programmer has set the default storage values according to your site's major sorting requirements. If you have an overnight or batch time window that must be met, increasing storage (using REGION or SIZE/MAINSIZE=n) can give you some relief from the time constraint. If you are concerned with processor time, decreasing storage (using REGION or SIZE/MAINSIZE=n) can reduce the processor time associated with sorting small files.

In general, when you vary the amount of storage available to DFSORT, several things occur:

1. If you increase the amount of storage:
   - EXCPs are reduced.
   - For larger files, processor time generally decreases; that is, overhead in managing the extra storage is offset by DFSORT having to make fewer passes over the data.
   - For a very heavily loaded system, elapsed time might increase because DFSORT can be swapped out more often.
   - For very small sorts, processor time might remain stable or increase because of the overhead in managing the extra storage. For larger files, processor time will usually decrease because the overhead in managing the extra storage would be less than the benefit gained by DFSORT making fewer passes over the data.
2. If you decrease the amount of storage:
   - EXCPs increase.
   - Elapsed time increases for most sorts.
   - Processor time decreases for very small files, but increases for larger files.

Changing the main storage allocation can affect system efficiency. By reducing the amount of main storage allocated, you impair performance of DFSORT to allow other programs to have the storage they need to operate simultaneously; by increasing the allocation, you can run large DFSORT applications efficiently at the risk of decreasing the efficiency of other applications sharing the multiprogramming environment.

Releasing main storage

Under some circumstances, DFSORT uses all the available storage in your REGION. This normally will not occur for storage above 16MB virtual (if it does, use the ARESINV or ARESALL options or lower your SIZE/MAINSIZE value). This section explains how to release storage within your REGION.

When SIZE/MAINSIZE=n is in effect and n is greater than the REGION parameter or the default REGION value, or when SIZE/MAINSIZE=MAX and TMAXLIM is greater than your REGION, specify the storage you need released in the following way:

- For applications with user exits:
  - For directly invoked DFSORT, you can choose one of the following:
    - Use the **m** parameter of the MODS control statement.
    - If SIZE=MAX is in effect, you can use the RESALL option.
    - Change your REGION so that REGION is greater than SIZE/MAINSIZE (the difference is available).
    - If the OVERRGN installation option is smaller than your system IEFUSI value, this difference is available. (OVERRGN is an installation option that can be modified only by your system programmer.)
  - For program invoked DFSORT, you can choose one of the following:
    - If the user exit address is not passed in the parameter list (that is, it is specified with a MODS statement), use the **m** parameter on the MODS statement.
    - If the user exit address is passed in the parameter list,and SIZE/MAINSIZE=MAX is in effect, use the RESINV option.
    - If the user exit address is passed in the parameter list, and SIZE/MAINSIZE=n is in effect, change your REGION so that the REGION is greater than SIZE/MAINSIZE (the difference is available).
    -
    - If many of your DFSORT applications pass the user exit address in the parameter list and SIZE/MAINSIZE=n is in effect, then consider having the OVERRGN value changed by your system programmer to less than your IEFUSI value.
- For applications without user exits:
  - For directly invoked DFSORT, you can choose one of the following:
    - If SIZE/MAINSIZE=MAX is in effect, use the RESALL option.
    - If SIZE/MAINSIZE=n is in effect, change your REGION so that REGION is greater than SIZE/MAINSIZE (the difference is available).
    - Have the OVERRGN value changed by your system programmer to less than your IEFUSI value.
  - For program invoked DFSORT, you can choose one of the following:
    - If SIZE/MAINSIZE=MAX is in effect, use the RESINV option.
    - If SIZE/MAINSIZE=n is in effect, change your REGION so that REGION is greater than SIZE/MAINSIZE (the difference is available).
    - Have the OVERRGN value changed by your system programmer to less than your IEFUSI value.

When SIZE/MAINSIZE is less than REGION, make sure the difference between SIZE/MAINSIZE and your REGION specification value or default provides sufficient storage for system or user exit routine use.

Allocate temporary work space efficiently

Performance is enhanced when multiple channels are available. Performance is also improved if the device is connected so that two channel paths exist between each device and the processor that is running the program.

Disk work storage devices

Program performance is improved if you use devices, storage areas, and channels efficiently. If you specify a particular device type with the UNIT parameter on the DD statements that define intermediate storage data sets (for example, UNIT=3390), DFSORT assigns areas, and some optimization occurs automatically. You can get the best performance using disk intermediate storage devices when you:

- Use two or more work data sets.
- Select the storage device with the fastest data transfer rate.

- Assign one work data set per volume if Parallel Access Volumes (PAV) are not used. This will help minimize I/O queue time on the SORTWKdd data sets.
- Use devices that are of the same type.
- Use two channel paths to devices.
- Make all work data sets the same size, or as nearly the same size as possible.
- Make sure the SORTWKdd data sets do not share devices or channels with the SORTIN, SORTOUT, or OUTFIL data sets.
- Specify contiguous space for work data sets, and make sure there is enough primary space so that the automatic secondary allocation is not needed.
- Provide adequate virtual storage when work data sets are allocated on non-synchronous devices, as explained in "Non-synchronous storage subsystems" on page 798.

Elapsed time is decreased when DFSORT can both read input while writing to SORTWKdd and write output while reading from SORTWKdd. If, for example, you have two channels, the best allocation of them is to have SORTIN, SORTOUT, and OUTFIL data sets on one and the SORTWKdd data sets on the other.

Storage requirements for different disk techniques can be estimated by using the guidelines found in Appendix A, "Using work space," on page 797.

Virtual I/O for work data sets

Although VIO data sets can provide performance improvements in many applications, these work data sets are generally not as effective as disk work data sets for DFSORT.

DFSORT temporary work data sets allocated to virtual devices (VIO) can provide reduced elapsed time at the cost of increased CPU time for DFSORT applications. In general, this is not a good trade-off and VIO should not be used for DFSORT work data sets.

If work data sets are allocated to VIO, the VIO installation option tells DFSORT how to handle allocation to VIOs:

- VIO=YES causes DFSORT to accept the use of VIOs for work data sets.
- VIO=NO causes DFSORT to reallocate work data sets from virtual devices to real devices. Note that in order for re-allocation to be successful, a real device with the same device type as the virtual device must be available.

Re-allocation of VIO data sets (VIO=NO) is recommended over no re-allocation (VIO=YES). However, it is better to avoid using VIO work data sets in the first place, since re-allocation wastes time and resources.

Tape work storage devices

The use of tape work storage devices prevents the use of the more efficient Blockset technique. Best performance, using tape intermediate storage, is usually obtained when you use six or more tape drives of the fastest type. As a general rule, you should use as many tapes as you have available for intermediate storage. A larger number of tapes increases the number of strings that can be merged in one pass, and, therefore, decreases the number of passes required in the intermediate merge phase. This then reduces elapsed time and, often, the number of I/O operations.

Increasing the number of work units, however, also reduces the block size used for intermediate storage; this can become a critical factor if you have relatively little main storage available for buffers. For example, if DFSORT has only 88KB in which to operate, you probably achieve no improvement (and might find deterioration) if you use more than four tape work units. Therefore, apply the general rule of using as many tapes as possible only when DFSORT has more than 100KB available.

For information on how to determine storage requirements when using different tape techniques, see Appendix A, "Using work space," on page 797.

Note:

1. The frequency with which the tape direction changes during DFSORT work file operations has more of an impact on the effective data rate of the IBM 3480/3490/3590 Magnetic Tape Subsystems than

on IBM 3420 Magnetic Tape Units. Because of this characteristic, performance comparisons between these tape units for intermediate storage cannot be reliably predicted and can vary widely.

2. Devices using the Improved Data Recording Capability (IDRC) feature are not recommended as intermediate storage devices, as they do not perform well with the read backward command.

Use Hipersorting

Hipersorting uses Hiperspace to improve the performance of sort applications that use DFSORT's Blockset Technique. A Hiperspace is a high-performance data space that resides in central storage and is backed by auxiliary storage when necessary. With Hipersorting, Hiperspace is used in place of and along with disk for temporary storage of records during a Blockset sort. Hipersorting reduces I/O processing, which in turn reduces elapsed time, EXCPs, and channel usage. Hipersorting is recommended when the input or output is a compressed sequential or VSAM data set.

You can control the maximum amount of Hiperspace for a Hipersorting application with the HIPRMAX parameter. HIPRMAX can direct DFSORT to dynamically determine the maximum amount of Hiperspace, subject to the available storage at the start of the run. You can also use HIPRMAX to suppress Hipersorting when optimizing CPU time is your major concern because Hipersorting can slightly degrade CPU time.

The actual amount of Hiperspace a Hipersorting application uses depends upon several factors. See the HIPRMAX description in "OPTION control statement" on page 173 for more details. Most important, throughout the run, DFSORT determines the amount of available storage as well as the amount of storage needed by other concurrent Hipersorting, memory object sorting, and datapace sorting applications. Based on this information, DFSORT switches dynamically from using Hiperspace to using disk work data sets when either a storage shortage is predicted or the total Hipersorting, memory object sorting and dataspace sorting activity on the system reaches the limits set by the EXPMAX, EXPOLD, and EXPRES installation options. See *z/OS DFSORT Installation and Customization* for a complete description of these installation options.

Sort with data space

dataspace sorting uses data space to improve the performance of sort applications that use DFSORT's Blockset Technique.

dataspace sorting allows DFSORT to sort large pieces of data at a time. This helps to reduce CPU time and elapsed time.

You can control the maximum size of a data space for a dataspace sorting application with the DSPSIZE parameter. DSPSIZE can direct DFSORT to dynamically determine the maximum size of a data space, subject to the available central storage at the start of the run. DSPSIZE=0 means that DFSORT will not use dataspace sorting.

The actual size of a data space that a dataspace sorting application uses depends upon several factors. See the DSPSIZE description in "OPTION control statement" on page 173 for more details.

The following functions and types of data sets are not supported for dataspace sorting:

- Spool, dummy, or pipe data set, or z/OS UNIX file, as input.
- User exits
- INREC, OUTFIL, OUTREC, and SUM
- EQUCOUNT

Dataspace sorting is seldom used for very small data sets of a few MB or so because it is more efficient to sort small amounts of data entirely in main storage.

The following are actions you can take that might increase the use of dataspace sorting:

- Specify sufficient main storage. The default is 6MB, the recommended minimum for dataspace sorting. If you increase the amount of main storage specified, more dataspace sorting is possible, especially when sorting large amounts of data (multiple hundred MBs). Specifying more than 12MB or so will

have no significant impact on DFSORT's decision to use dataspace sorting; it will, however, improve the performance of large non-dataspace sorting applications.

- Specify generous extent sizes for work data sets, especially for secondary extents. Dataspace sorting is frequently used in conjunction with disk work space but never with Hiperspace or with tape work data sets.
- Specify DSPSIZE=MAX.
- Verify that IEFUSI does not place any restrictions on the size of the data spaces a single address space can create.
- Ensure that DFSORT has accurate information about the input file size. DFSORT can automatically estimate the file size for disk input data sets and tape data sets managed by DFSMSrmm or a tape management system that uses ICETPEX. See "File size and dynamic allocation" on page 800 for information on situations where DFSORT cannot determine the file size accurately, and what to do about it.

Use memory object sorting

Memory object sorting uses a memory object in 64-bit virtual storage to improve the performance of sort applications that use DFSORT's Blockset Technique. A memory object is a data area in virtual storage that is allocated above the bar and backed by central storage. With memory object sorting, a memory object is used in place of and along with disk for temporary storage of records during a sort. Memory object sorting reduces I/O processing, which in turn reduces elapsed time, EXCPs, and channel usage. Memory object sorting is recommended for large input data sets when a sufficient amount of central storage is available.

You can control the maximum size of a memory object for a memory object sorting application with the MOSIZE parameter. MOSIZE can direct DFSORT to dynamically determine the maximum size of a memory object, subject to the available central storage at the start of the run. MOSIZE=0 means that DFSORT will not use memory object sorting.

The actual size of a memory object that a memory object sorting application uses depends upon several factors. See the MOSIZE description in "OPTION control statement" on page 173 for more details.

The following are actions you can take that might increase the use of memory object sorting:

- Verify that a sufficient size for memory objects is defined by the MEMLIMIT parameter on the JOB or EXEC JCL statement.
- Specify MOSIZE=MAX.
- Specify generous extent sizes for work data sets, especially for secondary extents.
- Verify that IEFUSI does not place any restrictions on the size of the memory objects a single address space can create.
- Ensure that DFSORT has accurate information about the input file size. DFSORT can automatically estimate the file size for disk input data sets and tape data sets managed by DFSMSrmm or a tape management system that uses ICETPEX. See "File size and dynamic allocation" on page 800 for information on situations where DFSORT cannot determine the file size accurately, and what to do about it.

Use ICEGENER instead of IEBGENER

You can achieve more efficient processing for applications set up to use the IEBGENER system utility by using DFSORT's ICEGENER facility. Qualifying IEBGENER jobs are processed by the equivalent (though not identical), but more efficient, DFSORT copy function. If, for any reason, the DFSORT copy function cannot be used (for example, if IEBGENER control statements are specified), control is automatically transferred to the IEBGENER system utility.

ICEGENER, like IEBGENER, will use an SDB=value parameter you supply using PARM='SDB=value', when appropriate. The valid SDB=value parameters are SDB=LARGE, SDB=YES, SDB=SMALL, SDB=INPUT and SDB=NO, as explained in "OPTION control statement" on page 173. If you supply an invalid SDB=value parameter, ICEGENER will transfer control to IEBGENER, which will terminate due to the invalid parameter. If you do not supply an SDB=value parameter, ICEGENER will use your site's DFSORT

installation default for SDB, when appropriate (the IBM-supplied default is SDB=INPUT). If ICEGENER transfers control to IEBGENER, IEBGENER will use the SDB=value parameter you supply, if any, or its normal default for SDB.

ICEGENER will also recognize DFSORT parameters other than SDB=value you supply using PARM='parameter' that are valid on DFSORT's OPTION statement as explained in "OPTION control statement" on page 173. However, IEBGENER does not recognize any parameters other than the valid SDB=value forms, so if DFSORT must transfer control to IEBGENER, IEBGENER will not recognize DFSORT's parameters and will terminate. Likewise, if you supply a DFSORT parameter using PARM='parameter' that is not valid on DFSORT's OPTION statement, DFSORT will transfer control to IEBGENER and IEBGENER will terminate due to the invalid parameter.

For example, if you specify:

```
//S1 EXEC PGM=ICEGENER,PARM='SIZE=2000,MAINSIZE=2000K'
```

ICEGENER will accept SIZE=2000 and MAINSIZE=2000K as valid DFSORT OPTION parameters that specify an exact file size of 2000 records and a limit of 2000K bytes of storage, respectively. If DFSORT copy can be used, these parameters will be used. But if DFSORT must transfer control to IEBGENER, IEBGENER will terminate because it treats SIZE=2000 and MAINSIZE=2000K as invalid parameters.

As another example, if you specify:

```
//S2 EXEC PGM=ICEGENER,PARM='SIZE=2000K'
```

ICEGENER will treat SIZE=2000K as an invalid DFSORT OPTION parameter and will transfer control to IEBGENER, which will terminate because it treats SIZE=2000K as an invalid parameter.

Thus, you can pass PARM parameters to ICEGENER that are valid as DFSORT OPTION parameters, but you must be aware that if ICEGENER transfers control to IEBGENER, those parameters will cause IEBGENER to terminate. PARM parameters that are not valid as DFSORT OPTION parameters (even if they are valid as DFSORT PARM parameters) will cause ICEGENER to transfer control to IEBGENER, which will terminate.

ICEGENER can transfer control to IEBGENER due to DFSPARM or SORTCNTL statement errors or other errors detected by DFSORT. Therefore, we recommend that ICEGENER not be used for any application for which IEBGENER cannot be used, to avoid unwanted IEBGENER processing. For example, if ICEGENER is used with an INCLUDE statement in DFSPARM, IEBGENER could be used and complete successfully, but the INCLUDE statement would be ignored. Instead, DFSORT copy should be used directly so that IEBGENER cannot be called.

However, if you know that ICEGENER will use DFSORT copy, you can use a DFSPARM data set with ICEGENER to pass control statements and parameters to DFSORT. For example, if you specify:

```
//DFSPARM DD *
  OPTION SPANINC=RC0
/*
```

and ICEGENER uses DFSORT copy, any incomplete spanned records DFSORT detects in a variable spanned input data set are eliminated.

If your site has installed ICEGENER to be invoked by the name IEBGENER, you need not make any changes to your applications to use ICEGENER. If your site has not chosen automatic use of ICEGENER, you can use ICEGENER by substituting the name ICEGENER for IEBGENER on the EXEC statement (when DFSORT is directly invoked) or LINK macro (when DFSORT is program-invoked) in any applications you choose. Program-invoked applications must be recompiled.

Following is an example of how an IEBGENER application can be changed to use ICEGENER by substituting the name ICEGENER for the name IEBGENER in the EXEC statement.

```
//GENER JOB...
// EXEC PGM=ICEGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=CONTROL.MASTER,DISP=OLD,UNIT=3380,VOL=SER=MASTER
```

```
//SYSUT2 DD DSN=CONTROL.BACKUP,DISP=OLD,UNIT=3380,VOL=SER=BACKUP
//SYSIN DD DUMMY
```

The IEBGENER DD statements SYSUT1 (input), SYSUT2 (output), and SYSPRINT (messages) are used by DFSORT for SORTIN, SORTOUT, and SYSOUT, respectively. These DD statement names will be translated by using an extended parameter list to invoke the copy function. If DFSORT cannot be used (for example, because IEBGENER control statements are specified), control will be transferred to IEBGENER.

Note:

1. The SYSUT2 data set should not be the same as the SYSUT1 data set because this can cause lost or incorrect data or unpredictable results.
2. Whether ICEGENER is invoked from a program or not, DFSORT will be invoked from ICEGENER using an extended parameter list. Therefore, the installation defaults for the ICEAM2 or ICEAM4 environment, or for an ICETDx environment if activated for ICEAM2 or ICEAM4, apply and SORTCNTL or DFSPARM can be used to provide additional control statements for the copy application; for example, OPTION. However, ICEGENER can transfer control to IEBGENER due to DFSPARM or SORTCNTL statement errors or other errors detected by DFSORT. Therefore, DFSORT copy should be used directly rather than ICEGENER if DFSORT processing statements such as INCLUDE, OUTREC, SUM and so on are required.
3. For most error conditions that prevent the use of DFSORT copy, control will be transferred to the IEBGENER system utility. DFSORT messages will not be printed unless a SORTDIAG DD statement is supplied. Use of the SORTDIAG DD statement will allow you to determine why DFSORT copy could not be used.
4. If DFSORT copy is used, its operation and messages will be equivalent to a directly called DFSORT copy application. If an unrecoverable error is encountered (for example, an I/O error), DFSORT's return code of 16 will be changed by ICEGENER to a return code of 12 to emulate the return code from a failing IEBGENER application.
5. DFSORT copy can perform some functions not provided by IEBGENER, such as certain padding and truncation operations. ICEGENER processing is not identical to IEBGENER processing in all cases, since DFSORT copy uses methods to enhance performance (EXCP, for example) that are not used by IEBGENER.
6. In some cases, IEBGENER terminates when the SYSUT2 LRECL is different from the SYSUT1 LRECL. ICEGENER takes one of three actions depending on the GNPAD (LRECL padding) or GNTRUNC (LRECL truncation) installation option, as appropriate.

   If you want ICEGENER to transfer control to IEBGENER when the SYSUT2 LRECL is larger than the SYSUT1 LRECL, use installation option GNPAD=IEB. If you want ICEGENER to handle LRECL padding, use GNPAD=RC0 (the supplied default) or GNPAD=RC4.

   If you want ICEGENER to transfer control to IEBGENER when the SYSUT2 LRECL is smaller than the SYSUT1 LRECL, use installation option GNTRUNC=IEB. If you want ICEGENER to handle LRECL truncation, use GNTRUNC=RC0 (the supplied default) or GNTRUNC=RC4.
7. For a call to ICEGENER, or to IEBGENER as an alias for ICEGENER, register 1 must point to a valid parameter list consisting of three addresses as follows:

   - Address1: The address of the Option List.
   - Address2: The address of the Alternate DDname List.
   - Address3: The address of the Page Number List.

   Methods of calling ICEGENER that generate a valid parameter list will allow ICEGENER to use DFSORT's copy feature, whereas methods of calling ICEGENER that generate an invalid parameter list will cause ICEGENER to transfer control to IEBGENER. For example:

   ```
   call *(icegener)
   ```

   on the TSO command line generates a valid parameter list, whereas:

   ```
   icegener
   ```

on the TSO command line generates an invalid parameter list.

ICEGENER return codes

ICEGENER can use either IEBGENER or the DFSORT copy function. However, for unsuccessful completion due to an unsupported operating system, ICEGENER passes back a return code of 24 to the operating system or the invoking program, without using either IEBGENER or DFSORT copy.

If ICEGENER transfers control to IEBGENER, IEBGENER passes back its return code to the operating system or the invoking program.

If ICEGENER uses the DFSORT copy function:

- For successful completion, ICEGENER passes back a return code of 0 or 4 to the operating system or the invoking program.
- For unsuccessful completion with NOABEND in effect, ICEGENER passes back a return code of 12 (changed from 16) to the operating system or the invoking program.
- For unsuccessful completion with ABEND in effect, DFSORT issues a user abend with the appropriate code as specified by the ABCODE installation option (either the error message number or a number between 1 and 99).

The meanings of the return codes that ICEGENER passes back (in register 15) are:

0

Successful completion. ICEGENER completed successfully.

4

Successful completion. ICEGENER completed successfully, and:

- Installation option GNPAD=RC4 was specified and the SYSUT2 LRECL was larger than the SYSUT1 LRECL (LRECL padding) or
- Installation option GNTRUNC=RC4 was specified and the SYSUT2 LRECL was smaller than the SYSUT1 LRECL (LRECL truncation), or
- SPANINC=RC4 was in effect and one or more incomplete spanned records was detected, or
- NULLOUT=RC4 was in effect and there were no records for the SYSUT2 data set, or
- NULLOFL=RC4 was in effect and there were no data records for an OUTFIL data set.

12

Unsuccessful completion. DFSORT detected an error that prevented ICEGENER from completing successfully.

24

Unsupported operating system. This operating system is not supported by this release of DFSORT.

Use DFSORT's performance booster for The SAS System

DFSORT provides significant CPU time improvements for SAS applications. To take advantage of this feature, contact SAS Institute Inc. for details of the support they provide to enable this enhancement.

Use DFSORT's BLDINDEX support

DFSORT provides support that enables IDCAMS BLDINDEX to automatically use DFSORT to improve the performance of most BLDINDEX jobs that require BLDINDEX external sorting.

Chapter 11. Examples of DFSORT job streams

Summary of examples

The following table summarizes the examples provided in this chapter.

Summary of the examples of DFSORT Job streams provided in this chapterApplication	No.	Input	Output	Functions/Options
Sort	1	Disk	Tape	ALTSEQ
Sort	2	Disk	Disk	OMIT, SUM, OUTREC, DYNALLOC, ZDPRINT
Sort	3	Tape	Tape	ASCII Tapes
Sort	4	Tape	Disk	E15, E35, FILSZ, AVGRLEN, DYNALLOC
Sort	5	Disk	Disk	Program-invoked, SORTCNTL, CHALT, DYNALLOC, FILSZ
Sort	6	Disk	Disk	VSAM Input/Output, DFSPARM, Option Override
Sort	7	Disk	Disk	COBOL E15, EXEC PARM, MSGDDN
Sort	8	Disk	Disk	Dynamic Link-editing of Exits
Sort	9	E15	Disk	Extended Parameter List Interface
Sort	10	Disk	Disk and SYSOUT	OUTFIL
Sort	11	Pipe	Pipes	Pipes, OUTFIL SPLIT, FILSZ, DYNALLOC
Sort	12	Disk	Disk	INCLUDE, LOCALE
Sort	13	z/OS UNIX files	z/OS UNIX file	
Sort	14	Disk	Disk	IFTHEN
Sort	15	E15	Disk	64-bit parameter list interfaces
Merge	1	Disk	Disk	EQUALS
Merge	2	Disk	Disk	LOCALE, OUTFIL
Copy	1	Tape	Disk	EXEC PARMs, SKIPREC, MSGPRT, ABEND
Copy	2	Disk	Disk	INCLUDE, VLSHRT
Copy	3	Disk	Disk	OUTREC, PARSE, BUILD
ICEGENER	1	Disk	Disk	
ICETOOL	1	Disk	Disk	OCCUR, COPY, SORT, MODE, VERIFY, STATS, DISPLAY

Storage administrator examples

DFSORT provides a set of sample jobs that demonstrate techniques of interest to Storage Administrators and others who analyze data collected from DFSMShsm, DFSMSrmm, DCOLLECT and SMF. These sample jobs can be found in the ICESTGEX member of the SICESAMP library (contact your System Programmer for details). You can also download these sample jobs from the DFSORT FTP site. These sample jobs show some of the many ways DFSORT features such as ICETOOL and OUTFIL can be used to analyze data and generate reports:

DCOLEX1
DCOLLECT Example 1: VSAM report

DCOLEX2
DCOLLECT Example 2: Conversion reports

DCOLEX3
DCOLLECT Example 3: Capacity planning analysis and reports

DFHSMEX1
DFHSM Example 1: Deciphering Activity Logs

DFHSMEX2
DFHSM Example 2: Recover a DFHSM CDS with a broken index

RMMEX1
DFSMSrmm Example 1: SMF audit report

RMMEX2
DFSMSrmm Example 2: Create ADDVOLUME commands

REXX examples

Both DFSORT and ICETOOL can be called from REXX. The key is to specify ALLOCATE statements for the data sets you need and then use an ADDRESS statement like this:

```
  ADDRESS LINKMVS name
```

which says to fetch the named program using the standard system search list.

Here is an example of a REXX CLIST to call DFSORT:

```
/* Simple REXX CLIST to call DFSORT */
  "FREE FI(SYSOUT SORTIN SORTOUT SYSIN)"
  "ALLOC FI(SYSOUT)   DA(*)"
  "ALLOC FI(SORTIN)   DA('Y897797.INS1') REUSE"
  "ALLOC FI(SORTOUT)  DA('Y897797.OUTS1') REUSE"
  "ALLOC FI(SYSIN)    DA('Y897797.SORT.STMTS') SHR REUSE"
  ADDRESS LINKMVS ICEMAN
```

Here are the DFSORT control statements that might appear in the Y897797.SORT.STMTS data set:

```
  SORT FIELDS=(5,4,CH,A)
  INCLUDE COND=(21,3,SS,EQ,C'L92,J82,M72')
```

Here is an example of a REXX CLIST to call ICETOOL:

```
/* Simple REXX CLIST to call ICETOOL */
  "FREE FI(TOOLMSG DFSMSG VLR LENDIST TOOLIN)"
  "ALLOC FI(TOOLMSG)   DA(*)"
  "ALLOC FI(DFSMSG)   DUMMY"
  "ALLOC FI(VLR)   DA('Y897797.VARIN') REUSE"
  "ALLOC FI(LENDIST)   DA(*)"
  "ALLOC FI(TOOLIN)    DA('Y897797.TOOLIN.STMTS') SHR REUSE"
  ADDRESS LINKMVS ICETOOL
```

Here are the ICETOOL statements that might appear in the Y897797.TOOLIN.STMTS data set:

```
 OCCURS FROM(VLR) LIST(LENDIST) -
   TITLE('LENGTH DISTRIBUTION REPORT') BLANK -
   HEADER('LENGTH')  HEADER('NUMBER OF RECORDS') -
   ON(VLEN)          ON(VALCNT)
```

CLIST examples

Both DFSORT and ICETOOL can be called from a CLIST. They key is to specify ALLOCATE statements for the data sets you need and then use a CALL statement like this:

```
   CALL *(name)
```

Here is an example of a CLIST to call DFSORT:

```
FREE FI(SYSOUT SORTIN SORTOUT SYSIN)
ALLOC FI(SYSOUT) DA(*)
ALLOC FI(SORTIN) DA('Y897797.INS1') REUSE
ALLOC FI(SORTOUT) DA('Y897797.OUTS1') REUSE
ALLOC FI(SYSIN) DA('Y897797.SORT.STMTS') SHR REUSE
CALL *(ICEMAN)
```

Here are the DFSORT control statements that might appear in the Y897797.SORT.STMTS data set:

```
   SORT FIELDS=(5,4,CH,A)
   INCLUDE COND=(21,3,SS,EQ,C'L92,J82,M72')
```

Here is an example of a CLIST to call ICETOOL:

```
FREE FI(TOOLMSG DFSMSG VLR LENDIST TOOLIN)
ALLOC FI(TOOLMSG) DA(*)
ALLOC FI(DFSMSG) DUMMY
ALLOC FI(VLR) DA('Y897797.VARIN') REUSE
ALLOC FI(LENDIST) DA(*)
ALLOC FI(TOOLIN)  DA('Y897797.TOOLIN.STMTS') SHR REUSE
CALL *(ICETOOL)
```

Here are the ICETOOL statements that might appear in the Y897797.TOOLIN.STMTS data set:

```
   OCCURS FROM(VLR) LIST(LENDIST) -
     TITLE('LENGTH DISTRIBUTION REPORT') BLANK -
     HEADER('LENGTH')  HEADER('NUMBER OF RECORDS') -
     ON(VLEN)          ON(VALCNT)
```

Sort examples

This section includes 14 sort examples.

Example 1. Sort with ALTSEQ

INPUT
Blocked variable-length records on disk

OUTPUT
Blocked variable-length records on 3490

WORK DATA SETS
Two 3390 data sets

USER EXITS
None

FUNCTIONS/OPTIONS
ALTSEQ

```
//EXAMP    JOB A400,PROGRAMMER                                    01
//S1       EXEC PGM=SORT                                          02
//SYSOUT   DD SYSOUT=A                                            03
//SORTIN   DD DSN=A123456.IN5,DISP=SHR                            04
//SORTOUT  DD DSN=OUT1,UNIT=3490,DISP=(,KEEP),VOL=SER=VOL001      05
//SORTWK01 DD UNIT=3390,SPACE=(CYL,(10,10))                       06
//SORTWK02 DD UNIT=3390,SPACE=(CYL,(10,10))                       07
//SYSIN    DD *                                                   08
* COLLATE $, # and @ AFTER Z                                     09
  SORT FIELDS=(7,5,AQ,A)                                         10
  ALTSEQ CODE=(5BEA,7BEB,7CEC)                                   11
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls DFSORT directly by its alias SORT.

03
SYSOUT DD statement. Directs DFSORT messages and control statements to system output class A.

04
SORTIN DD statement. The input data set is named A123456.IN5 and is cataloged. DFSORT determines from the data set label that the RECFM is VB, the maximum LRECL is 120, and the BLKSIZE is 2200.

05
SORTOUT DD statement. The output data set is named OUT1 and is to be allocated on 3490 volume VOL001 and kept. DFSORT sets the RECFM and LRECL from SORTIN and selects an appropriate BLKSIZE for this standard labeled tape.

06
SORTWK01 DD statement. The first work data set is allocated on 3390.

07
SORTWK02 DD statement. The second work data set is allocated on 3390.

08
SYSIN DD statement. DFSORT control statements follow.

09
Comment statement. Printed but otherwise ignored.

10
SORT statement. FIELDS specifies an ascending 5-byte character control field starting at position 7 (the third data byte, because the RDW occupies the first 4 bytes). The control field is to be collated according to the modified sequence described in the ALTSEQ statement.

11
ALTSEQ statement. CODE specifies that the three characters $, # and @ are to collate in that order after Z.

Example 2. Sort with OMIT, SUM, OUTREC, DYNALLOC and ZDPRINT

INPUT
Blocked fixed-length records on 3380 and 3390

OUTPUT
Blocked fixed-length records on 3390

WORK DATA SETS
Dynamically allocated

USER EXITS
None

FUNCTIONS/OPTIONS
OMIT, OUTREC, SUM, DYNALLOC, ZDPRINT

```
//EXAMP    JOB A400,PROGRAMMER                                    01
//STEP1    EXEC PGM=SORT                                          02
//SYSOUT   DD SYSOUT=H                                            03
//SORTIN   DD DSN=INP1,DISP=SHR,UNIT=3380,VOL=SER=SCR001          04
//         DD DSN=INP2,DISP=SHR,UNIT=3390,VOL=SER=SYS351          05
//SORTOUT  DD DSN=&&OUTPUT,DISP=(,PASS),UNIT=3390,                06
//   SPACE=(CYL,(5,1)),DCB=(LRECL=22)                             07
//SYSIN    DD *                                                   08
  OMIT COND=(5,1,CH,EQ,C'M')                                     09
  SORT FIELDS=(20,8,CH,A,10,3,FI,D)                              10
  SUM FIELDS=(16,4,ZD)                                           11
  OPTION DYNALLOC,ZDPRINT                                        12
  OUTREC FIELDS=(10,3,20,8,16,4,2Z,5,1,C' SUM')                  13
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls DFSORT directly by its alias SORT.

03
SYSOUT DD statement. Directs DFSORT messages and control statements to system output class H.

04-05
SORTIN DD statement. Consists of a concatenation of two data sets. The first input data set is named INP1 and resides on 3380 volume SCR001. The second input data set is named INP2 and resides on 3390 volume SYS351. DFSORT determines from the data set labels that the record format is FB, the LRECL is 80 and the largest BLKSIZE is 27920.

06-07
SORTOUT DD statement. The output data set is temporary and is to be allocated on a 3390. Because the OUTREC statement results in a reformatted output record length of 22 bytes, LRECL=22 must be specified. DFSORT sets the RECFM from SORTIN and selects an appropriate BLKSIZE.

08
SYSIN DD statement. DFSORT control statements follow.

09
OMIT statement. COND specifies that input records with a character M in position 5 are to be omitted from the output data set.

10
SORT statement. FIELDS specifies an ascending 8-byte character control field starting at position 20 and a descending 3-byte fixed-point control field starting at position 10.

11
SUM statement. FIELDS specifies a 4-byte zoned-decimal summary field starting at position 16. Whenever two records with the same control fields (specified in the SORT statement) are found, their summary fields (specified in the SUM statement) are to be added and placed in one of the records, and the other record is to be deleted.

12
OPTION statement. DYNALLOC specifies that work data sets are to be dynamically allocated using the installation defaults for the type of device and number of devices. ZDPRINT specifies that positive ZD SUM fields are to be printable.

13
OUTREC statement. FIELDS specifies how the records are to be reformatted for output. The reformatted records are 22 bytes long and look as follows:

Position
Content

1-3
Input positions 10 through 12

4-11
Input positions 20 through 27

12-15
Input positions 16 through 19

16-17
Zeros

18
Input position 5

19-22
The character string ' SUM'

Example 3. Sort with ASCII tapes

INPUT
Variable-length ASCII records on 3590

OUTPUT
Variable-length ASCII records on 3590

WORK DATA SETS
One SYSDA data set

USER EXITS
None

FUNCTIONS/OPTIONS
None

```
//EXAMP    JOB A400,PROGRAMMER                                    01
//RUNIT    EXEC SORTD                                             02
//SORTIN   DD DSN=SRTFIL,DISP=(OLD,DELETE),UNIT=3590,             03
//  DCB=(RECFM=D,LRECL=400,BLKSIZE=404,OPTCD=Q,BUFOFF=L),        04
//  VOL=SER=311500,LABEL=(1,AL)                                    05
//SORTOUT  DD DSN=OUTFIL,UNIT=3590,LABEL=(,AL),DISP=(,KEEP),     06
//  DCB=(BLKSIZE=404,OPTCD=Q,BUFOFF=L),VOL=SER=311501            07
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(4))                         08
//SYSIN    DD *                                                  09
  SORT FIELDS=(10,8,AC,D)                                        10
  RECORD TYPE=D,LENGTH=(,,,20,80)                                11
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Uses the SORTD cataloged procedure to call DFSORT directly.

03-05
SORTIN DD statement. The input data set is named SRTFIL and resides on 3590 volume 311500. It is to be deleted after this job step. It has a RECFM of D (variable-length ASCII records), a maximum LRECL of 400, a BLKSIZE of 404 and an ASCII label. For this job, the buffer offset is the block length indicator. The records are to be translated from ASCII to EBCDIC.

06-07
SORTOUT DD statement. The output data set is named OUTFIL and is to be allocated on 3590 volume 311501 and kept. It is to be written with an ASCII label. DFSORT sets the RECFM and LRECL from SORTIN and sets the BLKSIZE to 404 as indicated in the DD statement. For this job, the buffer offset is the block length indicator. The records are to be translated from EBCDIC to ASCII.

08
SORTWK01 DD statement. The work data set is allocated on SYSDA.

09
SYSIN DD statement. DFSORT control statements follow.

10
SORT statement. FIELDS specifies a descending 8-byte ASCII control field starting at position 10.

11
RECORD statement. TYPE specifies ASCII variable-length records. LENGTH specifies that the minimum record length is 20 and the average record length is 80.

Example 4. Sort with E15, E35, FILSZ, AVGRLEN and DYNALLOC

INPUT
Variable-length records on 3490

OUTPUT
Blocked variable-length records on SYSDA

WORK DATA SETS
Dynamically allocated

USER EXITS
E15 and E35

FUNCTIONS/OPTIONS
FILSZ, AVGRLEN, DYNALLOC

```
//EXAMP    JOB A400,PROGRAMMER                                    01
//STEP1    EXEC PGM=ICEMAN                                        02
//SYSOUT   DD SYSOUT=A                                            03
//SORTIN   DD DSN=INPUT,VOL=SER=FLY123,                           04
//  UNIT=3490,DISP=OLD                                            05
//SORTOUT  DD DSN=&&OUT,DISP=(,PASS),SPACE=(CYL,(10,12)),         06
//  UNIT=SYSDA,DCB=(RECFM=VB)                                     07
//MODLIB   DD DSN=EXIT1.RTNS,DISP=SHR                             08
//         DD DSN=EXIT2.RTNS,DISP=SHR                             09
//SYSIN    DD *                                                   10
  SORT FIELDS=(23,4,PD,A,10,6,FS,A)                              11
  OPTION DYNALLOC=(3390,3),AVGRLEN=75,FILSZ=E50000               12
  MODS E15=(MODREC,1024,MODLIB),E35=(ADDREC,1200,MODLIB)         13
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls DFSORT directly.

03
SYSOUT DD statement. Directs DFSORT messages and control statements to system output class A.

04-05
SORTIN DD statement. The input data set is named INPUT and resides on 3490 volume FLY123. DFSORT determines from the data set label of this standard labeled tape that the RECFM is V, the LRECL is 120 and the BLKSIZE is 124.

06-07
SORTOUT DD statement. The output data set is temporary and is to be allocated on SYSDA. Because the input is unblocked and the output is to be blocked, RECFM=VB must be specified. DFSORT sets the LRECL from SORTIN and selects an appropriate BLKSIZE.

08-09
MODLIB DD statement. Specifies the load libraries that contain the exit routines. When exit routines reside in more than one library, the libraries must be concatenated using a single DD statement.

10
SYSIN DD statement. DFSORT control statements follow.

11
SORT statement. FIELDS specifies an ascending 4-byte packed-decimal control field starting at position 23 and an ascending 6-byte floating-sign control field starting at position 10.

12
OPTION statement. DYNALLOC=(3390,3) specifies that three 3390 work data sets are to be allocated. AVGRLEN=75 specifies an average record length of 75. AVGRLEN helps DFSORT optimize work space for variable-length record input. FILSZ=E50000 specifies an estimate of 50000 records. Because the 3490 input data set is compacted, DFSORT might not be able to determine the file size accurately unless the data set is managed by DFSMSrmm or a tape management system that uses ICETPEX. Specification of FILSZ can make a significant difference in work space optimization when tape input data sets are not managed.

13
MODS statement. E15 specifies a user exit routine named MODREC. Approximately 1024 bytes are required for MODREC and the system services (for example, GETMAIN and OPEN) it performs. E35 specifies a user exit routine named ADDREC. Approximately 1200 bytes are required for ADDREC and

the system services it performs. MODREC and ADDREC reside in the libraries defined by the MODLIB DD statement.

Example 5. Called sort with SORTCNTL, CHALT, DYNALLOC and FILSZ

INPUT
Blocked fixed-length records on disk

OUTPUT
Blocked fixed-length records on disk

WORK DATA SETS
Dynamically allocated

USER EXITS
None

FUNCTIONS/OPTIONS
CHALT, DYNALLOC, FILSZ

```
//EXAMP    JOB A400,PROGRAMMER                                    01
//RUNSORT  EXEC PGM=MYPGM                                         02
//STEPLIB  DD DSN=M999999.LOAD,DISP=SHR                           03
//SYSOUT   DD SYSOUT=A                                            04
//SYSPRINT DD SYSOUT=A                                            05
//SORTIN   DD DSN=M999999.INPUT(MASTER),DISP=OLD                  06
//SORTOUT  DD DSN=M999999.OUTPUT.FILE,DISP=OLD                    07
//SORTCNTL DD *                                                   08
   OPTION  CHALT,DYNALLOC=(,3),FILSZ=U25000                       09
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls a program named MYPGM that in turn calls DFSORT.

03
STEPLIB DD statement. Specifies the load library that contains MYPGM.

04
SYSOUT DD statement. Directs DFSORT messages and control statements to system output class A.

05
SYSPRINT DD statement. Directs MYPGM output to system output class A.

06
SORTIN DD statement. The input data set is member MASTER in the cataloged partitioned data set M999999.INPUT. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

07
SORTOUT DD statement. The output data set is named M999999.OUTPUT.FILE and is cataloged. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

08
SORTCNTL DD statement. DFSORT control statements follow. Statements in SORTCNTL override or supplement statements passed by MYPGM in the DFSORT parameter list it uses.

09
OPTION statement. CHALT specifies that character format control fields (specified in the SORT statement passed by MYPGM) are to be sorted using the ALTSEQ installation option. DYNALLOC=(,3) specifies that three work data sets are to be dynamically allocated using the installation default for the type of device. FILSZ=U25000 specifies a file size of 25000 records is to be used by DFSORT to determine the amount of work space needed. Because the input data set is a member of a PDS, specifying FILSZ helps DFSORT optimize work data set space.

Example 6. Sort with VSAM input/output, DFSPARM and option override

INPUT
VSAM TYPE=V records

OUTPUT
VSAM TYPE=V records

WORK DATA SETS
Dynamically allocated

USER EXITS
None

FUNCTIONS/OPTIONS
Override of Various Options

```
//EXAMP    JOB A400,PROGRAMMER                                    01
//S1       EXEC PGM=SORT                                          02
//SYSOUT   DD SYSOUT=A                                            03
//SORTIN   DD DSN=TEST.SORTIN.FILE,DISP=SHR                       04
//SORTOUT  DD DSN=TEST.SORTOUT.FILE,DISP=SHR                      05
//DFSPARM  DD *                                                   06
  RECORD TYPE=V                                                   07
  SORT FIELDS=(30,4,BI,A)                                         08
  OPTION HIPRMAX=10,DYNALLOC=3390,MAINSIZE=3M,                    09
    MSGPRT=CRITICAL,NOLIST                                        10
```

For purposes of illustration, assume that none of the standard installation defaults for batch direct invocation of DFSORT have been changed by the site.

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls DFSORT directly by its alias SORT.

03
SYSOUT DD statement. Directs DFSORT messages and control statements to system output class A.

04
SORTIN DD statement. The input data set is TEST.SORTIN.FILE. DFSORT determines that it is a VSAM data set and obtains its attributes from the catalog.

05
SORTOUT DD statement. The output data set is TEST.SORTOUT.FILE. DFSORT determines that it is a VSAM data set and obtains its attributes from the catalog.

06
DFSPARM DD statement. DFSORT control statements follow. DFSPARM can be used for both direct-invocation and program-invocation of DFSORT and overrides options and statements from all other sources. Certain operands, such as MSGPRT and LIST/NOLIST, are used if supplied in DFSPARM, the EXEC PARM or the invocation parameter list, but not used if supplied in SYSIN or SORTCNTL.

07
RECORD statement. TYPE=V specifies that DFSORT is to treat the VSAM records as variable-length. In this case, the RECORD statement could be omitted, because DFSORT would automatically set a record type of V due to the use of VSAM data sets for SORTIN and SORTOUT.

08
SORT statement. FIELDS specifies an ascending 4-byte binary control field starting at position 30. This position corresponds to a specification of KEYS(4 25) for the VSAM CLUSTER (4 bytes at offset 25, which is equivalent to position 26 with 4 bytes added for the RDW that DFSORT supplies at input and removes at output for VSAM TYPE=V records).

09-10
OPTION statement. HIPRMAX=10 specifies that up to 10 MBs of Hiperspace can be used for Hipersorting, overriding the standard installation default of HIPRMAX=OPTIMAL. DYNALLOC=3390 specifies that work data sets are to be allocated on 3390s, overriding the standard installation default of SYSDA. The standard installation default of four work data sets is not overridden. MAINSIZE=3M specifies that up to 3 MBs of storage can be used, overriding the standard installation default of MAINSIZE=MAX. MSGPRT=CRITICAL specifies that only error messages are to be printed, overriding the standard installation default of MSGPRT=ALL. NOLIST specifies that control statements are not to be printed, overriding the standard installation default of LIST=YES.

Example 7. Sort with COBOL E15, EXEC PARM and MSGDDN

INPUT
Fixed-length records on disk

OUTPUT
Fixed-length records on SYSDA

WORK DATA SETS
None

USER EXITS
COBOL E15

FUNCTIONS/OPTIONS
MSGDDN

```
//EXAMP    JOB A400,PROGRAMMER                                    01
//STEP1    EXEC PGM=SORT,PARM='MSGDDN=DFSOUT'                     02
//STEPLIB  DD DSN=SYS1.SCEERUN,DISP=SHR                           03
//SYSOUT   DD SYSOUT=A                                            04
//DFSOUT   DD SYSOUT=A                                            05
//EXITC    DD DSN=COBEXITS.LOADLIB,DISP=SHR                       06
//SORTIN   DD DSN=SORT1.IN,DISP=SHR                               07
//SORTOUT  DD DSN=&&OUT,DISP=(,PASS),SPACE=(CYL,(5,5)),           08
//  UNIT=SYSDA,DCB=(LRECL=120)                                    09
//SYSIN    DD *                                                   10
  SORT FIELDS=(5,4,A,22,2,A),FORMAT=BI                           11
  MODS E15=(COBOLE15,37000,EXITC,C)                              12
  RECORD LENGTH=(,120)                                           13
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls DFSORT directly by its alias name SORT. MSGDDN=DFSOUT specifies an alternate message data set for DFSORT messages and control statements to prevent the COBOL messages in SYSOUT from being interleaved with the DFSORT messages and control statements.

03
STEPLIB statement. Specifies the Language Environment library.

04
SYSOUT statement. Directs COBOL messages to system output class A.

05
DFSOUT statement. Directs DFSORT messages and control statements to system output class A (this is the alternate message data set specified by MSGDDN in the PARM field of the EXEC statement).

06
EXITC statement. Specifies the load library that contains the exit routine.

07
SORTIN DD statement. The input data set is named SORT1.IN and is cataloged. DFSORT determines from the data set label that the RECFM is F, the LRECL is 100 and the BLKSIZE is 100.

08-09
SORTOUT DD statement. The output data set is temporary and is to be allocated on SYSDA. Because the E15 exit changes the length of the records from 100 bytes to 120 bytes, LRECL=120 must be specified. DFSORT sets the RECFM from SORTIN and sets the BLKSIZE to the LRECL (unblocked records).

10
SYSIN DD statement. DFSORT control statements follow.

11
SORT statement. FIELDS specifies an ascending 4-byte control field starting at position 5 and an ascending 2-byte control field starting at position 22. FORMAT specifies that the control fields are binary.

12
MODS statement. E15 specifies a user exit routine named COBOLE15 written in COBOL. Approximately 37000 bytes are required for the exit, the system services (for example, GETMAIN and OPEN) it performs, and the COBOL library subroutines. COBOLE15 resides in the library defined by the EXITC DD statement.

13
RECORD statement. LENGTH specifies that the COBOL E15 routine changes the length of the records to 120 bytes.

Example 8. Sort with dynamic link-editing of exits

INPUT
Blocked fixed-length records on disk

OUTPUT
Blocked fixed-length records on 3380

WORK DATA SETS
One SYSDA data set

USER EXITS
E11, E15, E17, E18, E19, E31, E35, E38, E39

FUNCTIONS/OPTIONS
None

```
//EXAMP    JOB A400,PROGRAMMER                                  01
//STEPA    EXEC SORT                                            02
//SORTIN   DD DSN=SMITH.INPUT,DISP=SHR                          03
//SORTOUT  DD DSN=SMITH.OUTPUT,DISP=(NEW,CATLG),                04
//  UNIT=3380,SPACE=(TRK,(10,2)),VOL=SER=XYZ003                 05
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1))                      06
//EXIT     DD DSN=SMITH.EXIT.OBJ,DISP=SHR                       07
//EXIT2    DD DSN=SMITH.EXIT2.OBJ,DISP=SHR                      08
//SORTMODS DD UNIT=SYSDA,SPACE=(TRK,(10,,3))                    09
//SYSIN    DD *                                                 10
  SORT FIELDS=(1,8,CH,A,20,4,BI,D)                              11
  MODS E11=(EXIT11,1024,EXIT,S),                                12
       E15=(E15,1024,SYSIN,T),                                  13
       E17=(EXIT17,1024,EXIT2,T),                               14
       E18=(EXIT18,1024,EXIT,T),                                15
       E19=(E19,1024,SYSIN,T),                                  16
       E31=(PH3EXIT,1024,EXIT,T),                               17
       E35=(PH3EXIT,1024,EXIT,T),                               18
       E38=(PH3EXIT,1024,EXIT,T),                               19
       E39=(E39,1024,SYSIN,T)                                   20
  END                                                           21
<object deck for E15 exit here>                                 22
<object deck for E19 exit here>                                 23
<object deck for E39 exit here>                                 24
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Uses the SORT cataloged procedure to call DFSORT directly and supply the DD statements (not shown) required by the linkage editor.

03
SORTIN DD statement. The input data set is named SMITH.INPUT and is cataloged. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

04-05
SORTOUT DD statement. The output data set is named SMITH.OUTPUT and is to be allocated on 3380 volume XYZ003 and cataloged. DFSORT sets the RECFM and LRECL from SORTIN and selects an appropriate BLKSIZE.

06
SORTWK01 DD statement. The work data set is allocated on SYSDA.

07
EXIT DD statement. Specifies the partitioned data set containing the object decks for the E11, E18, E31, E35 and E38 exit routines.

08
EXIT2 DD statement. Specifies the partitioned data set containing the object deck for the E17 exit routine.

09
SORTMODS DD statement. The partitioned data set to hold exit routine object decks from SYSIN for input to the linkage editor is to be allocated on SYSDA.

10
SYSIN DD statement. DFSORT control statements, and object decks to be used by the linkage editor, follow.

11
SORT statement. FIELDS specifies an ascending 8-byte character control field starting at position 1 and a descending 4-byte binary control field starting at position 20.

12-20
MODS statement. Specifies the exit routines to be used, the approximate number of bytes required for each exit and that:

- The EXIT11 routine in the EXIT library is to be link-edited separately from other input phase exit routines and associated with user exit E11.
- The E15 and E19 routines in SYSIN, the EXIT17 routine in EXIT2, and the EXIT18 routine in EXIT are to be link-edited together and associated with user exits E15, E19, E17, and E18, respectively.
- The E31, E35, and E38 routines in the PH3EXIT object deck and the E39 routine in SYSIN are to be link-edited together and associated with user exits E31, E35, E38, and E39, respectively.

21
END statement. Marks the end of the DFSORT control statements and the beginning of the exit routine object decks.

22-24
Object decks. The three object decks for the E15, E19, and E39 exit routines follow the END statement.

Example 9. Sort with the extended parameter list interface

INPUT
Fixed-length records from E15

OUTPUT
Blocked fixed-length records on SYSDA

WORK DATA SETS
Dynamically allocated

USER EXITS
E15

FUNCTIONS/OPTIONS
OMIT, FILSZ, RESINV, DYNALLOC

The JCL for running program MYSORT, and highlights of the code used by MYSORT to invoke DFSORT with the extended parameter list, are shown below. For purposes of illustration, assume that none of the standard installation defaults for batch program invocation of DFSORT have been changed by the site.

```
//EXAMP    JOB A400,PROGRAMMER                                    01
//STEP1    EXEC PGM=MYSORT                                        02
//SYSOUT   DD SYSOUT=C                                            03
//MSGOUT   DD SYSOUT=C                                            04
//STEPLIB  DD DSN=A123456.LOAD,DISP=SHR                           05
//SORTOUT  DD DSN=&&OUTPUT,DISP=(,PASS),UNIT=SYSDA,               06
//  SPACE=(CYL,(8,4))                                             07
//SORTCNTL DD *                                                   08
* Update file size estimate                                      09
  OPTION FILSZ=E30000                                            10
-----------------------------------------------------------------------

-----------------------------------------------------------------------
MYSORT CSECT                                                      11
       .
       .
       .
       LA   R1,PL1            SET ADDRESS OF PARAMETER LIST     12
*                              TO BE PASSED TO DFSORT            13
       ST   R2,PL4            SET ADDRESS OF GETMAINED AREA     14
*                              TO BE PASSED TO E15               15
       LINK EP=SORT           INVOKE DFSORT                     16
       .
       .
       .
PL1    DC   A(CTLST)          ADDRESS OF CONTROL STATEMENTS     17
PL2    DC   A(E15)            ADDRESS OF E15 ROUTINE            18
PL3    DC   A(0)              NO E35 ROUTINE                    19
PL4    DS   A                 USER EXIT ADDRESS CONSTANT        20
PL5    DC   F'-1'             INDICATE END OF LIST              21
CTLST  DS   0H                CONTROL STATEMENTS AREA           22
       DC   AL2(CTL2-CTL1)    LENGTH OF CHARACTER STRING        23
CTL1   DC   C' SORT FIELDS=(5,8,CSF,A)'                         24
       DC   C' RECORD TYPE=F,LENGTH=80 '                        25
       DC   C' OPTION FILSZ=E25000,DYNALLOC,'                   26
       DC   C'RESINV=8000 '                                     27
       DC   C' OMIT FORMAT=CSF,COND=(5,8,EQ,13,8) '             28
CTL2   EQU  *                                                   29
OUT    DCB  DDNAME=MSGOUT,...                                   30
E15    DS   0H                E15 ROUTINE                       31
       .
       .
       .
       L    R2,4(,R1)         GET ADDRESS OF GETMAINED AREA     32
       .
       .
       .
       BR   R14               RETURN TO DFSORT                  33
       .
       .
       .
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls a program named MYSORT that in turn calls DFSORT.

03
SYSOUT DD statement. Directs DFSORT messages and control statements to SYSOUT class C.

04
MSGOUT DD statement. Directs MYSORT messages to SYSOUT class C.

05
STEPLIB DD statement. Specifies the load library that contains MYSORT.

06-07
SORTOUT DD statement. The output data set is temporary and is to be allocated on SYSDA. Because SORTIN is not used, DFSORT sets the RECFM and LRECL from the RECORD statement and sets the BLKSIZE to the LRECL (unblocked records).

08
SORTCNTL DD statement. DFSORT control statements follow. Statements in SORTCNTL override or supplement statements passed by MYSORT in the extended parameter list it uses.

09
Comment statement. Printed but otherwise ignored.

10
OPTION statement. FILSZ=E30000 specifies an estimate of 30000 records, overriding FILSZ=E25000 in the OPTION statement of the extended parameter list. Because the E15 routine supplies all of the input records, DFSORT will not be able to determine the file size accurately; therefore, specifying FILSZ can make a significant difference in work space optimization when an E15 routine supplies all of the input records. It's important to change the FILSZ value whenever the number of input records increases significantly.

11
This is the start of program MYSORT. Assume that it GETMAINs a work area, saves its address in register 2, and initializes it with information to be used by the E15 routine.

12-13
MYSORT places the address of the extended parameter list to be passed to DFSORT in register 1.

14-15
MYSORT places the address of the GETMAINed work area in the user exit address constant field in the extended parameter list. DFSORT will pass this address to the E15 routine (in the second word of the E15 parameter list) when it is entered.

16
MYSORT calls DFSORT by it's alias SORT.

17-21
The extended parameter list specifies: the address of the control statements area, the address of the E15 routine, that no E35 routine is present, and the address of the GETMAINed work area. F'-1' indicates the end of the extended parameter list. Subsequent parameter list fields, such as the address of an ALTSEQ table, are not used in this application.

Because the address of the E15 routine is passed in the parameter list, SORTIN cannot be used; if a SORTIN DD statement were present, it would be ignored.

22-23
This is the start of the control statements area. The total length of the control statements is specified.

24
SORT statement. FIELDS specifies an ascending 8-byte floating-sign control field starting at position 5.

25
RECORD statement. TYPE=F and LENGTH=80 specify that the E15 inserts fixed-length records of 80 bytes. In this case, TYPE=F could be omitted, because DFSORT would automatically set a record type of F. However, LENGTH must be specified when an E15 supplies all of the input records.

26-27
OPTION statement. FILSZ=E25000 specifies an estimate of 25000 records, which is overridden by FILSZ=E30000 in SORTCNTL's OPTION statement. DYNALLOC specifies that work data sets are to be dynamically allocated using the installation defaults for the type of device and number of devices. RESINV=8000 specifies that approximately 8000 bytes are required for the system services (for example, GETMAIN and OPEN) that MYSORT's E15 exit routine performs.

28
OMIT statement. FORMAT specifies that the compare fields are floating-sign. COND specifies that input records with equal 8-byte floating-sign compare fields starting in position 5 (also the control field) and position 13 are to be omitted from the output data set.

29
This is the end of the control statements area.

30
This is the DCB for MYSORT's MSGOUT output.

31-33
This is MYSORT's E15 routine. The E15 routine loads the address of the GETMAINed work area from the second word of the E15 parameter list. The E15 routine must supply each input record by placing its address in register 1 and placing a 12 (insert) in register 15. When all the records have been passed, the E15 routine must place an 8 ("do not return") in register 15.

Example 10. Sort with OUTFIL

INPUT
Fixed-length record data set

OUTPUT
Multiple fixed-length record data sets

WORK DATA SETS
Dynamically allocated (by default)

USER EXITS
None

FUNCTIONS/OPTIONS
OUTFIL

```
//EXAMP    JOB A400,PROGRAMMER                                    01
//OUTFIL   EXEC  PGM=SORT                                         02
//SYSOUT   DD  SYSOUT=A                                           03
//SORTIN   DD DSN=GRP.RECORDS,DISP=SHR                            04
//ALLGPS   DD DSN=GRP.ALLGRPS,DISP=OLD                            05
//ALLBU    DD DSN=GRP.BU,DISP=(NEW,CATLG,DELETE),                 06
//            UNIT=3390,SPACE=(TRK,(10,10))                       07
//G1STATS  DD SYSOUT=A                                            08
//G2STATS  DD SYSOUT=A                                            09
//SYSIN    DD    *                                                10
 SORT FIELDS=(6,5,CH,A)                                          11

 OUTFIL FNAMES=(ALLGPS,ALLBU)                                    12

 OUTFIL FNAMES=G1STATS,                                          13
   INCLUDE=(1,3,CH,EQ,C'G01'),                                   14
   HEADER2=(1:'GROUP 1 STATUS REPORT FOR ',&DATE,                15
            ' - PAGE ',&PAGE,2/,                                 16
           6:'ITEM ',16:'STATUS      ',31:'PARTS',/,             17
           6:'-----',16:'------------',31:'-----'),              18
    OUTREC=(6:6,5,                                               19
           16:14,1,CHANGE=(12,                                   20
                           C'1',C'SHIP',                         21
                           C'2',C'HOLD',                         22
                           C'3',C'TRANSFER'),                    23
                   NOMATCH=(C'*CHECK CODE*'),                    24
           31:39,1,BI,M10,LENGTH=5,                              25
           120:X)                                                26

 OUTFIL FNAMES=G2STATS,                                          27
   INCLUDE=(1,3,CH,EQ,C'G02'),                                   28
   HEADER2=(1:'GROUP 2 STATUS REPORT FOR ',&DATE,                29
            ' - PAGE ',&PAGE,2/,                                 30
           6:'ITEM ',16:'STATUS      ',31:'PARTS',/,             31
           6:'-----',16:'------------',31:'-----'),              32
    OUTREC=(6:6,5,                                               33
           16:14,1,CHANGE=(12,                                   34
                           C'1',C'SHIP',                         35
                           C'2',C'HOLD',                         36
                           C'3',C'TRANSFER'),                    37
```

```
                NOMATCH=(C'*CHECK CODE*'),                  38
            31:39,1,BI,M10,LENGTH=5,                        39
            120:X)                                          40
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls DFSORT directly by its alias name SORT.

03
SYSOUT DD statement. Directs DFSORT messages and control statements to sysout class A.

04
SORTIN DD statement. The input data set is named GRP.RECORDS and is cataloged. DFSORT determines from the data set label that the RECFM is FB, the LRECL is 80 and the BLKSIZE is 23440.

05
ALLGPS DD statement. The first OUTFIL output data set is named GRP.ALLGRPS and is catalogued. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

06-07
ALLBU DD statement. The second OUTFIL output data set is named GRP.BU and is to be allocated on a 3390 and catalogued. DFSORT sets the RECFM and LRECL from SORTIN and selects an appropriate BLKSIZE.

08
G1STATS DD statement. The third OUTFIL output data set is directed to sysout class A. Because this is an OUTFIL report data set, DFSORT sets the RECFM to FBA (FB from SORTIN and A for ANSI control characters) and the LRECL to 121 (1 byte for the ANSI control character and 120 bytes for the data). DFSORT sets an appropriate BLKSIZE.

09
G2STATS DD statement. The fourth OUTFIL output data set is directed to sysout class A. Because this is an OUTFIL report data set, DFSORT sets the RECFM to FBA (FB from SORTIN and A for ANSI control characters) and the LRECL to 121 (1 byte for the ANSI control character and 120 bytes for the data). DFSORT sets an appropriate BLKSIZE.

10
SYSIN DD statement. DFSORT control statements follow.

11
SORT statement. FIELDS specifies an ascending 5-byte character control field starting at position 6.

12
OUTFIL statement. The sorted input records are written to the ALLGPS and ALLBU data sets.

13-26
OUTFIL statement. The subset of sorted input records containing 'G01' in positions 1 through 3 are used to produce a report, which is written to the G1STATS data set.

27-40
OUTFIL statement. The subset of sorted input records containing 'G02' in positions 1 through 3 are used to produce a report, which is written to the G2STATS data set.

Example 11. Sort with Pipes and OUTFIL SPLIT

INPUT
Pipes

OUTPUT
Pipes

WORK DATA SETS
Dynamically allocated

USER EXITS
None

FUNCTIONS/OPTIONS
FILSZ, OUTFIL, DYNALLOC

```
//EXAMP   JOB A400,PROGRAMMER                                    01
//RUNSORT EXEC PGM=ICEMAN                                        02
//SYSOUT  DD SYSOUT=H                                            03
//SORTIN  DD DSN=INPUT.PIPE,SUBSYS=PIPE,                         04
//           DCB=(LRECL=60,RECFM=FB,BLKSIZE=32760)               05
//OUT1    DD DSN=OUTPUT.PIPE1,SUBSYS=PIPE,                       06
//           DCB=(LRECL=60,RECFM=FB,BLKSIZE=32760)               07
//OUT2    DD DSN=OUTPUT.PIPE2,SUBSYS=PIPE,                       08
//           DCB=(LRECL=60,RECFM=FB,BLKSIZE=32760)               09
//SYSIN   DD *                                                   10
  OPTION DYNALLOC,FILSZ=U1000000                                 11
  SORT FIELDS=(1,20,CH,A,25,4,BI,A)                              12
  OUTFIL FNAMES=(OUT1,OUT2),SPLIT                                13
```

Line
Explanation

01
Job statement. Introduces this job to the operating system.

02
EXEC statement. Calls DFSORT directly.

03
SYSOUT DD statement. Directs DFSORT messages and control statements to system output class H.

04-05
SORTIN DD statement. The SUBSYS=PIPE parameter directs the allocation to the 'PIPE' subsystem for the pipe named INPUT.PIPE. The DCB statement describes the data set characteristics to subsystem PIPE.

06-07
OUT1 DD statement. The SUBSYS=PIPE parameter directs the allocation to the 'PIPE' subsystem for the pipe named OUTPUT.PIPE1. The DCB statement describes the data set characteristics to subsystem PIPE.

08-09
OUT2 DD statement. The SUBSYS=PIPE parameter directs the allocation to the 'PIPE' subsystem for the pipe named OUTPUT.PIPE2. The DCB statement describes the data set characteristics to subsystem PIPE.

10
SYSIN DD statement. DFSORT control statements follow.

11
OPTION statement. DYNALLOC specifies that work data sets are to be dynamically allocated using the installation defaults for type of device and number of devices. FILSZ=U1000000 specifies an estimate of one million input records.

12
SORT statement. FIELDS specifies an ascending 20-byte character control field starting at position 1 and an ascending 4 byte binary control field starting at position 25.

13
OUTFIL statement. The records from the SORTIN pipe are sorted and written alternatively to the OUT1 and OUT2 pipes (that is, the sorted records are split evenly between the two output pipes).

Example 12. Sort with INCLUDE and LOCALE

INPUT
Fixed-length record data set

OUTPUT
Fixed-length record data set

WORK DATA SETS
Dynamically allocated (by default)

USER EXITS
None

FUNCTIONS/OPTIONS
INCLUDE, LOCALE

```
//EXAMP    JOB  A400,PROGRAMMER                                  01
//STEP1    EXEC PGM=SORT,PARM='LOCALE=FR_CA'                     02
//STEPLIB  DD  DSN=SYS1.SCEERUN,DISP=SHR                         03
//SYSOUT   DD  SYSOUT=A                                          04
//SORTIN   DD  DSN=INPUT.FRENCH.CANADA,DISP=SHR                  05
//SORTOUT  DD  DSN=OUTPUT.FRENCH.CANADA,DISP=OLD                 06
//SYSIN    DD  *                                                 07
 SORT FIELDS=(1,20,CH,A,25,1,BI,D,30,10,CH,A)                   08
 INCLUDE COND=(40,6,CH,EQ,50,6,CH)                             09
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls DFSORT directly by its alias name SORT. LOCALE specified in EXEC PARM overrides the installation default for LOCALE. The locale for the French language and the cultural conventions of Canada will be active.

03
STEPLIB DD statement. Specifies the Language Environment run-time library containing the dynamically loadable locales.

04
SYSOUT statement. Directs DFSORT messages and control statements to sysout class A.

05
SORTIN DD statement. The input data set is named INPUT.FRENCH.CANADA and is cataloged. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

06
SORTOUT DD statement. The output data set is named OUTPUT.FRENCH.CANADA and is cataloged. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

07
SYSIN DD statement. DFSORT control statements follow.

08
SORT statement. FIELDS specifies an ascending 20-byte character control field starting at position 1, a one-byte descending binary control field starting at position 25, and a 10-byte ascending character control field starting at position 30. The character (CH) control fields will be sorted according to the collating rules defined in locale FR_CA.

09
INCLUDE statement. COND specifies that only input records with equal 6-byte character compare fields starting in position 40 and position 50 are to be included in the output data set. The character (CH) compare fields will be compared according to the collating rules defined in locale FR_CA.

Example 13: Sort with z/OS UNIX files

INPUT
Concatenated z/OS UNIX Files

OUTPUT
z/OS UNIX File

WORK DATA SETS
Dynamically allocated (by default)

USER EXITS
None

FUNCTIONS/OPTIONS
None

```
//EXAMP   JOB A400,PROGRAMMER                                        01
//S1      EXEC PGM=SORT                                              02
//SYSOUT  DD SYSOUT=A                                                03
//SORTIN  DD  PATH='/user/hfs.inp1.txt',PATHOPTS=ORDONLY,            04
//        LRECL=80,BLKSIZE=240,RECFM=FB,FILEDATA=TEXT                05
//        DD  PATH='/user/hfs.inp2.txt',PATHOPTS=ORDONLY,            06
//        LRECL=80,BLKSIZE=80,RECFM=F,FILEDATA=TEXT                  07
//SORTOUT DD  PATH='/user/hfs.ut.txt',PATHOPTS=OWRONLY,              08
//        LRECL=80,BLKSIZE=80,RECFM=F,FILEDATA=TEXT                  09
//SYSIN DD *                                                         10
  SORT FIELDS=(10,8,CH,A)                                          11
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls DFSORT directly by its alias name SORT.

03
SYSOUT DD statement. Directs DFSORT messages and control statements to system output class A.

04-05
SORTIN DD statement. The first input file is a z/OS UNIX file named /user/hfs.inp1.txt. Only read access is allowed. The file is defined as a text file. It has fixed-length records with a record size of 80 and a block size of 240.

06-07
The second input file is a z/OS UNIX file named /user/hfs.inp2.txt. Only read access is allowed. The file is defined as a text file. It has fixed-length records with a record size of 80 and a block size of 80.

08-09
SORTOUT DD statement. The output file is a z/OS UNIX file named /user/hfs.ut.txt. Only write access is allowed. The file is defined as a text file. It has fixed-length records with a record size of 80 and a block size of 80.

10
SYSIN DD statement. DFSORT control statements follow.

11
SORT statement. FIELDS specifies an ascending 8-byte character control field starting at position 10.

Example 14. Sort with IFTHEN

```
INPUT                  Three fixed-length record data sets
OUTPUT                 Fixed-length record data set
WORK DATA SETS         Dynamically allocated (by default)
USER EXITS             None
FUNCTIONS/OPTIONS      IFTHEN

//EXAMP JOB A400,PROGRAMMER                                        01
//S1    EXEC PGM=SORT                                              02
//SYSOUT DD SYSOUT=A                                               03
//SORTIN DD DSN=INPUT.FILE1,DISP=SHR                               04
//       DD DSN=INPUT.FILE2,DISP=SHR                               05
//       DD DSN=INPUT.FILE3,DISP=SHR                               06
//SORTOUT DD DSN=OUTPUT.FILE,DISP=(NEW,CATLG,DELETE),              07
//           SPACE=(CYL,(5,5)),UNIT=SYSDA                          08
//SYSIN DD *                                                       09
  INREC IFTHEN=(WHEN=(1,3,CH,EQ,C'HDR'),                          10
          OVERLAY=(6:YDDD=(D4/),81:C'0',82:SEQNUM,2,ZD)),          11
        IFTHEN=(WHEN=(1,3,CH,EQ,C'TRL'),                          12
          OVERLAY=(11:YDDD=(D4/),81:C'9',82:SEQNUM,2,ZD)),         13
        IFTHEN=(WHEN=NONE,                                        14
          OVERLAY=(81:C'1'))                                      15
```

```
  SORT FIELDS=(81,1,CH,A,8,5,CH,A)                                  16
  OUTFIL REMOVECC,                                                  17
    OMIT=(81,1,SS,EQ,C'0,9',AND,82,2,ZD,GT,+1),                     18
    OUTREC=(1,80)                                                   19
```

This example shows how you can use three input files, each with a header record ('HDR'), detail records ('DTL') and a trailer record ('TRL'), and create an output file with one header record with the current date, the sorted detail records, and one trailer record with the current date.

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls DFSORT directly by its alias name SORT.

03
SYSOUT DD statement. Directs DFSORT messages and control statements to sysout class A.

04-06
SORTIN DD statement. Consists of a concatenation of three input data sets: INPUT.FILE1, INPUT.FILE2 and INPUT.FILE3. DFSORT determines from the data set labels that each data set has RECFM=FB and LRECL=80. The BLKSIZEs vary. Each input data set has a header record, detail records, and a trailer record.

07-08
SORTOUT DD statement. Creates a new output data set: OUTPUT.FILE. DFSORT sets RECFM=FB, LRECL=80 and selects an appropriate BLKSIZE. The output data set will have one header record, the sorted detail records, and one trailer record.

09
SYSIN DD statement. DFSORT control statements follow.

10-15
INREC statement. The first IFTHEN WHEN=(logexp) clause identifies and operates on header records ('HDR' in positions 1-3); OVERLAY puts today's date in the form 'ddd/yyyy' in positions 6-13, adds a '0' in position 81, adds a ZD sequence number in positions 82-83 and does not affect the rest of the record.

The second IFTHEN WHEN=(logexp) clause identifies and operates on trailer records ('TRL' in positions 1-3); OVERLAY puts today's date in the form 'ddd/yyyy' in positions 11-18, adds a '9' in position 81, adds a ZD sequence number in positions 82-83 and does not affect the rest of the record.

The IFTHEN WHEN=NONE clause identifies and operates on detail records (not 'HDR' or 'TRL' in positions 1-3); OVERLAY adds a '1' in position 81 and does not affect the rest of the record.

DFSORT extends the reformatted input records from 80 bytes to 83 bytes to accommodate the identifier byte added in position 81 and the sequence number added in positions 82-83.

The '0', '1' or '9' identifier byte added in position 81 allows us to sort the header records ('0') first, followed by the detail records ('1'), and then the trailer records ('9'). The sequence number added in positions 82-83 will allow us to keep only the first header record and the first trailer record. The sequence number will be 1 for the first header record, 2 for the second header record and 3 for the third header record. Likewise, the sequence number will be 1 for the first trailer record, 2 for the second trailer record and 3 for the third trailer record. Since the sequence number is not specified for the detail records, it will be blank.

16
SORT statement. FIELDS specifies an ascending 1-byte character control field at position 81 (the identifier byte added by INREC), and an ascending 5-byte character control field starting at position 8 (the key for the detail records).

17-19
OUTFIL statement. REMOVECC removes the ANSI carriage control characters and ensures that the RECFM is FB rather than FBA. OMIT specifies that reformatted output records with '0' or '9' in position 81 (header or trailer records) and a sequence number in positions 82-83 greater than 1 (second and subsequent header or trailer records), are omitted. OUTREC keeps only positions 1-80 for the OUTFIL

output records, thus removing the identifier byte and sequence number we added in positions 81-83 with the INREC statement (we do not want these temporary fields in the OUTFIL output records).

Example 15. Sort with 64-bit parameter lists, E15, E35 and OUTFIL

INPUT
Variable-length records from E15

OUTPUT
Variable-length output data set

WORK DATA SETS
Dynamically allocated (by default)

USER EXITS
E15 and E35

FUNCTIONS/OPTIONS
64-bit parameter lists, OUTFIL

The JCL for running program PGM1, and the code used by PGM1 to invoke DFSORT with the 64-bit invocation parameter list and use the 64-bit E15 and E35 parameter lists, are shown below.

```
//EXAMP    JOB A400,PROGRAMMER                                  01
//STEP1    EXEC PGM=PGM1                                        02
//SYSOUT   DD SYSOUT=A                                          03
//SORTOUT  DD DSN=PL64.OUTPUT,DISP=(NEW,CATLG,DELETE),          04
//  SPACE=(CYL,(8,4)),UNIT=SYSDA                                05
------------------------------------------------------------------------

------------------------------------------------------------------------
PGM1     CSECT                                                 06
PGM1     AMODE 64
PGM1     RMODE 31
         STM   14,12,12(13)        SAVE LOW PART REGS
         BASR  15,0                SET THE TEMPORARY
         USING *,15                 BASE REG
M0       DS    0H
         STMH  0,15,HIGHREGS       SAVE HIGH PART REGS
         LLGTR 12,15               SET THE PERMANENT
         DROP  15                   BASE
         USING M0,12                 REG
         LMH   0,15,HIGHZERO       SET HIGH PART OF REGS TO ZERO
         ST    13,SAVE+4           SAVE SAVE AREA ADDR
         LR    11,13               RELOAD SAVE AREA ADDR
         LA    13,SAVE             LOAD NEW SAVE AREA ADDR
         ST    13,8(11)            SAVE NEW SA ADDR INTO OLD SA
* Obtain 64-bit storage (memory object):
* - for 64-bit Invocation Parameter List
* - for 64-bit Control Statements Area
* - for work area passed to E15, E35 exits via USER ADDRESS CONSTANT
         STG   12,MOTOKEN          SAVE USERTOKEN
         LA    2,2                 SET NUMBER OF SEGMENTS
         STG   2,MOSEGM             FOR MEMORY OBJECT (MO) - 2MB
         IARV64 REQUEST=GETSTOR,   GET MEMORY OBJECT                  +
               ORIGIN=MOADDR,      ADDRESS OF MEMORY OBJECT           +
               USERTKN=MOTOKEN,    USERTOKEN                          +
               SEGMENTS=MOSEGM,    SIZE OF MEMORY OBJECT              +
               COND=YES,           CONDITIONAL REQUEST                +
               MF=(E,MOWORK)
         LTR   15,15               IF MEMORY OBJECT NOT OBTAINED,
         BNZ   NOMO                 EXIT WITH ERROR RETURN CODE
         LG    10,MOADDR           GET 64-BIT ADDR IN MO FOR
*                                   64-BIT INVOCATION PARMLIST
         USING ICE64INV,10         MAKE ADDRESSABLE
         XC    0(ICE64LNG,10),0(10) CLEAR 64-BIT INVOCATION PARM LIST
         MVC   ICEPLID(8),PL64SORT MOVE 64-BIT PARM LIST IDENTIFIER
         LA    2,1024(,10)         GET 64-BIT ADDR IN MEMORY OBJECT
*                                   FOR 64-BIT CONTROL STATEMENTS AREA
         STG   2,ICECTL             AND STORE IN 64-BIT PARM LIST
         MVC   0(CTLNG,2),CTLST    MOVE CONTROL STATEMENTS INTO MO
*
         LA    2,1024(,2)          GET 64-BIT ADDR IN MO FOR WORK AREA
         STG   2,MOWA               USED IN E15, E35 EXITS
         MVC   ICEUC(8),UADCON     MOVE USER EXIT ADDRESS CONSTANT
*
         LLGF  2,=A(E15)           STORE E15 ADDR
```

```
         STG   2,ICE15E32           INTO PARM LIST
         OI    ICEMDEX1,ICE15A64     WITH AMODE 64 AND
         OI    ICEMDEX2,ICE15PLT      WITH 64-BIT EXEC PARAMETER LIST
*
         LLGF  2,=A(E35)           STORE E35 ADDR
         STG   2,ICE35              INTO PARM LIST
         OI    ICEMDEX1,ICE35A64     WITH AMODE 64 AND
         OI    ICEMDEX2,ICE35PLT      WITH 64-BIT EXEC PARAMETER LIST
*                                   INTO 64-BIT INVOCATION PARM LIST
         LGR   1,10                LOAD ADDR OF 64-BIT INVOCATION
*                                   PARAMETER LIST
         LINK  EP=SORT64           INVOKE SORT WITH 64-BIT NAME
         LR    11,15               SAVE RETURN CODE
FREEMO   DS    0H
         IARV64 REQUEST=DETACH,      FREE MEMORY OBJECT               +
               MATCH=USERTOKEN,                                       +
               USERTKN=MOTOKEN,                                       +
               COND=YES,                                              +
               MF=(E,MOWORK)
         LR    15,11              RESTORE RETURN CODE
*
EXITALL  DS    0H
         LMH   0,14,HIGHREGS      RESTORE HIGH PART REGS
         L     13,SAVE+4          RESTORE OLD SA ADDR
         L     14,12(13)          RESTORE CALLER'S
         LM    1,12,24(13)         REGS
         BSM   0,14               RETURN TO CALLER
*
NOMO     LA    15,20              ERROR RETURN CODE (NOT MO)
         B     EXITALL            RETURN TO CALLER
*
SAVE     DC    18F'0'              SAVE AREA
HIGHREGS DC    16F'0'
HIGHZERO DC    16F'0'
*
CTLST    DS    0H                  ADDR OF CONTROL STMTS
         DC    AL2(CTL2-CTL1)      LENGTH OF CONTROL STATEMENT STRING
CTL1     EQU   *
         DC    C' SORT FIELDS=(5,4,BI,A)'
         DC    C' RECORD LENGTH=(80,80,80),TYPE=V '
         DC    C' OPTION FILSZ=E1000 '
         DC    C' OUTFIL FNAMES=SORTOUT '
CTL2     EQU   *
CTLNG    EQU   *-CTLST             LENGTH OF CONTROL STATEMENTS AREA
PL64SORT DC    C'PL64SORT'         IDENTIFIER OF 64-BIT PARM LIST
UADCON   DC    F'0',A(MOWA)        USER ADDRESS CONSTANT
MOWA     DC    D'0'                ADDR OF WORK AREA IN MO
*
MOTOKEN  DC    D'0'                USERTOKEN
MOSEGM   DC    D'0'                SIZE OF MO (IN MB)
MOADDR   DC    D'0'                ADDR OF MO
*
MOWORK   DS    0D                  IARV64 MACRO WORK AREA
         IARV64 MF=(L,MOV64L)
*
         LTORG
*
         ICEPL64                   MAPPING OF 64-BIT PARM LISTS
*
         DROP  12
PGM1     CSECT
************** E15 ********************************************
*
E15      DS    0H
* THIS E15 EXIT ROUTINE FORMS ALL RECORDS IN MEMORY OBJECT
* AND INSERTS THEM TO DFSORT AS 64-BIT ADRESSED RECORDS.
*
         STMG  14,12,8(13)        SAVE CALLER's REGISTERS
         BASR  11,0               SET
         USING *,11                ADDRESSABILITY
         LLGTR 11,11              SET CLEAN BASE REG
         LARL  14,SAVEE15         GET NEW SAVE AREA ADDRESS
         STG   13,128(,14)        CHAIN TO PREVIOUS SAVE AREA
         STG   14,136(,13)        CHAIN TO NEW SAVE AREA
         LGR   13,14              SET R13 TO SAVE AREA ADDRESS
         LMH   0,15,E15HZERO      SET HIGH PART OF REGS TO ZERO
         LR    10,1               SAVE EXIT PARAMETER LIST ADDRESS
         USING ICE64E15,10        SET ADDRESSABILITY
         ICM   15,15,COUNT        EXIT IF ALL RECORDS WERE
         BZ    EOF                 INSERTED
         LLGF  1,ICE15UC+4        GET USER CONSTANT ADDR
         LG    1,0(1)             GET ADDR OF THE WORK AREA IN MO
```

```
         MVC   0(80,1),RECE15     MOVE RECORD INTO MEMORY OBJECT
         BCTR  15,0               SET NEW
         ST    15,COUNT            SORT FIELD
*
         LA    15,12              RC=12 (INSERT RECORD)
* RETURN TO DFSORT
BACKE15  DS    0H
         LG    13,128(,13)        RESTORE CALLER's R13
         LG    14,8(,13)          RESTORE CALLER's R14
         LMG   2,12,40(13)        RESTORE CALLER's R2-R12
         BSM   0,14               RETURN
*
EOF      DS    0H
* Create final record in MO for its insert to DFSORT as 64-bit
* addressed record in the E35 exit
         LLGF  15,ICE15UC+4       GET USER CONSTANT ADDR
         LG    15,0(15)           GET ADDR IN MO FOR FINAL RECORD
         XC    0(80,15),0(15)     CLEAR RECORD's DATA
         MVI   1(15),8            SET NEW LENGTH OF FINAL RECORD
         MVC   4(4,15),RECNUM     SET 'NUMBER OF INSERTED RECORDS'
         LA    15,8               SET RETURN CODE
         B     BACKE15            RETURN CODE
*
E15HZERO DC    18F'0'
SAVEE15  DC    0D'0',F'0',C'F4SA',17FD'0'
HIGHE15  DC    16F'0'
FLAGE15  DC    X'00'
E15MOADR DC    D'0'               ADDR OF MO
E15SEGM  DC    D'0'               SIZE OF MO (IN MB)
E15TOKEN DC    D'0'               USERTOKEN
RECNUM   DC    F'1000'            NUMBER OF RECORDS
RECE15   DC    H'80'              LENGTH OF RECORD
         DC    H'0'
COUNT    DC    F'1000'            SORT FIELD
         DC    6C'ABCDEF'         DATA
         DC    6C'123456'          FIELDS
*
         LTORG
*
         DROP
PGM1     CSECT
************** E35 ******************************************
E35      DS    0H
*
* THIS E35 EXIT ACCEPTS ALL RECORDS FROM DFSORT
* AND AT EOF INSERTS ONE RECORD TO DFSORT AS 64-BIT RECORD.
* THE ADDRESS OF THE FINAL RECORD IN MEMORY OBJECT IS PASSED
* VIA USER ADDRESS CONSTANT FIELD OF 64-BIT PARAMETER LIST.
*
         STMG  14,12,8(13)        SAVE CALLER's REGISTERS
         BASR  11,0               SET
         USING *,11                ADDRESSABILITY
         LLGTR 11,11              SET CLEAN BASE REG
         LARL  14,SAVEE35         GET NEW SAVE AREA ADDRESS
         STG   13,128(,14)        CHAIN TO PREVIOUS SAVE AREA
         STG   14,136(,13)        CHAIN TO NEW SAVE AREA
         LGR   13,14              SET R13 TO SAVE AREA ADDRESS
         LMH   0,15,E35HZERO      SET HIGH PART REGS TO ZERO
         LR    10,1               SAVE EXIT PARM LIST ADDRESS
         USING ICE64E35,10        SET ADDRESSABILITY
         ICM   1,15,ICE35RL+4     GET ADDR OF NEW RECORD
* IF ADDR OF NEW RECORD IT IS ZERO, EOF IS INDICATED
         BZ    RC8E35             BR IF EOF
*
         LA    15,0               'ACCEPT' RETURN CODE
* RETURN TO DFSORT
BACKE35  DS    0H
         LG    13,128(,13)        RESTORE CALLER's R13
         LG    14,8(,13)          RESTORE CALLER's R14
         LMG   2,12,40(13)        RESTORE CALLER's R2-R12
         BSM   0,14               RETURN
*
RC8E35   DS    0H
         TM    FLAGE35,X'01'      IF FINAL FLAG IS ON
         BO    RC8E35A             EXIT WITH 8 RETURN CODE
         MVI   FLAGE35,X'01'      SET FINAL FLAG
         LLGF  1,ICE35UC+4         GET USER CONSTANT ADDR
         LG    1,0(1)             GET MO ADDR WITH FINAL RECORD
         LA    15,12              SET 'INSERT' RETURN CODE
         B     BACKE35            RETURN TO DFSORT
RC8E35A  DS    0H
         LA    15,8               SET RETURN CODE
```

```
         B     BACKE35            RETURN TO DFSORT
*
E35HZERO DC    18F'0'
HIGHE35  DC    18F'0'
SAVEE35  DC    0D'0',F'0',C'F4SA',17FD'0'
FLAGE35  DC    X'00'
*
         LTORG
*
         END
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls a program named PGM1 that in turn calls DFSORT.

03
SYSOUT DD statement. Directs DFSORT messages and control statements to SYSOUT class A.

04-05
SORTOUT DD statement. The OUTFIL data set.

06
Start of the complete source code for PGM1. PGM1 uses the ICEPL64 mapping macro in the DFSORT target library, SICEUSER, to provide separate Assembler DSECTs for the 64-bit invocation and exit parameter lists.

PGM1 LINKs to DFSORT using the name SORT64 to indicate that it is using the 64-bit invocation parameter list. PGM1 passes SORT, RECORD, OPTION and OUTFIL statements in the control statement area.

PGM1's E15 user exit passes 64-bit addressed records to DFSORT.

PGM1's E35 user exit accepts all of the sorted records and inserts one additional record at EOF. DFSORT writes all of the records to the OUTFIL data set.

MERGE examples

Although the MERGE operators in the examples in this section could all be contained in a single ICETOOL job step, they are shown and discussed separately for clarity.

Example 1. Merge with EQUALS

INPUT
Blocked fixed-length records on disk

OUTPUT
Blocked fixed-length records on 3390

WORK DATA SETS
Not applicable

USER EXITS
None

FUNCTIONS/OPTIONS
EQUALS

```
//EXAMP    JOB A400,PROGRAMMER                                  01
//STEP1    EXEC PGM=SORT                                        02
//SYSOUT   DD   SYSOUT=A                                        03
//SORTIN01 DD   DSN=M1234.INPUT1,DISP=SHR                       04
//SORTIN02 DD   DSN=M1234.INPUT2,DISP=SHR                       05
//SORTIN03 DD   DSN=M1234.INPUT3,DISP=SHR                       06
//SORTOUT  DD   DSN=M1234.MERGOUT,DISP=(,KEEP),                 07
//  SPACE=(CYL,(2,4)),UNIT=3390                                 08
//SYSIN    DD *                                                 09
```

```
    MERGE FIELDS=(1,8,CH,A,20,4,PD,A)                          10
    OPTION EQUALS                                              11
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls DFSORT directly by its alias SORT.

03
SYSOUT DD statement. Directs DFSORT messages and control statements to sysout class A.

04
SORTIN01 DD statement. The first input data set is named M1234.INPUT1 and is cataloged. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

05
SORTIN02 DD statement. The second input data set is named M1234.INPUT2 and is cataloged. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

06
SORTIN03 DD statement. The third input data set is named M1234.INPUT3 and is cataloged. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

07-08
SORTOUT DD statement. The output data set is named M1234.MERGOUT and is to be allocated on 3390 and kept. DFSORT sets the RECFM and LRECL from the SORTINnn data sets and selects an appropriate BLKSIZE.

09
SYSIN DD statement. DFSORT control statements follow.

10
MERGE statement. FIELDS specifies an ascending 8-byte character control field starting at position 1 and an ascending 4-byte packed-decimal field starting at position 20. The records in each input data set must already be in the order specified.

25
OPTION statement. EQUALS specifies that the order of output records with equal control fields is to be based on the file number of the input data sets and the original order of the records within each input data set.

Example 2. Merge with LOCALE and OUTFIL

INPUT
Fixed-length record data set

OUTPUT
Multiple fixed-length record data sets

WORK DATA SETS
Not applicable

USER EXITS
None

FUNCTIONS/OPTIONS
LOCALE, OUTFIL

```
//EXAMP    JOB  A400,PROGRAMMER                               01
//STEP1    EXEC PGM=SORT                                      02
//STEPLIB  DD  DSN=SYS1.SCEERUN,DISP=SHR                      03
//SYSOUT   DD  SYSOUT=A                                       04
//SORTIN01 DD  DSN=INPUT01.GERMAN.GERMANY,DISP=SHR            05
//SORTIN02 DD  DSN=INPUT02.GERMAN.GERMANY,DISP=SHR            06
//SORTIN03 DD  DSN=INPUT03.GERMAN.GERMANY,DISP=SHR            07
//GP1      DD  DSN=OUTPUT.GERMAN.GP1,DISP=OLD                 08
```

```
//GP2      DD  DSN=OUTPUT.GERMAN.GP2,DISP=OLD                  09
//GP3      DD  DSN=OUTPUT.GERMAN.SAVE,DISP=OLD                10
//DFSPARM  DD  *                                              11
 LOCALE=De_DE.IBM-1047                                        12
 MERGE FIELDS=(25,5,CH,A,40,4,PD,D)                           13
 OUTFIL FNAMES=GP1,                                           14
   INCLUDE=(23,1,CH,LE,C'Ö')                                  15
 OUTFIL FNAMES=GP2,                                           16
   INCLUDE=(23,1,CH,GT,C'Ö',AND,                              17
            23,1,CH,LT,C'Ü')                                  18
 OUTFIL FNAMES=GP3,SAVE                                       19
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls DFSORT directly by its alias name SORT.

03
STEPLIB DD statement. Specifies the Language Environment run-time library containing the dynamically loadable locales.

04
SYSOUT statement. Directs DFSORT messages and control statements to sysout class A.

05
SORTIN01 DD statement. The first input data set is named INPUT01.GERMAN.GERMANY and is cataloged. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

06
SORTIN02 DD statement. The second input data set is named INPUT02.GERMAN.GERMANY and is cataloged. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

07
SORTIN03 DD statement. The third input data set is named INPUT03.GERMAN.GERMANY and is cataloged. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

08
GP1 DD statement. The first OUTFIL output data set is named OUTPUT.GERMAN.GP1 and is cataloged. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

09
GP2 DD statement. The second OUTFIL output data set is named OUTPUT.GERMAN.GP2 and is cataloged. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

10
GP3 DD statement. The third OUTFIL output data set is named OUTPUT.GERMAN.GP3 and is cataloged. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

11
DFSPARM DD statement. DFSORT control statements follow.

12
LOCALE parameter. Overrides the installation default for LOCALE. The locale for the German language and the cultural conventions of Germany based on the IBM-1047 encoded character set will be active.

13
MERGE statement. FIELDS specifies an ascending 5-byte character control field starting at position 25, and a descending 4-byte packed decimal control field starting at position 40. The character (CH) control field will be merged according to the collating rules defined in locale De_DE.IBM-1047. The records in each input data set must already be in the order specified.

14-15
OUTFIL statement. The subset of records with character values less than or equal to 'Ö' in position 23 are written to the GP1 output data set. The character (CH) compare field and character constant will be compared according to the collating rules defined in locale De_DE.IBM-1047.

16-18
OUTFIL statement. The subset of records with character values greater than 'Ö' but less than 'Ü' in position 23 are written to the GP2 output data set. The character (CH) compare fields and character constants will be compared according to the collating rules defined in locale De_DE.IBM-1047.

19
OUTFIL statement. Any records not written to the GP1 or GP2 output data sets are written to the GP3 output data set.

Copy examples

This section contains 2 copy examples.

Example 1. Copy with EXEC PARMs, SKIPREC, MSGPRT and ABEND

INPUT
Blocked fixed-length records on multivolume 3490

OUTPUT
Blocked fixed-length records on disk

WORK DATA SETS
Not applicable

USER EXITS
None

FUNCTIONS/OPTIONS
SKIPREC, MSGPRT, ABEND

```
//EXAMP    JOB A400,PROGRAMMER                                    01
//STEP1    EXEC   PGM=SORT,                                       02
//  PARM='SKIPREC=500,MSGPRT=CRITICAL,ABEND'                       03
//SYSOUT   DD SYSOUT=A                                            04
//SORTIN   DD DSN=FLY.RECORDS,VOL=SER=(000333,000343),            05
//  UNIT=(3490,2),DISP=OLD,LABEL=(,NL),                           06
//  DCB=(RECFM=FB,LRECL=12000,BLKSIZE=24000)                      07
//SORTOUT  DD DSN=FLY.RECORDS.COPY,DISP=OLD                       08
//SYSIN DD *                                                      09
  SORT FIELDS=COPY                                              10
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02-03
EXEC statement. Calls DFSORT directly by its alias SORT. SKIPREC=500 specifies that the first 500 input records are not to be included in the output data set. MSGPRT=CRITICAL specifies that error messages, but not informational messages, are to be printed. ABEND specifies that DFSORT is to terminate with a user ABEND if it issues an error message.

04
SYSOUT DD statement. Directs DFSORT messages and control statements to sysout class A.

05-07
SORTIN DD statement. The input data set is named FLY.RECORDS and resides on 3490 volumes 000333 and 000343. The UNIT parameter requests two tape drives, one for each volume of the data set. Because the tape is unlabeled, DCB parameters must be supplied to indicate that the RECFM is FB, the LRECL is 12000 and the BLKSIZE is 24000.

08
SORTOUT DD statement. The output data set is named FLY.RECORDS.COPY and is cataloged. DFSORT determines the RECFM, LRECL and BLKSIZE from the data set label.

09
SYSIN DD statement. DFSORT control statements follow.

10
SORT statement. FIELDS=COPY specifies a copy application.

Example 2. Copy with INCLUDE and VLSHRT

INPUT
Blocked spanned records on disk

OUTPUT
Blocked spanned records on SYSDA

WORK DATA SETS
Not applicable

USER EXITS
None

FUNCTIONS/OPTIONS
INCLUDE, VLSHRT

```
//EXAMP    JOB A400,PROGRAMMER                                    01
//COPY     EXEC PGM=SORT                                          02
//SYSOUT   DD SYSOUT=A                                            03
//SORTIN   DD DSN=SMF.DATA,DISP=SHR                               04
//SORTOUT  DD DSN=SMF.VIOL,DISP=(,KEEP),SPACE=(CYL,(2,5)),        05
//  UNIT=SYSDA                                                    06
//SYSIN    DD *                                                   07
  INCLUDE COND=(6,1,FI,EQ,80,AND,19,1,BI,EQ,B'1.......')         08
  OPTION COPY,VLSHRT                                             09
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls DFSORT directly by its alias SORT.

03
SYSOUT DD statement. Directs DFSORT messages and control statements to sysout class A.

04
SORTIN DD statement. The input data set is named SMF.DATA and is cataloged. DFSORT determines from the data set label that the RECFM is VBS, the LRECL is 32760 and the BLKSIZE is 23476.

05-06
SORTOUT DD statement. The output data set is named SMF.VIOL and is to be allocated on SYSDA and kept. DFSORT sets the RECFM and LRECL from SORTIN and selects an appropriate BLKSIZE.

07
SYSIN DD statement. DFSORT control statements follow.

08
INCLUDE statement. COND specifies that only input records with decimal 80 in the 1-byte fixed-point field at position 6 and bit 0 on in the 1-byte binary field at position 19 are to be included in the output data set.

09
OPTION statement. COPY specifies a copy application. VLSHRT specifies that records that are too short to contain all of the INCLUDE compare fields are not to be included in the output data set.

Example 3. Copy with OUTREC, PARSE and BUILD

```
INPUT                Blocked fixed-length records on disk
OUTPUT               Blocked fixed-length records on disk
WORK DATA SETS       Not applicable
USER EXITS           None
FUNCTIONS/OPTIONS    OUTREC, PARSE, BUILD
```

```
//EXAMP JOB A400,PROGRAMMER                                          01
//S1 EXEC PGM=ICEMAN                                                 02
//SYSOUT DD SYSOUT=A                                                 03
//SORTIN DD DSN=FLY.VAR.FIELDS.IN,DISP=SHR                           04
//SORTOUT DD DSN=FLY.FIX.FIELDS.OUT,DISP=OLD                         05
//SYSIN DD *                                                         06
  OPTION COPY                                                        07
  OUTREC PARSE=(%00=(ABSPOS=21,STARTAFT=BLANKS,                      08
                    ENDBEFR=C',',FIXLEN=5),                          09
       %=(ENDBEFR=C','),                                             10
       %01=(ENDBEFR=C',',FIXLEN=10),                                 11
       %02=(FIXLEN=7)),                                              12
  BUILD=(1,15,                                                       13
    18:%00,JFY=(SHIFT=RIGHT,LEAD=C'(',TRAIL=C')',LENGTH=7),          14
    30:%02,SFF,TO=ZD,LENGTH=7,                                       15
    40:%01,60:X)                                                     16
```

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls DFSORT directly.

03
SYSOUT DD statement. Directs DFSORT messages and control statements to sysout class A.

04
SORTIN DD statement. The input data set is named FLY.VAR.FIELDS.IN and is cataloged. DFSORT determines from the data set label that the RECFM is FB, the LRECL is 60 and the BLKSIZE is 27960.

05
SORTOUT DD statement. The output data set is named FLY.FIX.FIELDS.OUT and is cataloged. DFSORT determines from the data set label that the RECFM is FB, the LRECL is 60 and the BLKSIZE is 27960.

06
SYSIN DD statement. DFSORT control statements follow.

07
OPTION statement. COPY specifies a copy application.

08-16
OUTREC statement. PARSE and BUILD reformat the input records containing one fixed position/length field and four variable position/length fields to output records containing four fixed position/length fields.

ICEGENER example

This section contains an ICEGENER example.

INPUT
Same as for IEBGENER job

OUTPUT
Same as for IEBGENER job

WORK DATA SETS
Not applicable

USER EXITS
None

FUNCTIONS/OPTIONS
None

```
//EXAMP    JOB A400,PROGRAMMER                                       01
//GENR     EXEC PGM=ICEGENER                                         02
//SYSPRINT DD SYSOUT=A                                               03
//SYSUT1   DD DSN=CTL.MASTER,DISP=SHR                                04
```

```
//SYSUT2   DD DSN=CTL.BACKUP,DISP=OLD                              05
//SYSIN    DD DUMMY                                                06
```

This example shows how to use the ICEGENER facility for an IEBGENER job if your site has not installed ICEGENER as an automatic replacement for IEBGENER. The ICEGENER facility selects the more efficient DFSORT copy function for this IEBGENER job.

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls the ICEGENER facility. PGM=IEBGENER has been replaced by PGM=ICEGENER.

03-06
No other changes to the IEBGENER job are required.

ICETOOL example

This section contains an example of ICETOOL with various operators.

INPUT
Multiple data sets

OUTPUT
Multiple data sets

WORK DATA SETS
Dynamically allocated (automatic)

USER EXITS
ICETOOL's E35 (automatic)

FUNCTIONS/OPTIONS
OCCUR, COPY, SORT, MODE, VERIFY, STATS, DISPLAY

```
//EXAMP    JOB A400,PROGRAMMER                                     01
//TOOLRUN  EXEC PGM=ICETOOL,REGION=1024K                           02
//TOOLMSG  DD SYSOUT=A                                             03
//DFSMSG   DD SYSOUT=A                                             04
//TOOLIN   DD *                                                    05
* Print report showing departments with less than 5 employees      06
  OCCUR  FROM(IN1) LIST(LT5) LOWER(5) BLANK -                      07
    TITLE('Small Departments') PAGE -                              08
    HEADER('Department') HEADER('Employees') -                     09
    ON(45,3,CH)          ON(VALCNT)                                10
* Sort by last name and first name                                 11
   SORT FROM(IN1) TO(DEPTSD,DEPTSP) USING(ABCD)                    12
* Do following operators even if a previous operator failed,       13
* but stop processing if a subsequent operator fails.             14
 MODE STOP                                                        15
* Verify decimal fields                                           16
   VERIFY FROM(IN2) ON(22,6,PD) ON(30,3,ZD)                        17
* Print statistics for record length and numeric fields           18
   STATS FROM(IN2) ON(VLEN) ON(22,6,PD) ON(30,3,ZD)                19
* Sort and produce total for each unique key                      20
   SORT FROM(IN2) TO(OUT4) USING(CTL1)                             21
* Print report containing:                                        22
*   - key and total for each unique key                           23
*   - lowest and highest of the totals                            24
  DISPLAY FROM(OUT4) LIST(LIST1) -                                 25
    TITLE('Unique key totals report') DATE TIME -                  26
    ON(5,10,CH) ON(22,6,PD) ON(30,3,ZD) -                          27
    MINIMUM('Lowest') MAXIMUM('Highest') PLUS                      28
//LT5      DD SYSOUT=A                                             29
//IN1      DD DSN=FLY.INPUT1,DISP=SHR                              30
//ABCDCNTL DD *                                                    31
* Sort by last name, first name                                    32
  SORT FIELDS=(12,15,CH,A,1,10,CH,A)                               33
```

```
//DEPTSD   DD DSN=FLY.OUTPUT1,DISP=SHR                          34
//DEPTSP   DD SYSOUT=A                                          35
//IN2      DD DSN=FLY.INPUT2,DISP=SHR                           36
//OUT4     DD DSN=FLY.OUTPUT2,DISP=OLD                          37
//CTL1CNTL DD *                                                 38
* Sort and produce totals in one record for each unique key    39
  SORT FIELDS=(5,10,CH,A)                                       40
  SUM FIELDS=(22,6,PD,30,3,ZD)                                  41
//LIST1    DD SYSOUT=A                                          42
```

This example shows how ICETOOL can be used to perform multiple operations in a single step.

Line
Explanation

01
JOB statement. Introduces this job to the operating system.

02
EXEC statement. Calls ICETOOL specifying the recommended REGION of 1024K.

03
TOOLMSG DD statement. Directs ICETOOL messages and statements to system output class A.

04
DFSMSG DD statement. Directs DFSORT messages and control statements to SYSOUT class A.

05
TOOLIN DD statement. ICETOOL statements follow. The MODE for the ICETOOL run is initially set to STOP. If an error is detected for an operator, SCAN mode will be entered.

06
Comment statement. Printed but otherwise ignored.

07-10
OCCUR operator. Prints, in the LT5 data set, a report detailing each value for the specified field in the IN1 data set and the number of times that value occurs.

11
Comment statement.

12
SORT operator. Records from the IN1 data set are sorted to the DEPTSD and DEPTSP data sets using the DFSORT control statements in the ABCDCNTL data set. As a result, FLY.OUTPUT1 and DEPTSP (SYSOUT) contain the sorted records from FLY.INPUT1.

13-14
Comment statements.

15
MODE operator. The MODE is reset to STOP (needed in case SCAN mode was entered due to an error for a previous operator). If an error is detected for a subsequent operator, SCAN mode will be entered. This divides the previous operators and subsequent operators into two unrelated groups.

16
Comment statement.

17
VERIFY operator. Identifies invalid values, if any, in the specified decimal fields of the IN2 data set. Used to stop subsequent operations if any invalid value is found in FLY.INPUT2.

18
Comment statement.

19
STATS operator. Prints the minimum, maximum, average, and total for the specified fields of the IN2 data set.

ON(VLEN) operates on the record length of the records in FLY.INPUT2. Thus, the values printed for ON(VLEN) represent the shortest record, the longest record, the average record length, and the total number of bytes for FLY.INPUT2.

20
Comment statement.

21
SORT operator. Records from the IN2 data set are sorted and summarized to the OUT4 data set using the DFSORT control statements in the CTL1CNTL data set. As a result, FLY.OUTPUT2 contains one record from FLY.INPUT2 for each unique sort field with totals for the sum fields.

22-24
Comment statements.

25-28
DISPLAY operator. Prints, in the LIST1 data set, a report detailing each sort and sum value for the OUT4 data set resulting from the previous operation, and the lowest and highest value for each sum field.

29-42
DD statements. Defines the data sets and DFSORT control statements used for the ICETOOL operations described previously in this section.

Appendix A. Using work space

Introduction

When a sort application cannot be performed entirely in virtual storage, DFSORT must use work space. The amount of work space required depends on:

- The amount of data being sorted
- The amount of virtual storage available to DFSORT
- The amount of Hiperspace available to DFSORT
- The type of devices you use
- The DFSORT functions and features you use (for example, VLSHRT, locale processing, EFS, and ALTSEQ can increase the amount of work space required).

There are three ways to supply work space for a DFSORT application:

- Hiperspace
- Dynamic allocation of work data sets
- JCL allocation of work data sets.

For best performance, an optimal amount of Hiperspace, in combination with dynamically allocated disk work data sets, is strongly recommended. See "Use Hipersorting" on page 759 for more information on using the HIPRMAX option. The DYNAUTO installation option, or the DYNALLOC run-time option, can be used to dynamically allocate work data sets.

Central storage

DFSORT leverages central storage as intermediate work space through the use of either memory objects or Hiperspaces. Using the installation defaults of HIPRMAX=OPTIMAL, MOSIZE=MAX and MOWRK=YES ensures that DFSORT will use memory object or Hiperspace whenever possible. Sites can tune their definition of HIPRMAX=OPTIMAL and MOSIZE=MAX through use of the EXPMAX, EXPOLD, EXPRES and TUNE installation options. See *z/OS DFSORT Installation and Customization* for more information.

DFSORT's use of memory object or Hiperspace storage depends upon the availability of central storage, the needs of other concurrent memory object sorting, Hipersorting, and dataspace sorting applications throughout the time the application runs, and the settings of the EXPMAX, EXPOLD, EXPRES and TUNE installation options. Consequently, it is possible for the same application to use varying amounts of memory object or Hiperspace storage from run to run. If enough memory object or Hiperspace storage is available, DFSORT uses it exclusively for intermediate storage. If the amount of available storage is insufficient, DFSORT uses a combination of work data sets with Hiperspace or memory object storage, or even work data sets alone.

DFSORT's use of memory object and Hiperspace storage is very dynamic; multiple concurrent sort applications always know each other's storage needs and never try to back their memory objects or Hiperspaces with the same portion of storage. In addition, DFSORT checks the available storage prior to creating each memory object or Hiperspace. Depending on the TUNE installation default each SORT application can be more or less dynamic:

- by checking available storage once and then allocating all available storage expected to be used at initialization, or
- by checking available storage multiple times and allocating it in increments throughout the run.

DFSORT also has the capability of switching from using memory objects or Hiperspaces to using disk work data sets when either of the following occur:

- allocation of an additional memory object or Hiperspace is likely to cause a storage shortage (for more information on how DFSORT uses central storage, see “Recommendations” under “Hipersorting, Memory Object Sorting, and Data Space Sorting” in *z/OS DFSORT Tuning Guide*).
- the total Hipersorting, memory object sorting, and dataspace sorting activity on the system reaches the limits set by the EXPMAX, EXPOLD, and EXPRES installation options.

Memory object and Hipersorting requires that work data sets be available, as well as central storage. DFSORT forces the use of dynamic allocation for memory object sorting or Hipersorting if work data sets are not requested explicitly. For further details, see the MOSIZE , MOWRK and HIPRMAX options of the “OPTION control statement” on page 173.

Work data set devices

The type of device selected for work data sets can have a significant effect on performance. Consider the following when selecting devices for work data sets.

Disk and tape devices

For optimal performance, use disk devices to which little other activity is directed, for work data sets. Specify high-speed devices such as IBM's TotalStorage DS8000 storage subsystems, and avoid specifying tape or virtual (VIO).

Using tape devices for work data sets rather than disk causes significant performance degradation for the following reasons:

- Tape work data sets prevent DFSORT from using its more efficient sorting techniques, Blockset and Peerage/Vale. Disk work data sets allow DFSORT to use these techniques.
- Tape work data sets must be accessed sequentially. Disk data sets can be accessed randomly.
- Disk control units can provide additional features, such as cache fast write, that are not available with tape devices.

Number of devices

Although one work data set is sufficient, using two or more work data sets on separate devices usually reduces the elapsed time of the application significantly. In general, using more than three work data sets does not reduce elapsed time any further, and is only necessary if the work data sets are small or the file size is large.

For optimum allocation of resources such as virtual storage, avoid specifying a large number of work data sets unnecessarily.

No more than 255 work data sets can be specified. If you specify more than 32 work data sets, and the Blockset technique is not selected, a maximum of 32 work data sets is used.

Non-synchronous storage subsystems

Allocation of work data sets on devices attached to non-synchronous storage subsystems can affect the performance of certain DFSORT applications. Whether or not a particular application is affected depends on many factors, the most critical of which is the ratio of input file size to available storage.

In general, to maximize performance, DFSORT needs the following:

- Accurate knowledge of the size of the file being sorted

  In most cases, DFSORT is able to calculate the file size accurately. However, for applications that sort many input tapes that are not managed by DFSMSrmm or a tape management system that uses ICETPEX (especially compacted input tapes) or that use E15 exits that add or delete records, we recommend that you specify the file size using the FILSZ or SIZE parameter.
- Adequate storage relative to the size of the file being sorted

Table 98 on page 799 shows the minimum recommended storage to provide DFSORT based on various input file sizes.

Table 98. Minimum Storage Required for Various File Sizes

Minimum Storage Required for Various File SizesInput file size	Minimum storage
Less than 200MB	4MB
200MB to 500MB	8MB
500MB to 1GB	16MB
More than 1GB	16-32MB

Allocation of work data sets

Dynamic allocation has the following advantages over JCL allocation:

- Installation options can be used to automatically activate dynamic allocation for all sort applications.

  To use JCL allocation, appropriate DD statements must be specified for each individual application.
- As the characteristics (file size, virtual storage, and so on) of an application change over time, DFSORT can automatically optimize the amount of dynamically allocated work space for the application. This eliminates unneeded allocation of disk space.

  JCL allocation is fixed; DFSORT cannot adjust it. Disk space might be wasted.
- Dynamically allocated work data sets are automatically allocated as large format so they can use more than 65535 tracks on a single volume.
- When disk work space requirements are larger than expected, DFSORT can automatically recover by increasing allocation sizes or using additional work data sets.
- As the amount of Hiperspace available to the application varies from run to run, DFSORT can automatically adjust the amount of space it dynamically allocates to complement the amount of Hiperspace. This eliminates unneeded allocation of disk space.

  JCL allocation is fixed; DFSORT cannot adjust it, even if all sorting can be done in Hiperspace. Disk space might be wasted.

Dynamic allocation has one drawback: for certain applications, as described in “File size and dynamic allocation” on page 800, you might need to give DFSORT a reasonable estimate of the input file size. Later, if the input file size for the application increases significantly, you must update the file size estimate accordingly.

However, JCL allocation has a similar drawback, except that it applies to all applications. Unless you overallocate the work data sets initially and waste space, you have to update the JCL allocation when the input file size increases significantly for any application to avoid out-of-space abends.

If you can allocate enough work data set space with JCL to guarantee your applications will never exceed the space allocated, you do not need dynamic allocation. However, since efficient use of disk space is usually desirable, dynamic allocation is recommended over JCL allocation.

For both dynamic allocation and JCL allocation:

- The amount of work space actually used will often be less than the amount allocated. DFSORT tries to minimize dynamic over-allocation while making certain that the application will not fail due to lack of space. With JCL allocation, you could minimize the amount of allocated space manually, but this might require changes to JCL allocation as the characteristics of the application change over time.
- Limiting the virtual storage available to DFSORT can increase the amount of work space required. With a reasonable amount of storage, 4MB for example, DFSORT can sort using a reasonable amount of work space. If storage is limited, more work space might be required. If storage is drastically limited (for example, to 200KB), significantly more work space might be required.

Dynamic allocation of work data sets

Installation options are available to request automatic dynamic allocation of work data sets and to supply defaults for the device type and number of devices.

For certain applications, it is very important to specify a reasonable estimate of the input file size when using dynamic allocation.

Automatic dynamic allocation

Your system programmer has set the DYNAUTO installation option to control whether dynamic allocation is used automatically, or only when requested by the DYNALLOC run-time option.

DYNAUTO also controls whether dynamic allocation or JCL allocation takes precedence when JCL work data sets are specified.

If your system programmer selected DYNAUTO=IGNWKDD, dynamic allocation takes precedence over JCL allocation (JCL work data sets are actually deallocated). If you want the opposite precedence for selected applications, use the run-time option USEWKDD.

If your system programmer selected DYNAUTO=YES, JCL allocation takes precedence over dynamic allocation. If you want the opposite precedence, you must remove the JCL allocation statements.

If your system programmer selected DYNAUTO=NO, dynamic allocation of work data sets is not used unless you specify the DYNALLOC run-time option. JCL allocation takes precedence over dynamic allocation.

Device defaults

When the device type, or the number of devices for dynamic allocation, is not explicitly specified, DFSORT obtains the missing information from the DYNALOC installation option information supplied by your system programmer.

Note: In rare cases where DFSORT is unable to dynamically allocate the required space using the value supplied by d, allocation may be attempted with a more generic unit name (such as 3390) in an effort to allocate the required space on volumes from a different storage group. Refer to "System Managed Storage" in *z/OS DFSORT Tuning Guide* for recommendations on controlling data set allocation.

File size and dynamic allocation

DFSORT bases the amount of work space it dynamically allocates on the number of bytes to be sorted—the input file size. Generally, DFSORT can automatically make an accurate determination of the file size by determining the number of input records. However, DFSORT cannot always determine the input file size accurately in the following cases:

- An E15 user exit routine supplies all input records (an input data set is not present). DFSORT cannot automatically determine the number of records to be inserted.
- An input data set is present, along with an E15 user exit routine. DFSORT can automatically determine the number of records in the input data set, but cannot automatically determine the number of records to be inserted or deleted.
- A spool (DD *) or pipe data set is used as input.
- The input consists of small data sets on tape that are not managed by DFSMSrmm or a tape management system that uses ICETPEX. When the tape data sets are not managed, DFSORT cannot know how much of the tapes are used, so it determines the file size assuming full volumes at the maximum regular density for the drives.
- The Improved Data Recording Capability (IDRC) feature is used for the input device and the tape data sets are not managed by DFSMSrmm or a tape management system that uses ICETPEX.
- Input data sets are members of partitioned data sets. DFSORT cannot determine the size of a member in a partitioned data set. Therefore, when input data sets are partitioned, DFSORT uses the size of the

entire data set as the input file size. This is usually an over-estimation, which leads to over-allocation of work space.

In these circumstances, if the number of records is not supplied by the FILSZ or SIZE option, you will receive message ICE118I. If dynamic allocation of work data sets is used, DFSORT allocates the primary space according to the DYNSPC value in effect. This can result in underallocation or overallocation, possibly leading to wasted space or an out-of-space condition, respectively. DFSORT will also allocate additional work data sets with zero space that can be used to recover from an out of space condition. The percentage of additional work data sets will be the greater of 50% or the DYNAPCT value in effect. To avoid unknown file size situations, you should specify FILSZ=En with a reasonably accurate estimate of the number of records to be sorted. If you cannot specify FILSZ=En, you should use DYNSPC=n to adjust the primary space for dynamically allocated work data sets and DYNAPCT=x to increase the number of additional work data sets, as appropriate.

Note: FILSZ=E0 is ignored.

For variable-length records, DFSORT uses one-half of the maximum record length (LRECL) in conjunction with the number of records to determine the input file size, unless you specify AVGRLEN=n. If your actual average record length is significantly different from one-half of the maximum record length, specifying AVGRLEN=n can prevent DFSORT from overallocating or underallocating dynamic work space.

See “OPTION control statement” on page 173 for more information about the AVGRLEN, DYNSPC, FILSZ, and SIZE options.

Dynamic over-allocation of work space

DFSORT can dynamically over-allocate the work space even when you specify the number of records under the following circumstances:

- When you delete a significant number of records using:
  - An INCLUDE or OMIT statement, or the SKIPREC option. Use of these statements and options does *not* force DFSORT to use a SIZE=En or FILSZ=En specification. DFSORT ignores the En value unless it cannot compute the input file size.
  - One or more partitioned data set members as input. DFSORT uses the size of the entire partitioned data set rather than the size of the member in its calculations. DFSORT ignores any SIZE=En or FILSZ=En value unless it cannot determine the input file size itself.

  You can avoid over-allocation in these cases by specifying SIZE=Un or FILSZ=Un.
- When the average record length of variable-length records is substantially shorter than one-half of the maximum record length. If DFSORT uses your exact or estimated number of records, it uses one-half of the maximum record length to determine the file size. You can avoid over-allocation in this case by specifying AVGRLEN=n.

Dynamic over-allocation of work space can occur when you do not specify the number of records (for example, with small input data sets on tape), or even when you do (for example, when a significant number of records is deleted). In these cases, you might prefer to use JCL allocation of work data sets to control the amount of space allocated. However, there are drawbacks to doing so, as previously explained. If DYNAUTO=IGNWKDD is used, remember to specify run-time option USEWKDD when you want to use JCL allocation of work data sets.

JCL allocation of work data sets

The amount of required work space is dependent on many factors such as virtual storage and type of devices used, but is especially sensitive to the file size of the input data set.

Because of the number of variables involved, an exact formula cannot be given for calculating the needed work space. However, the following guidelines usually hold true:

- For fixed length record (FLR) sort applications, 1.5 to 2 times the input file size is usually adequate.
- For variable-length record (VLR) sort applications, 1.5 to 2.5 times the input file size is usually adequate.

These guidelines assume that a reasonable amount of storage (at least 1M) is available to DFSORT. Limiting the available amount of storage can increase the amount of needed work space.

DFSORT can often run with less than the amount of work space indicated by the guidelines described previously in this section.

To get the best performance using JCL allocation of work data sets:

- Use devices without much activity on them.
- Use high-speed disk devices such as IBM's TotalStorage DS8000 subsystems for work data sets, and avoid using tape or virtual (VIO) for work data sets.
- Allocate space in cylinders.
- Specify contiguous space for each work data set, and make sure there is enough primary space so that secondary space is not needed.
- Allocate two or more work data sets.
- Use multiple channel paths to the devices.
- Use different volumes and separate channel paths for the work data sets and the input/output data sets.

The following table shows the work data set space needed with 4M of storage for applications with various characteristics when Hipersorting, dataspace sorting, and memory object sorting are not used (HIPRMAX=0, DSPSIZE=0 and MOSIZE=0).

Table 99. Work Space Requirements for Various Input Characteristics

Input Data set Characteristics				Cylinders (3390)	
Filesize (MB)	**FLR/VLR**	**Max LRECL**	**BLKSIZE**	**Input Data Set**	**Work Data Set**
4	FLR	80	27920	6	6
4	FLR	160	27840	6	6
20	FLR	80	27920	26	36
20	FLR	160	27840	26	36
20	FLR	1000	27000	26	36
40	FLR	80	27920	51	56
40	FLR	160	27840	51	56
40	FLR	1000	27000	52	56
150	FLR	160	27840	189	198
4	VLR	300	27998	6	9
40	VLR	300	27998	51	63
40	VLR	6000	27998	55	59
150	VLR	300	27998	188	200
150	VLR	6000	27998	203	200

Disk capacity considerations

You can specify a mixture of disk devices for a given sort application. Any IBM disk device supported by your operating system can be used for work data sets.

For best performance, use high-speed devices such as IBM's TotalStorage DS8000 storage subsystems for work data sets.

System performance is improved if work data sets are specified in cylinders, rather than tracks or blocks. Storage on temporary work data sets will be readjusted to cylinders if possible. The number of tracks per cylinder for disk devices is shown in Table 100 on page 803.

Table 100. Number of Tracks per Cylinder for Disk Devices

Number of Tracks per Cylinder for Disk DevicesDevice	Tracks per Cylinder	Maximum Bytes used per Track
3380	15	47476
3390	15	56664
9345	15	46456

If WRKSEC is in effect and the work data set is not allocated to VIO, DFSORT allocates secondary extents as required, even if not requested in the JCL.

Exceeding disk work space capacity

If during sorting, the allocation of secondary space on one of the work data sets fails, the system issues a B37 informational message. DFSORT can recover by allocating space on one of the other work data sets, if one is available.

DFSORT normally allocates secondary extents for work data sets, even if not requested in the JCL. This reduces the probability of exceeding work space capacity.

If the disk work space is not sufficient to perform the sort, DFSORT issues a message and terminates.

Tape capacity considerations

Any IBM tape device supported by your operating system can be used for work space. However, using tape devices for work data sets rather than disk causes significant performance degradation and should, therefore, be avoided.

Three different tape work data set techniques are available to DFSORT: Balanced, Polyphase, and Oscillating. For information on how to calculate their requirements, see Table 101 on page 803.

Note: The value you obtain for "min" is literally a minimum value; if, for example, your input uses a more efficient blocking factor than DFSORT or is spanned, you need more work space. Space requirements are also summarized in Table 101 on page 803. DFSORT selects the most appropriate tape technique using these criteria.

Table 101. Work Space Requirements of the Various Tape Techniques

Work Space Requirements of the Various Tape TechniquesTape Technique	Maximum Input	Work Space Areas Required	Max. No. of Work Areas	Comments
Balanced tape (BALN)	15 volumes	Min=2(V+1)* tape units	32 volumes	Used if more than three work storage tapes are provided and file size is not given.
Polyphase tape (POLY)	1 volume	Min=3 tape units	17 volumes	Used if three work storage tapes are provided.
Oscillating tape (OSCL)	15 volumes	Min=V+2* or 4 tape units, whichever is greater	17 volumes	File size must be given. The tape drive containing SORTIN cannot be used as a work unit.

Note: V = Number of input volumes. Number of input volumes of blocking equals work space blocking.

Exceeding tape work space capacity

At the beginning of a sort using tape work data sets, DFSORT estimates the maximum sort capacity (Nmax) and issues message ICE038I. See the explanation of this message for details.

The value for Nmax printed in message ICE038I is an average value rounded down to the nearest thousand. This value assumes random input. If you have a reversed sequenced file and tape work space, sort capacity may be exceeded at a lower value because of the higher number of partly empty, end-of-string blocks.

For magnetic tape, a tape length of 2400 feet is assumed in calculating Nmax. For tapes of other lengths, the figure is not correct. When tapes with mixed density are used, the smallest density is used in the calculation.

If you specify an actual data set size, and that size is larger than the maximum capacity estimated by the program (Nmax), the program terminates before beginning to sort. If you specify an estimated data set size, or none at all, and the number of records reaches the maximum (Nmax), the program gives control to your routine at user exit E16, if you have written and included one. This routine can direct the program to take one of the following actions:

- Continue sorting the entire input data set with available work space. If the estimate of the input data set size was high, enough work space may remain to complete the application.
- Continue sorting with only part of the input data set; the remainder could be sorted later and the two results merged to complete the application.
- Terminate the program without any further processing.

If you do not include an E16 routine, DFSORT continues to process records for as long as possible. If the work space is sufficient to contain all the records in the input data set, DFSORT completes normally; when work space is not sufficient, DFSORT issues a message and terminates.

The program generates a separate message for each of the three possible error conditions. They are:

1. **ICE041A—N GT NMAX:** Generated before sorting begins when the exact file size is greater than Nmax.
2. **ICE046A—SORT CAPACITY EXCEEDED:** Generated when the sort has used all available work space.
3. **ICE048I—NMAX EXCEEDED:** Generated when the sort has exceeded Nmax and has transferred control to a user-written E16 routine for further action.

The test for message ICE041A is made with the maximum possible calculated value, that is, DFSORT is sure it will fail. In case of doubt, the message is not issued.

Appendix B. Specification/override of DFSORT options

"Installation defaults" on page 17 discusses DFSORT's installation options and environments, and shows you how to use ICETOOL's DEFAULTS operator to list the installation defaults selected at your site.

Listed below are the places in DFSORT where you can specify various options that will override the IBM-supplied defaults. The sources for the options are listed in override order; that is, any option specified in a higher place in the list overrides one specified in a lower place.

`Directly Invoked DFSORT`

- DFSPARM data set
  - PARM options
  - DEBUG and OPTION control statements
  - Other control statements.
- EXEC statement PARM options
- SYSIN data set
  - DEBUG and OPTION control statements
  - Other control statements.
- Installation options set with PARMLIB ICEPRMxx members (JCL, TSO or TDx).
- Installation options set with ICEMAC (JCL, TSO or TDx).

`Program Invoked DFSORT`

- DFSPARM data set
  - PARM options
  - DEBUG and OPTION control statements
  - Other control statements.
- SORTCNTL data set
  - DEBUG and OPTION control statements
  - Other control statements.
- Parameter list
  - DEBUG and OPTION control statements
  - Other control statements.
- Installation options set with PARMLIB ICEPRMxx members (INV, TSOINV or TDx).
- Installation options set with ICEMAC (INV, TSO or TDx).

Note:

1. DFSORT's EXEC PARMs can be specified in the EXEC statement when DFSORT is directly invoked. For example:

   ```
   //S1 EXEC PGM=ICEMAN,PARM='EQUALS,ABEND'
   ```

   DFSORT's EXEC PARMs are ignored if DFSORT is invoked from a program.
2. DFSPARM PARMs can be specified in the DFSPARM data set when DFSORT is directly invoked or invoked from a program. For example:

```
//DFSPARM DD *
  EQUAL,ABEND
```

3. DFSORT's control statements can be specified in the DFSPARM or SYSIN data set when DFSORT is directly invoked. At least one blank must precede the operation field (SORT, MERGE, INCLUDE, and so on). For example:

```
//SYSIN DD *
  SORT FIELDS=(5,4,CH,A)
  OPTION EQUALS
  DEBUG ABEND
```

4. DFSORT's control statements can be specified in the DFSPARM or SORTCNTL data set, or in the parameter list, when DFSORT is invoked from a program. At least one blank must precede the operation field (SORT, MERGE, INCLUDE, and so on). For example:

```
//SORTCNTL DD *
  SORT FIELDS=(5,4,CH,A)
  OPTION EQUALS
  DEBUG ABEND
```

5. For the DEBUG and OPTION statements, override is at the option level. For example, with:

```
//DFSPARM DD *
   OPTION EQUALS
//SYSIN DD *
   OPTION NOEQUALS,SKIPREC=50
```

   EQUALS from DFSPARM overrides NOEQUALS from SYSIN, but SKIPREC=50 from SYSIN in not affected by the OPTION statement in DFSPARM, so both EQUALS and SKIPREC=50 will be used.

   For control statements other than DEBUG and OPTION, override is at the statement level. For example, with:

```
//DFSPARM DD *
  MODS E15=(CHECK,4096,EXIT)
//SYSIN DD *
  MODS E35=(MOVE,2048,EXITX)
```

   the MODS statement in DFSPARM completely overrides the MODS statement in SYSIN, so the E15 exit will be used, but the E35 exit will not.

6. An EFS program or an installation initialization exit (ICEIEXIT) routine can also be used to override options. ICEIEXIT changes override any corresponding changes made by an EFS program.
7. For OUTFIL statements, override is at the ddname level. See “OUTFIL statements notes” on page 360 for details.

Main features of sources of DFSORT options

There are five sources for run-time options that allow you to override your site’s installation defaults. To help you decide which source is most efficient for you for a given situation, compare their main features using the following lists:

DFSPARM data set

- Use with direct or program invocation.
- Overrides all other sources.
- Accepts all DFSORT program control statements, and all EXEC PARM options, including those OPTION statement parameters ignored by SYSIN and SORTCNTL.
- Permits comment statements, blank statements, and remarks.

EXEC statement PARM options

- Use with direct invocation only.
- Accepts all EXEC PARM options, including those equivalent to the OPTION statement parameters ignored by SYSIN and SORTCNTL.

SORTCNTL data set

- Use with program invocation only.
- Accepts all DFSORT program control statements.
- Ignores these OPTION statement parameters: EFS, LIST, NOLIST, LISTX, NOLISTX, LOCALE, MSGPRT, MSGDDN, SMF, SORTDD, SORTIN, SORTOUT, and USEWKDD.
- Permits comment statements, blank statements, and remarks.
- Using multiple parameter lists to rename the SORTCNTL data set permits different control statements to be used for a program that invokes DFSORT more than once.

SYSIN data set

- Use with direct invocation only.
- Accepts all DFSORT program control statements.
- Ignores these OPTION statement parameters: EFS, LIST, NOLIST, LISTX, NOLISTX, LOCALE, MSGPRT, MSGDDN, SMF, SORTDD, SORTIN, SORTOUT, and USEWKDD.
- Permits comment statements, blank statements, and remarks.
- Can contain user exit routines in object deck format for binding or link-editing.

Parameter lists

- Use with program invocation only.
- Extended parameter list accepts all DFSORT program control statements, including those OPTION statement parameters ignored by SYSIN and SORTCNTL.
- 24-bit parameter list accepts a subset of DFSORT program control statements.
- Using multiple parameter lists to rename the SORTCNTL data set permits different control statements to be used for a program that invokes DFSORT more than once.
- Can be used to pass the addresses of any user exits that your program has placed in main storage.

Note: The extended parameter list can perform a superset of the functions in the 24-bit parameter list.

Override tables

The following tables show the possible sources of specification and order of override for individual options.

- The order of override between sources of specification is from left to right. A specification overrides all specifications to its right.
- The order of override within a source is from top to bottom. A specification overrides all specifications below it.
- For DFSPARM, 'keyword' (for example, BSAM) indicates a PARM option, whereas 'operation keyword' (for example, DEBUG BSAM) indicates a control statement option.
- The Function columns indicate which functions (S=sort, M=merge, or C=copy) can use the option.
- Although alias names are available for many of the options, they are not shown here.

Directly invoked DFSORT

Table 102 on page 808 shows where each sort, merge, or copy option may be specified when DFSORT is directly invoked (that is, not invoked by programs).

DFSPARM: PARM options selectively override corresponding options in any other source. DEBUG and OPTION control statement options selectively override corresponding options in EXEC PARM and SYSIN. Control statements other than DEBUG and OPTION completely override corresponding control statements in SYSIN.

EXEC PARM options selectively override options in SYSIN.

SORT and MERGE are considered to be corresponding control statements.

INCLUDE and OMIT are considered to be corresponding control statements.

Table 102. Directly Invoked DFSORT Option Specification/Override. Options are arranged alphabetically on the Installation column. If "NO" is specified in the Installation column, move to the next column to the left and so on. The order of override is from left to right and from top to bottom within a row.

Specified with DFSPARM	Specified with EXEC PARM	Specified with SYSIN	Installation (JCL, TSO or TDx)	Description of Option	Function
NO	NO	NO	ABCODE	ABEND code	S,M,C
DEBUG ABSTP	NO	DEBUG ABSTP	NO	Abnormal stop	S,M,C
ALTSEQ CODE	NO	ALTSEQ CODE	ALTSEQ	Alternate sequence	S,M
ARESALL OPTION ARESALL	ARESALL	OPTION ARESALL	ARESALL	System storage above 16MB virtual	S,M,C
DEBUG NOASSIST	NO	DEBUG NOASSIST	NO	Bypass Sorting Instructions	S
AVGRLEN OPTION AVGRLEN	AVGRLEN	OPTION AVGRLEN	NO	Average record length	S
BSAM DEBUG BSAM	BSAM	DEBUG BSAM	NO	Force BSAM	S,M,C
DEBUG CFW|NOCFW	NO	DEBUG CFW|NOCFW	CFW	Cache fast write	S
OPTION CHALT|NOCHALT	NO	OPTION CHALT|NOCHALT	CHALT	CH field sequence	S,M
OPTION CHECK|NOCHECK	NO	OPTION CHECK|NOCHECK	CHECK	Record count check	S,M,C
CINV|NOCINV OPTION CINV|NOCINV	CINV|NOCINV	OPTION CINV|NOCINV	CINV	Control interval access	S,M,C
COBEXIT OPTION COBEXIT	COBEXIT	OPTION COBEXIT	COBEXIT	COBOL library	S,M,C
COLLKEY OPTION COLLKEY	COLLKEY	OPTION COLLKEY	COLLKEY	Collation Version for Unicode Data	S,M,C
INCLUDE|OMIT COND|FORMAT	NO	INCLUDE|OMIT COND|FORMAT	NO	Include|Omit fields	S,M,C
OPTION COPY SORT|MERGE FIELDS	NO	OPTION COPY SORT|MERGE FIELDS	NO	Copy records	C
DEBUG CTRx	NO	DEBUG CTRx	NO	ABEND record count	S,M
NO	NO	NO	Time-of-day for activation	Simulate SORTDIAG DD Statement	S,M,C
NO	NO	NO	DIAGSIM	Simulate SORTDIAG DD Statement	S,M,C
DSA OPTION DSA	DSA	OPTION DSA	DSA	Dynamic storage adjustment limit	S
DSPSIZE OPTION DSPSIZE	DSPSIZE	OPTION DSPSIZE	DSPSIZE	Dataspace sorting	S
DYNALLOC OPTION DYNALLOC SORT DYNALLOC	DYNALLOC	OPTION DYNALLOC SORT DYNALLOC	DYNALOC[1]	Dynamic SORTWKs	S

Table 102. Directly Invoked DFSORT Option Specification/Override. Options are arranged alphabetically on the Installation column. If "NO" is specified in the Installation column, move to the next column to the left and so on. The order of override is from left to right and from top to bottom within a row. *(continued)*

Specified with DFSPARM	Specified with EXEC PARM	Specified with SYSIN	Installation (JCL, TSO or TDx)	Description of Option	Function
DYNALLOC OPTION DYNALLOC|USEWKDD SORT DYNALLOC	DYNALLOC	OPTION DYNALLOC SORT DYNALLOC	DYNAUTO	Automatic dynamic allocation	S
DYNAPCT OPTION DYNAPCT	DYNAPCT	OPTION DYNAPCT	DYNAPCT	Additional work data sets	S
DYNSPC OPTION DYNSPC	DYNSPC	OPTION DYNSPC	DYNSPC	Dynamic allocation default space	S
EFS OPTION EFS	EFS	NO[2]	EFS	EFS program specified	S,M,C
NO	NO	NO	ENABLE	Enable Time-of-Day modules	S,M,C
EQUALS|NOEQUALS OPTION EQUALS|NOEQUALS SORT|MERGE EQUALS|NOEQUALS	EQUALS|NOEQUALS	OPTION EQUALS|NOEQUALS SORT|MERGE EQUALS|NOEQUALS	EQUALS	Equal record order	S,M
DEBUG EQUCOUNT	NO	DEBUG EQUCOUNT	NO	Equal key count message	S
ABEND|NOABEND DEBUG ABEND|NOABEND	ABEND|NOABEND	DEBUG ABEND|NOABEND	ERET	Error action	S,M,C
DEBUG ESTAE|NOESTAE	NO	DEBUG ESTAE|NOESTAE	ESTAE	ESTAE routine	S,M,C
OPTION EXITCK	NO	OPTION EXITCK	EXITCK	E15/E35 return code checking	S,M,C
NO	NO	NO	EXPMAX	Available expanded storage limit for all DFSORT Hiperspaces	S
NO	NO	NO	EXPOLD	Old expanded storage limit for all DFSORT Hiperspaces	S
NO	NO	NO	EXPRES	Available expanded storage reserved for non-Hipersorting use	S
E15=COB PARM E35=COB MODS Exx|HILEVEL=YES	E15=COB E35=COB	MODS Exx|HILEVEL=YES	NO	User Exit Exx (xx=11,15-19, 31,35,37-39, and 61)	S,M,C[3]
INREC parameters	NO	INREC parameters	NO	INREC reformatting	S,M,C
JOINKEYS parameters	NO	JOINKEYS parameters	NO	JOINKEYS processing	S,C
JOIN parameters	NO	JOIN parameters	NO	JOIN options	S,C
OUTREC parameters	NO	OUTREC parameters	NO	OUTREC reformatting	S,M,C
REFORMAT parameters	NO	REFORMAT parameters	NO	REFORMAT fields	S,C
SORT|MERGE FIELDS|FORMAT	NO	SORT|MERGE FIELDS|FORMAT	NO	Control fields	S,M
SUM FIELDS/FORMAT	NO	SUM FIELDS/FORMAT	NO	Sum fields	S,M
MERGE FILES	NO	MERGE FILES	NO	Merge input files	M
FILSZ OPTION FILSZ|SIZE SORT|MERGE FILSZ|SIZE	FILSZ	OPTION FILSZ|SIZE SORT|MERGE FILSZ|SIZE	FSZEST	File size	S,M
HIPRMAX OPTION HIPRMAX	HIPRMAX	OPTION HIPRMAX	HIPRMAX	Hipersorting	S
NO	NO	NO	IDRCPCT	IDRC compaction	S
NO	NO	NO	IEXIT	ICEIEXIT	S,M,C
OPTION CKPT[4] SORT CKPT[4]	NO	OPTION CKPT[4] SORT CKPT[4]	IGNCKPT	Checkpoints	S

Table 102. Directly Invoked DFSORT Option Specification/Override. Options are arranged alphabetically on the Installation column. If "NO" is specified in the Installation column, move to the next column to the left and so on. The order of override is from left to right and from top to bottom within a row. *(continued)*

Specified with DFSPARM	Specified with EXEC PARM	Specified with SYSIN	Installation (JCL, TSO or TDx)	Description of Option	Function
NO	NO	NO	IOMAXBF	Maximum SORTIN/ SORTOUT data set buffer space	S,M,C
RECORD LENGTH	NO	RECORD LENGTH	NO	Record lengths	S,M,C
LIST|NOLIST OPTION LIST|NOLIST	LIST|NOLIST	NO[2]	LIST	Print DFSORT control statements[5]	S,M,C
LISTX|NOLISTX OPTION LISTX|NOLISTX	LISTX|NOLISTX	NO[2]	LISTX	Print control statements returned by an EFS program[5]	S,M,C
LOCALE OPTION LOCALE	LOCALE	NO[2]	LOCALE	Locale processing	S,M,C
NO	NO	NO	MAXLIM	Maximum storage below 16MB virtual[6]	S,M,C
OPTION MERGEIN	NO	NO[2]	NO	Alternate MERGE ddnames	M
NO	NO	NO	MINLIM	Minimum storage	S,M,C
MOSIZE OPTION MOSIZE	MOSIZE	OPTION MOSIZE	MOSIZE	Memory object sorting	S
MOWRK|NOMOWRK OPTION MOWRK|NOMOWRK	MOWRK|NOMOWRK	OPTION MOWRK|NOMOWRK	MOWRK	Memory objects as work storage	S
MSGDDN OPTION MSGDDN	MSGDDN	NO[2]	MSGDDN	Alternate message data set	S,M,C
NO	NO	NO	MSGCON	Write messages on master console	S,M,C
MSGPRT OPTION MSGPRT	MSGPRT	NO[2]	MSGPRT	Print messages	S,M,C
OPTION NOBLKSET	NO	OPTION NOBLKSET	NO	Bypass Blockset	S,M
NO	NO	NO	NOMSGDD	Action when message data set missing	S,M,C
NULLOUT OPTION NULLOUT	NULLOUT	OPTION NULLOUT	NULLOUT	Action when no records for SORTOUT	S,M,C
ODMAXBF OPTION ODMAXBF	ODMAXBF	OPTION ODMAXBF	ODMAXBF	Maximum OUTFIL data set buffer space	S,M,C
OUTFIL[9]	OUTFIL[9]	OUTFIL[9]	NO	OUTFIL processing	S,M,C
OUTREL|NOOUTREL OPTION NOOUTREL	OUTREL|NOOUTREL	OPTION NOOUTREL	OUTREL	Release output data set space	S,M,C
OPTION NOOUTSEC	NO	OPTION NOOUTSEC	OUTSEC	Output data set secondary allocation	S,M,C
NO	NO	NO	OVERRGN	Storage over REGION	S,M,C
OVFLO OPTION OVFLO	OVFLO	OPTION OVFLO	OVFLO	Summary fields overflow action	S,M
PAD OPTION PAD	PAD	OPTION PAD	PAD	DFSORT LRECL padding action	S,M,C
NO	NO	NO	PARMDDN	Alternate ddname for DFSPARM	S,M,C
RESALL OPTION RESALL	RESALL	OPTION RESALL	RESALL	System reserved storage[6]	S,M,C

Table 102. Directly Invoked DFSORT Option Specification/Override. Options are arranged alphabetically on the Installation column. If "NO" is specified in the Installation column, move to the next column to the left and so on. The order of override is from left to right and from top to bottom within a row. *(continued)*

Specified with DFSPARM	Specified with EXEC PARM	Specified with SYSIN	Installation (JCL, TSO or TDx)	Description of Option	Function
RESET|NORESET OPTION RESET|NORESET	RESET|NORESET	OPTION RESET|NORESET	RESET	NEW or MOD VSAM output	S,M,C
SDB OPTION SDB	SDB	OPTION SDB	SDB	System-determined output data set block size	S,M,C
NO	NO	NO	SDBMSG	System-determined block size for message and list data sets	S,M,C
SIZE OPTION MAINSIZE	SIZE	OPTION MAINSIZE	SIZE	Storage	S,M,C
SKIPREC OPTION SKIPREC SORT SKIPREC	SKIPREC	OPTION SKIPREC SORT SKIPREC	NO	Skip records	S,C
OPTION SMF	NO	NO	SMF	SMF records	S,M,C
SOLRF|NOSOLRF OPTION SOLRF|NOSOLRF	SOLRF|NOSOLRF	OPTION SOLRF|NOSOLRF	SOLRF	SORTOUT length	S,M,C
OPTION SORTDD	NO	NO[2]	NO	ddname prefix	S,M,C
OPTION SORTIN[7]	NO	NO[2]	NO	Alternate SORTIN ddname	S,C
NO	NO	NO	SORTLIB	Conventional modules library	S,M
OPTION SORTOUT[8]	NO	NO[2]	NO	Alternate SORTOUT ddname	S,M,C
SPANINC OPTION SPANINC	SPANINC	OPTION SPANINC	SPANINC	Incomplete spanned records action	S,M,C
STOPAFT OPTION STOPAFT SORT STOPAFT	STOPAFT	OPTION STOPAFT SORT STOPAFT	NO	Input limit	S,C
NO	NO	NO	SVC	DFSORT SVC Information	S,M,C
SZERO|NOSZERO OPTION SZERO|NOSZERO	SZERO|NOSZERO	OPTION SZERO|NOSZERO	SZERO	Signed or unsigned zero	S,M,C
NO	NO	NO	TEXIT	ICETEXIT	S,M,C
NO	NO	NO	TMAXLIM	Maximum storage above and below 16MB virtual[6]	S,M,C
NO	NO	NO	TUNE	Optimize central storage or disk work space	S
TRUNC OPTION TRUNC	TRUNC	OPTION TRUNC	TRUNC	DFSORT LRECL truncation action	S,M,C
RECORD TYPE	NO	RECORD TYPE	NO	Record format	S,M,C
VERIFY|NOVERIFY OPTION VERIFY|NOVERIFY	VERIFY|NOVERIFY	OPTION VERIFY|NOVERIFY	VERIFY	Sequence check	S,M
NO	NO	NO	VIO	SORTWK virtual I/O	S
VLLONG|NOVLLONG OPTION VLLONG|NOVLLONG	VLLONG|NOVLLONG	OPTION VLLONG|NOVLLONG	VLLONG	Truncate long output records	S,M,C
VLSCMP|NOVLSCMP OPTION VLSCMP|NOVLSCMP	VLSCMP|NOVLSCMP	OPTION VLSCMP|NOVLSCMP	VLSCMP	Pad short compare fields	S,M,C
VLSHRT|NOVLSHRT OPTION VLSHRT|NOVLSHRT	VLSHRT|NOVLSHRT	OPTION VLSHRT|NOVLSHRT	VLSHRT	Action for short control or compare fields	S,M,C

Table 102. Directly Invoked DFSORT Option Specification/Override. Options are arranged alphabetically on the Installation column. If "NO" is specified in the Installation column, move to the next column to the left and so on. The order of override is from left to right and from top to bottom within a row. *(continued)*

Specified with DFSPARM	Specified with EXEC PARM	Specified with SYSIN	Installation (JCL, TSO or TDx)	Description of Option	Function
NO	NO	NO	VSAMBSP	VSAM buffer space	S
VSAMEMT|NVSAMEMT OPTION VSAMEMT|NVSAMEMT	VSAMEMT|NVSAMEMT	OPTION VSAMEMT|NVSAMEMT	VSAMEMT	Emty VSAM input	S,M,C
VSAMIO|NOVSAMIO OPTION VSAMIO|NOVSAMIO	VSAMIO|NOVSAMIO	OPTION VSAMIO|NOVSAMIO	VSAMIO	Same VSAM input and output	S
WRKREL|NOWRKREL OPTION WRKREL|NOWRKREL	WRKREL|NOWRKREL	OPTION WRKREL|NOWRKREL	WRKREL	Release SORTWK space	S
WRKSEC|NOWRKSEC OPTION WRKSEC|NOWRKSEC	WRKSEC|NOWRKSEC	OPTION WRKSEC|NOWRKSEC	WRKSEC	SORTWK secondary allocation	S
Y2PAST OPTION Y2PAST SORT|MERGE Y2PAST	Y2PAST	OPTION Y2PAST SORT|MERGE Y2PAST	Y2PAST	Set century window	S,M,C
ZDPRINT|NZDPRINT OPTION ZDPRINT|NZDPRINT	ZDPRINT|NZDPRINT	OPTION ZDPRINT|NZDPRINT	ZDPRINT	ZD SUM results	S,M

Notes to directly invoked DFSORT table

1

Does not request dynamic allocation; only supplies defaults.

2

Not used in SYSIN.

3

All functions do not apply to all exits. See Table 66 on page 470 and Table 67 on page 470 for applicable exits.

4

Not used if Blockset is selected and IGNCKPT=YES was specified.

5

Not used if MSGPRT=NONE is in effect; in this case control statements are not printed.

6

Not used unless MAINSIZE=MAX is in effect.

7

Overrides SORTDD for the SORT input ddname.

8

Overrides SORTDD for the SORT output ddname.

9

Override is at the ddname level.

Program invoked DFSORT with the extended parameter list

Table 103 on page 813 shows where each sort, merge, or copy option may be specified when DFSORT is program invoked and an extended parameter list is passed to it.

DFSPARM: PARM options selectively override corresponding options in any other source. DEBUG and OPTION control statement options selectively override corresponding options in SORTCNTL and the Parameter List. Control statements other than DEBUG and OPTION completely override corresponding control statements in SORTCNTL and the Parameter List.

SORTCNTL: DEBUG and OPTION control statement options selectively override corresponding options in the Parameter List. Control statements other than DEBUG and OPTION completely override corresponding control statements in the Parameter List.

SORT and MERGE are considered to be corresponding control statements.

INCLUDE and OMIT are considered to be corresponding control statements.

Table 103. Extended Parameter List DFSORT Option Specification/Override. Options are arranged alphabetically on the Installation column. If "NO" is specified in the Installation column, move to the next column to the left and so on. The order of override is from left to right and from top to bottom within a row.

Specified with DFSPARM	Specified with SORTCNTL	Specified with Extended Parameter List	Installation (INV, TSOINV or TDx)	Description of Option	Function
NO	NO	NO	ABCODE	ABEND code	S,M,C
DEBUG ABSTP	DEBUG ABSTP	DEBUG ABSTP	NO	Abnormal stop	S,M,C
ALTSEQ CODE	ALTSEQ CODE	Offset 16 entry ALTSEQ CODE	ALTSEQ	Alternate sequence	S,M
ARESALL OPTION ARESALL	OPTION ARESALL	OPTION ARESALL	ARESALL	System storage above 16MB virtual	S,M,C
OPTION ARESINV	OPTION ARESINV	OPTION ARESINV	ARESINV	Storage above 16MB virtual for invoking program	S,M,C
DEBUG NOASSIST	DEBUG NOASSIST	DEBUG NOASSIST	NO	Bypass Sorting Instructions	S
AVGRLEN OPTION AVGRLEN	OPTION AVGRLEN	OPTION AVGRLEN	NO	Average record length	S
BSAM DEBUG BSAM	DEBUG BSAM	DEBUG BSAM	NO	Force BSAM	S,M,C
DEBUG CFW|NOCFW	DEBUG CFW|NOCFW	DEBUG CFW|NOCFW	CFW	Cache fast write	S
OPTION CHALT|NOCHALT	OPTION CHALT|NOCHALT	OPTION CHALT|NOCHALT	CHALT	CH field sequence	S,M
OPTION CHECK|NOCHECK	OPTION CHECK|NOCHECK	OPTION CHECK|NOCHECK	CHECK	Record count check	S,M,C
CINV|NOCINV OPTION CINV|NOCINV	OPTION CINV|NOCINV	OPTION CINV|NOCINV	CINV	Control interval access	S,M,C
COBEXIT OPTION COBEXIT	OPTION COBEXIT	OPTION COBEXIT	COBEXIT	COBOL library	S,M,C
COLLKEY OPTION COLLKEY	COLLKEY	OPTION COLLKEY	COLLKEY	Collation Version for Unicode Data	S,M,C
INCLUDE|OMIT COND|FORMAT	INCLUDE|OMIT COND|FORMAT	INCLUDE|OMIT COND|FORMAT	NO	Include|Omit fields	S,M,C
OPTION COPY SORT|MERGE FIELDS	OPTION COPY SORT|MERGE FIELDS[2]	OPTION COPY SORT|MERGE FIELDS	NO	Copy records	C
DEBUG CTRx	DEBUG CTRx	DEBUG CTRx	NO	ABEND record count	S,M
NO	NO	NO	day	Time-of-day for activation	S,M,C
NO	NO	NO	DIAGSIM	Simulate SORTDIAG DD statement	S,M,C
DSA OPTION DSA	OPTION DSA	OPTION DSA	DSA	Dynamic storage adjustment limit	S
DSPSIZE OPTION DSPSIZE	OPTION DSPSIZE	OPTION DSPSIZE	DSPSIZE	dataspace sorting	S
DYNALLOC OPTION DYNALLOC SORT DYNALLOC	OPTION DYNALLOC SORT DYNALLOC[2]	OPTION DYNALLOC SORT DYNALLOC	DYNALOC[1]	Dynamic SORTWKs	S
DYNALLOC OPTION DYNALLOC| USEWKDD SORT DYNALLOC	OPTION DYNALLOC SORT DYNALLOC	OPTION DYNALLOC| USEWKDD SORT DYNALLOC	DYNAUTO	Automatic DYNALLOC	S
DYNAPCT OPTION DYNAPCT	OPTION DYNAPCT	OPTION DYNAPCT	DYNAPCT	Additional work data sets	S
DYNSPC OPTION DYNSPC	OPTION DYNSPC	OPTION DYNSPC	DYNSPC	Dynamic allocation default space	S
EFS OPTION EFS	NO[3]	OPTION EFS	EFS	EFS program specified	S,M,C
NO	NO	NO	ENABLE	Enable Time-of-Day modules	S,M,C

Table 103. Extended Parameter List DFSORT Option Specification/Override. Options are arranged alphabetically on the Installation column. If "NO" is specified in the Installation column, move to the next column to the left and so on. The order of override is from left to right and from top to bottom within a row. *(continued)*

Specified with DFSPARM	Specified with SORTCNTL	Specified with Extended Parameter List	Installation (INV, TSOINV or TDx)	Description of Option	Function
EQUALS|NOEQUALS OPTION EQUALS|NOEQUALS SORT|MERGE EQUALS|NOEQUALS	OPTION EQUALS|NOEQUALS SORT|MERGE EQUALS|NOEQUALS[2]	OPTION EQUALS|NOEQUALS SORT|MERGE EQUALS|NOEQUALS	EQUALS	Equal record order	S,M
DEBUG EQUCOUNT	DEBUG EQUCOUNT	DEBUG EQUCOUNT	NO	Equal key count message	S
ABEND|NOABEND DEBUG ABEND|NOABEND	DEBUG ABEND|NOABEND	DEBUG ABEND|NOABEND	ERET	Error action	S,M,C
DEBUG ESTAE|NOESTAE	DEBUG ESTAE|NOESTAE	DEBUG ESTAE|NOESTAE	ESTAE	ESTAE routine	S,M,C
OPTION EXITCK	OPTION EXITCK	OPTION EXITCK	EXITCK	E15/E35 return code checking	S,M,C
NO	NO	NO	EXPMAX	Available expanded storage limit for all DFSORT Hiperspaces	S
NO	NO	NO	EXPOLD	Old expanded storage limit for all DFSORT Hiperspaces	S
NO	NO	NO	EXPRES	Available expanded storage reserved for non-Hipersorting use	S
E15=COB MODS E15[4]|HILEVEL=YES	MODS E15[4]|HILEVEL=YES	Offset 4 entry[4] MODS E15[4]|HILEVEL=YES	NO	Exit E15	S,C
MODS E18[4]	MODS E18[4]	Offset 24 entry[4] MODS E18[4]	NO	Exit E18	S
NO	NO	Offset 4 entry	NO	Exit E32	M
E35=COB MODS E35[4]|HILEVEL=YES	MODS E35[4]|HILEVEL=YES	Offset 8 entry[4] MODS E35[4]|HILEVEL=YES	NO	Exit E35	S,M,C
MODS E39[4]	MODS E39[4]	Offset 28 entry[4] MODS E39[4]	NO	Exit E39	S,M,C
MODS Exx	MODS Exx	MODS Exx	NO	User Exit Exx (xx=11,16,17,19, 31,37,38, and 61)	S,M,C[5]
INREC parameters	INREC parameters	INREC parameters	NO	INREC reformatting	S,M,C
JOINKEYS parameters	JOINKEYS parameters	JOINKEYS parameters	NO	JOINKEYS processing	S,C
JOIN parameters	JOIN parameters	JOIN parameters	NO	JOIN options	S.C
OUTREC parameters	OUTREC parameters	OUTREC parameters	NO	OUTREC reformatting	S,M,C
REFORMAT parameters	REFORMAT parameters	REFORMAT parameters	NO	REFORMAT fields	S,C
SORT|MERGE FIELDS|FORMAT	SORT|MERGE FIELDS|FORMAT	SORT|MERGE FIELDS|FORMAT	NO	Control fields	S,M
SUM FIELDS|FORMAT	SUM FIELDS|FORMAT	SUM FIELDS|FORMAT	NO	Sum fields	S,M
MERGE FILES	MERGE FILES	MERGE FILES	NO	Merge input files	M
FILSZ OPTION FILSZ|SIZE SORT|MERGE FILSZ|SIZE	OPTION FILSZ|SIZE SORT|MERGE FILSZ|SIZE[2]	OPTION FILSZ|SIZE SORT|MERGE FILSZ|SIZE	FSZEST	File size	S,M
NO	NO	NO	GENER	IEBGENER name	C
NO	NO	NO	GNPAD	ICEGENER LRECL padding action	C
NO	NO	NO	GNTRUNC	ICEGENER LRECL truncation action	C
HIPRMAX OPTION HIPRMAX	OPTION HIPRMAX	OPTION HIPRMAX	HIPRMAX	Hipersorting	S
NO	NO	NO	IDRCPCT	IDRC compaction	S
NO	NO	NO	IEXIT	ICEIEXIT	S,M,C
OPTION CKPT[6] SORT|MERGE CKPT[6]	OPTION CKPT[6] SORT|MERGE CKPT[2,6]	OPTION CKPT[6] SORT|MERGE CKPT[6]	IGNCKPT	Checkpoints	S

Table 103. Extended Parameter List DFSORT Option Specification/Override. Options are arranged alphabetically on the Installation column. If "NO" is specified in the Installation column, move to the next column to the left and so on. The order of override is from left to right and from top to bottom within a row. *(continued)*

Specified with DFSPARM	Specified with SORTCNTL	Specified with Extended Parameter List	Installation (INV, TSOINV or TDx)	Description of Option	Function
NO	NO	NO	IOMAXBF	Maximum SORTIN/ SORTOUT data set buffer space	S,M,C
RECORD LENGTH	RECORD LENGTH	RECORD LENGTH	NO	Record lengths	S,M,C
LIST|NOLIST OPTION LIST|NOLIST	NO[3]	OPTION LIST|NOLIST	LIST	Print DFSORT control statements[7]	S,M,C
LISTX|NOLISTX OPTION LISTX|NOLISTX	NO[3]	OPTION LISTX|NOLISTX	LISTX	Print control statements returned by an EFS program[7]	S,M,C
LOCALE OPTION LOCALE	NO[3]	OPTION LOCALE	LOCALE	Locale processing	S,M,C
NO	NO	NO	MAXLIM	Maximum storage below 16MB virtual[8]	S,M,C
OPTION MERGEIN	NO [3]	OPTION MERGEIN	NO	Alternate MERGE ddnames	M
NO	NO	NO	MINLIM	Minimum storage	S,M,C
MOSIZE OPTION MOSIZE	OPTION MOSIZE	OPTION MOSIZE	MOSIZE	Memory object sorting	S
MOWRK|NOMOWRK OPTION MOWRK|NOMOWRK	OPTION MOWRK|NOMOWRK	OPTION MOWRK|NOMOWRK	MOWRK	Memory objects as work storage	S
MSGDDN OPTION MSGDDN	NO[3]	OPTION MSGDDN	MSGDDN	Alternate message ddname	S,M,C
NO	NO	NO	MSGCON	Write messages on master console	S,M,C
MSGPRT OPTION MSGPRT	NO[3]	OPTION MSGPRT	MSGPRT	Print messages	S,M,C
OPTION NOBLKSET	OPTION NOBLKSET	OPTION NOBLKSET	NO	Bypass Blockset	S,M
NO	NO	NO	NOMSGDD	Action when message data set missing	S,M,C
NULLOUT OPTION NULLOUT	OPTION NULLOUT	OPTION NULLOUT	NULLOUT	Action when no records for SORTOUT	S,M,C
ODMAXBF OPTION ODMAXBF	OPTION ODMAXBF	OPTION ODMAXBF	ODMAXBF	Maximum OUTFIL data set buffer space	S,M,C
OUTFIL[11]	OUTFIL[11]	OUTFIL[11]	NO	OUTFIL processing	S,M,C
OUTREL|NOOUTREL OPTION NOOUTREL	OPTION NOOUTREL	OPTION NOOUTREL	OUTREL	Release output data set space	S,M,C
OPTION NOOUTSEC	OPTION NOOUTSEC	OPTION NOOUTSEC	OUTSEC	Output data set secondary allocation	S,M,C
NO	NO	NO	OVERRGN	Storage over REGION	S,M,C
OVFLO OPTION OVFLO	OPTION OVFLO	OPTION OVFLO	OVFLO	Summary fields overflow action	S,M
PAD OPTION PAD	OPTION PAD	OPTION PAD	PAD	DFSORT LRECL padding action	S,M,C
NO	NO	NO	PARMDDN	Alternate ddname for DFSPARM	S,M,C
RESALL OPTION RESALL	OPTION RESALL	OPTION RESALL	RESALL	System reserved storage[8]	S,M,C
RESET|NORESET OPTION RESET|NORESET	OPTION RESET|NORESET	OPTION RESET|NORESET	RESET	NEW or MOD VSAM output	S,M,C
OPTION RESINV	OPTION RESINV	OPTION RESINV	RESINV	Program reserved storage[8]	S,M,C
SDB OPTION SDB	OPTION SDB	OPTION SDB	SDB	System-determined output data set block size	S,M,C

Table 103. Extended Parameter List DFSORT Option Specification/Override. Options are arranged alphabetically on the Installation column. If "NO" is specified in the Installation column, move to the next column to the left and so on. The order of override is from left to right and from top to bottom within a row. *(continued)*

Specified with DFSPARM	Specified with SORTCNTL	Specified with Extended Parameter List	Installation (INV, TSOINV or TDx)	Description of Option	Function
NO	NO	NO	SDBMSG	System-determined block size for message and list data sets	S,M,C
SIZE OPTION MAINSIZE	OPTION MAINSIZE	OPTION MAINSIZE	SIZE	Storage	S,M,C
SKIPREC OPTION SKIPREC SORT|MERGE SKIPREC	OPTION SKIPREC SORT|MERGE SKIPREC[2]	OPTION SKIPREC SORT|MERGE SKIPREC	NO	Skip records	S,C
OPTION SMF	NO	OPTION SMF	SMF	SMF records	S,M,C
SOLRF|NOSOLRF OPTION SOLRF|NOSOLRF	OPTION SOLRF|NOSOLRF	OPTION SOLRF|NOSOLRF	SOLRF	SORTOUT length	S,M,C
OPTION SORTDD	NO[3]	OPTION SORTDD	NO	ddname prefix	S,M,C
OPTION SORTIN[9]	NO[3]	OPTION SORTIN[9]	NO	Alternate SORTIN ddname	S,C
NO	NO	NO	SORTLIB	Conventional modules library	S,M
OPTION SORTOUT[10]	NO[3]	OPTION SORTOUT[10]	NO	Alternate SORTOUT ddname	S,M,C
SPANINC OPTION SPANINC	OPTION SPANINC	OPTION SPANINC	SPANINC	Incomplete spanned records action	S,M,C
STOPAFT OPTION STOPAFT SORT|MERGE STOPAFT	OPTION STOPAFT SORT|MERGE STOPAFT[2]	OPTION STOPAFT SORT|MERGE STOPAFT	NO	Input limit	S,C
NO	NO	NO	SVC	DFSORT SVC information	S,M,C
SZERO|NOSZERO OPTION SZERO|NOSZERO	OPTION SZERO|NOSZERO	OPTION SZERO|NOSZERO	SZERO	Signed or unsigned zero	S,M,C
NO	NO	NO	TEXIT	ICETEXIT	S,M,C
NO	NO	NO	TMAXLIM	Maximum storage above and below 16MB virtual[8]	S,M,C
NO	NO	NO	TUNE	Optimize central storage or disk work space	S
TRUNC OPTION TRUNC	OPTION TRUNC	OPTION TRUNC	TRUNC	DFSORT LRECL truncation action	S,M,C
RECORD TYPE	RECORD TYPE	RECORD TYPE	NO	Record format	S,M,C
VERIFY|NOVERIFY OPTION VERIFY|NOVERIFY	OPTION VERIFY|NOVERIFY	OPTION VERIFY|NOVERIFY	VERIFY	Sequence check	S,M
NO	NO	NO	VIO	SORTWK virtual I/O	S
VLLONG|NOVLLONG OPTION VLLONG|NOVLLONG	OPTION VLLONG|NOVLLONG	OPTION VLLONG|NOVLLONG	VLLONG	Ttruncate long output records	S,M,C
VLSCMP|NOVLSCMP OPTION VLSCMP|NOVLSCMP	OPTION VLSCMP|NOVLSCMP	OPTION VLSCMP|NOVLSCMP	VLSCMP	Pad short compare fields	S,M,C
VLSHRT|NOVLSHRT OPTION VLSHRT|NOVLSHRT	OPTION VLSHRT|NOVLSHRT	OPTION VLSHRT|NOVLSHRT	VLSHRT	Action for short control field or compare field	S,M,C
NO	NO	NO	VSAMBSP	VSAM buffer space	S
VSAMEMT|NVSAMEMT OPTION VSAMEMT| NVSAMEMT	OPTION VSAMEMT|NVSAMEMT	OPTION VSAMEMT|NVSAMEMT	VSAMEMT	Empty VSAM input	S,M,C
VSAMIO|NOVSAMIO OPTION VSAMIO|NOVSAMIO	OPTION VSAMIO|NOVSAMIO	OPTION VSAMIO|NOVSAMIO	VSAMIO	Same VSAM input and output	S
WRKREL|NOWRKREL OPTION WRKREL|NOWRKREL	OPTION WRKREL|NOWRKREL	OPTION WRKREL|NOWRKREL	WRKREL	Release SORTWK space	S

Table 103. Extended Parameter List DFSORT Option Specification/Override. Options are arranged alphabetically on the Installation column. If "NO" is specified in the Installation column, move to the next column to the left and so on. The order of override is from left to right and from top to bottom within a row. *(continued)*

Specified with DFSPARM	Specified with SORTCNTL	Specified with Extended Parameter List	Installation (INV, TSOINV or TDx)	Description of Option	Function
WRKSEC|NOWRKSEC OPTION WRKSEC| NOWRKSEC	OPTION WRKSEC|NOWRKSEC	OPTION WRKSEC|NOWRKSEC	WRKSEC	SORTWK secondary allocation	S
Y2PAST OPTION Y2PAST SORT|MERGE Y2PAST	OPTION Y2PAST SORT|MERGE Y2PAST[2]	OPTION Y2PAST SORT|MERGE Y2PAST	Y2PAST	Set century window	S,M,C
ZDPRINT|NZDPRINT OPTION ZDPRINT|NZDPRINT	OPTION ZDPRINT|NZDPRINT	OPTION ZDPRINT|NZDPRINT	ZDPRINT	ZD SUM results	S,M

Notes to extended parameter list table

1

Does not request dynamic allocation; only supplies defaults.

2

Does not override corresponding option in an OPTION statement specified via the extended parameter list.

3

Not used in SORTCNTL.

4

DFSORT terminates if the exit is specified via the parameter list entry and the exit is specified in a MODS statement.

5

All functions do not apply to all exits. See Table 66 on page 470 and Table 67 on page 470 for applicable exits.

6

Not used if Blockset is selected and IGNCKPT=YES was specified.

7

Not used if MSGPRT=NONE is in effect; in this case control statements are not printed.

8

Not used unless MAINSIZE=MAX is in effect.

9

Overrides SORTDD for the sort input ddname.

10

Overrides SORTDD for the sort output ddname.

11

Override is at the ddname level.

Program invoked DFSORT with the 24-bit parameter list

Table 104 on page 818 shows where each sort, merge, or copy option may be specified when DFSORT is program invoked and a 24-bit parameter list is passed to it.

DFSPARM: PARM options selectively override corresponding options in any other source. DEBUG and OPTION control statement options selectively override corresponding options in SORTCNTL and the Parameter List. Control statements other than DEBUG and OPTION completely override corresponding control statements in SORTCNTL and the Parameter List.

SORTCNTL: DEBUG control statement options selectively override corresponding options in the Parameter List. Control statements other than DEBUG completely override corresponding control statements in the Parameter List.

SORT and MERGE are considered to be corresponding control statements.

INCLUDE and OMIT are considered to be corresponding control statements.

Table 104. 24-Bit List DFSORT Option Specification/Override. Options are arranged alphabetically on the Installation column. If "NO" is specified in the Installation column, move to the next column to the left and so on.

Specified with DFSPARM	Specified with SORTCNTL	Specified with 24-Bit List	Installation (INV, TSOINV or TDx)	Description of Option	Function
NO	NO	NO	ABCODE	ABEND code	S,M,C
DEBUG ABSTP	DEBUG ABSTP	DEBUG ABSTP	NO	Abnormal stop	S,M,C
ALTSEQ CODE	ALTSEQ CODE	X'F6' entry ALTSEQ CODE	ALTSEQ	Alternate sequence	S,M
ARESALL OPTION ARESALL	OPTION ARESALL	NO	ARESALL	System storage above 16MB virtual	S,M,C
OPTION ARESINV	OPTION ARESINV	NO	ARESINV	Storage above 16MB virtual for invoking program	S,M,C
DEBUG NOASSIST	DEBUG NOASSIST	DEBUG NOASSIST	NO	Bypass Sorting Instructions	S
AVGRLEN OPTION AVGRLEN	OPTION AVGRLEN	NO	NO	Average record length	S
BSAM DEBUG BSAM	DEBUG BSAM	DEBUG BSAM	NO	Force BSAM	S,M,C
DEBUG CFW|NOCFW	DEBUG CFW|NOCFW	DEBUG CFW|NOCFW	CFW	Cache fast write	S
OPTION CHALT|NOCHALT	OPTION CHALT|NOCHALT	NO	CHALT	CH field sequence	S,M
OPTION CHECK|NOCHECK	OPTION CHECK|NOCHECK	NO	CHECK	Record count check	S,M,C
CINV|NOCINV OPTION CINV|NOCINV	OPTION CINV|NOCINV	NO	CINV	Control interval access	S,M,C
COBEXIT OPTION COBEXIT	OPTION COBEXIT	NO	COBEXIT	COBOL library	S,M,C
COLLKEY OPTION COLLKEY	COLLKEY	OPTION COLLKEY	COLLKEY	Collation Version for Unicode Data	S,M,C
INCLUDE|OMIT COND|FORMAT	INCLUDE|OMIT COND|FORMAT	INCLUDE|OMIT COND| FORMAT	NO	Include|Omit fields	S,M,C
OPTION COPY SORT|MERGE FIELDS	OPTION COPY SORT|MERGE FIELDS	SORT|MERGE FIELDS	NO	Copy records	C
DEBUG CTRx	DEBUG CTRx	DEBUG CTRx	NO	ABEND record count	S,M
NO	NO	NO	day	Time-of-day for activation	S,M,C
NO	NO	NO	DIAGSIM	Simulate SORTDIAG DD statement	S,M,C
DSA OPTION DSA	OPTION DSA	NO	DSA	Dynamic storage adjustment limit	S
DSPSIZE OPTION DSPSIZE	OPTION DSPSIZE	NO	DSPSIZE	dataspace sorting	S
DYNALLOC OPTION DYNALLOC SORT DYNALLOC	OPTION DYNALLOC SORT DYNALLOC	SORT DYNALLOC	DYNALOC[1]	Dynamic SORTWKs	S
DYNALLOC OPTION DYNALLOC|USEWKDD SORT DYNALLOC	OPTION DYNALLOC SORT DYNALLOC	SORT DYNALLOC	DYNAUTO	Automatic DYNALLOC	S
DYNAPCT OPTION DYNAPCT	OPTION DYNAPCT	NO	DYNAPCT	Additional work data sets	S
DYNSPC OPTION DYNSPC	OPTION DYNSPC	NO	DYNSPC	Dynamic allocation default space	S
EFS OPTION EFS	NO[2]	NO	EFS	EFS program specified	S,M,C
NO	NO	NO	ENABLE	Enable Time-of-Day modules	S,M,C

Table 104. 24-Bit List DFSORT Option Specification/Override. Options are arranged alphabetically on the Installation column. If "NO" is specified in the Installation column, move to the next column to the left and so on. *(continued)*

Specified with DFSPARM	Specified with SORTCNTL	Specified with 24-Bit List	Installation (INV, TSOINV or TDx)	Description of Option	Function
EQUALS|NOEQUALS OPTION EQUALS|NOEQUALS SORT|MERGE EQUALS| NOEQUALS	OPTION EQUALS|NOEQUALS SORT|MERGE EQUALS| NOEQUALS	SORT|MERGE EQUALS| NOEQUALS	EQUALS	Equal record order	S,M
DEBUG EQUCOUNT	DEBUG EQUCOUNT	DEBUG EQUCOUNT	NO	Equal key count message	S
ABEND|NOABEND DEBUG ABEND|NOABEND	DEBUG ABEND|NOABEND	DEBUG ABEND|NOABEND	ERET	Error action	S,M,C
DEBUG ESTAE|NOESTAE	DEBUG ESTAE|NOESTAE	DEBUG ESTAE|NOESTAE	ESTAE	ESTAE routine	S,M,C
OPTION EXITCK	OPTION EXITCK	NO	EXITCK	E15/E35 return code checking	S,M,C
NO	NO	NO	EXPMAX	Available expanded storage limit for all DFSORT Hiperspaces	S
NO	NO	NO	EXPOLD	Old expanded storage limit for all DFSORT Hiperspaces	S
NO	NO	NO	EXPRES	Available expanded storage reserved for non-Hipersorting use	S
E15=COB MODS E15[3]| HILEVEL=YES	MODS E15[3]| HILEVEL=YES	Offset 18 entry[3] MODS E15[3]| HILEVEL=YES	NO	User exit E15	S,C
NO	NO	Offset 18 entry	NO	User exit E32	M
E35=COB MODS E35[3]|HILEVEL=YES	MODS E35[3]|HILEVEL=YES	Offset 22 entry[3] MODS E35[3]| HILEVEL=YES	NO	User exit E35	S,M,C
MODS Exx	MODS Exx	MODS Exx	NO	User Exit Exx (xx=11,16-19, 31,37-39, and 61)	S,M,C[4]
INREC parameters	INREC parameters	INREC parameters	NO	INREC reformatting	S,M,C
JOINKEYS parameters	JOINKEYS parameters	JOINKEYS parameters	NO	JOINKEYS processing	S,C
JOIN parameters	JOIN parameters	JOIN parameters	NO	JOIN options	S,C
OUTREC parameters	OUTREC parameters	OUTREC parameters	NO	OUTREC reformatting	S,M,C
REFORMAT parameters	REFORMAT parameters	REFORMAT parameters	NO	REFORMAT fields	S,C
SORT|MERGE FIELDS|FORMAT	SORT|MERGE FIELDS|FORMAT	SORT|MERGE FIELDS| FORMAT	NO	Control fields	S,M,C
SUM FIELDS|FORMAT	SUM FIELDS|FORMAT	SUM FIELDS|FORMAT	NO	Sum fields	S,M
MERGE FILES	MERGE FILES	X'04' entry MERGE FILES	NO	Merge input files	M
FILSZ OPTION FILSZ|SIZE SORT|MERGE FILSZ|SIZE	OPTION FILSZ|SIZE SORT|MERGE FILSZ|SIZE	SORT|MERGE FILSZ|SIZE	FSZEST	File size	S,M
NO	NO	NO	GENER	IEBGENER name	C
NO	NO	NO	GNPAD	ICEGENER LRECL padding action	C
NO	NO	NO	GNTRUNC	ICEGENER LRECL truncation action	C
HIPRMAX OPTION HIPRMAX	OPTION HIPRMAX	NO	HIPRMAX	Hipersorting	S
NO	NO	NO	IDRCPCT	IDRC compaction	S
NO	NO	NO	IEXIT	ICEIEXIT	S,M,C
OPTION CKPT[5] SORT|MERGE CKPT[5]	OPTION CKPT[5] SORT|MERGE CKPT[5]	SORT|MERGE CKPT[5]	IGNCKPT	Checkpoints	S

Table 104. 24-Bit List DFSORT Option Specification/Override. Options are arranged alphabetically on the Installation column. If "NO" is specified in the Installation column, move to the next column to the left and so on. *(continued)*

Specified with DFSPARM	Specified with SORTCNTL	Specified with 24-Bit List	Installation (INV, TSOINV or TDx)	Description of Option	Function
NO	NO	NO	IOMAXBF	Maximum SORTIN/ SORTOUT data set buffer space	S,M,C
RECORD LENGTH	RECORD LENGTH	RECORD LENGTH	NO	Record lengths	S,M,C
LIST|NOLIST OPTION LIST|NOLIST	NO[2]	NO	LIST	Print DFSORT control statements[6]	S,M,C
LISTX|NOLISTX OPTION LISTX|NOLISTX	NO[2]	NO	LISTX	Print control statements returned by an EFS program[6]	S,M,C
LOCALE OPTION LOCALE	NO[2]	NO	LOCALE	Locale processing	S,M,C
NO	NO	NO	MAXLIM	Maximum storage below 16MB virtual[7]	S,M,C
OPTION MERGEIN	NO[2]	NO	NO	Alternate MERGE ddnames	M
NO	NO	NO	MINLIM	Minimum storage	S,M,C
MOSIZE OPTION MOSIZE	OPTION MOSIZE	NO	MOSIZE	Memory object sorting	S
MOWRK|NOMOWRK OPTION MOWRK|NOMOWRK	OPTION MOWRK|NOMOWRK	NO	MOWRK	Memory objects as work storage	S
MSGDDN OPTION MSGDDN	NO[2]	X'03' entry	MSGDDN	Alternate message ddname	S,M,C
NO	NO	NO	MSGCON	Write messages on master console	S,M,C
MSGPRT OPTION MSGPRT	NO[2]	X'FF' entry	MSGPRT	Print messages	S,M,C
OPTION NOBLKSET	OPTION NOBLKSET	NO	NO	Bypass Blockset	S,M
NO	NO	NO	NOMSGDD	Action when message data set missing	S,M,C
NULLOUT OPTION NULLOUT	OPTION NULLOUT	NO	NULLOUT	Action when no records for SORTOUT	S,M,C
ODMAXBF OPTION ODMAXBF	OPTION ODMAXBF	NO	ODMAXBF	Maximum OUTFIL data set buffer space	S,M,C
OUTFIL[10]	OUTFIL[10]	OUTFIL[10]	NO	OUTFIL processing	S,M,C
OUTREL|NOOUTREL OPTION NOOUTREL	OPTION NOOUTREL	NO	OUTREL	Release output data set space	S,M,C
OPTION NOOUTSEC	OPTION NOOUTSEC	NO	OUTSEC	Output data set secondary allocation	S,M,C
NO	NO	NO	OVERRGN	Storage over REGION	S,M,C
OVFLO OPTION OVFLO	OPTION OVFLO	NO	OVFLO	Summary fields overflow action	S,M
PAD OPTION PAD	OPTION PAD	NO	PAD	DFSORT LRECL padding action	S,M,C
NO	NO	NO	PARMDDN	Alternate ddname for DFSPARM	S,M,C
RESALL OPTION RESALL	OPTION RESALL	NO	RESALL	System reserved storage[7]	S,M,C
RESET|NORESET OPTION RESET|NORESET	OPTION RESET|NORESET	NO	RESET	NEW or MOD VSAM output	S,M,C

Table 104. 24-Bit List DFSORT Option Specification/Override. Options are arranged alphabetically on the Installation column. If "NO" is specified in the Installation column, move to the next column to the left and so on. *(continued)*

Specified with DFSPARM	Specified with SORTCNTL	Specified with 24-Bit List	Installation (INV, TSOINV or TDx)	Description of Option	Function
OPTION RESINV	OPTION RESINV	X'01' entry	RESINV	Program reserved storage[7]	S,M,C
SDB OPTION SDB	OPTION SDB	NO	SDB	System-determined output data set block size	S,M,C
NO	NO	NO	SDBMSG	System-determined block size for message and list data sets	S,M,C
SIZE OPTION MAINSIZE	OPTION MAINSIZE	X'00' entry	SIZE	Storage	S,M,C
SKIPREC OPTION SKIPREC SORT|MERGE SKIPREC	OPTION SKIPREC SORT|MERGE SKIPREC	SORT|MERGE SKIPREC	NO	Skip records	S,C
OPTION SMF	NO	NO	SMF	SMF records	S,M,C
SOLRF|NOSOLRF OPTION SOLRF|NOSOLRF	OPTION SOLRF|NOSOLRF	NO	SOLRF	SORTOUT length	S,M,C
OPTION SORTDD	NO[2]	Prefix entry	NO	ddname prefix	S,M,C
OPTION SORTIN[8]	NO[2]	NO	NO	Alternate SORTIN ddname	S,C
NO	NO	NO	SORTLIB	Conventional modules library	S,M
OPTION SORTOUT[9]	NO[2]	NO	NO	Alternate SORTOUT ddname	S,M,C
SPANINC OPTION SPANINC	OPTION SPANINC	NO	SPANINC	Incomplete spanned records action	S,M,C
STOPAFT OPTION STOPAFT SORT|MERGE STOPAFT	OPTION STOPAFT SORT|MERGE STOPAFT	SORT|MERGE STOPAFT	NO	Input limit	S,C
NO	NO	NO	SVC	DFSORT SVC information	S,M,C
SZERO|NOSZERO OPTION SZERO|NOSZERO	OPTION SZERO|NOSZERO	NO	SZERO	Signed or unsigned zero	S,M,C
NO	NO	NO	TEXIT	ICETEXIT	S,M,C
NO	NO	NO	TMAXLIM	Maximum storage above and below 16MB virtual[7]	S,M,C
NO	NO	NO	TUNE	Optimize central storage or disk work space	S
TRUNC OPTION TRUNC	OPTION TRUNC	NO	TRUNC	DFSORT LRECL truncation action	S,M,C
RECORD TYPE	RECORD TYPE	RECORD TYPE	NO	Record format	S,M,C
VERIFY|NOVERIFY OPTION VERIFY|NOVERIFY	OPTION VERIFY|NOVERIFY	NO	VERIFY	Sequence check	S,M
NO	NO	NO	VIO	SORTWK virtual I/O	S
VLLONG|NOVLLONG OPTION VLLONG|NOVLLONG	OPTION VLLONG|NOVLLONG	NO	VLLONG	Truncate long output records	S,M,C
VLSCMP|NOVLSCMP OPTION VLSCMP|NOVLSCMP	OPTION VLSCMP|NOVLSCMP	NO	VLSCMP	Pad short compare fields	S,M,C
VLSHRT|NOVLSHRT OPTION VLSHRT|NOVLSHRT	OPTION VLSHRT|NOVLSHRT	NO	VLSHRT	Action for short control or compare field	S,M,C
NO	NO	NO	VSAMBSP	VSAM buffer space	S
VSAMEMT|NVSAMEMT OPTION VSAMEMT| NVSAMEMT	OPTION VSAMEMT|NVSAMEMT	NO	VSAMEMT	Empty VSAM input	S,M,C

Table 104. 24-Bit List DFSORT Option Specification/Override. Options are arranged alphabetically on the Installation column. If "NO" is specified in the Installation column, move to the next column to the left and so on. *(continued)*

Specified with DFSPARM	Specified with SORTCNTL	Specified with 24-Bit List	Installation (INV, TSOINV or TDx)	Description of Option	Function
VSAMIO|NOVSAMIO OPTION VSAMIO|NOVSAMIO	OPTION VSAMIO|NOVSAMIO	NO	VSAMIO	Same VSAM input and output	S
WRKREL|NOWRKREL OPTION WRKREL|NOWRKREL	OPTION WRKREL|NOWRKREL	NO	WRKREL	Release SORTWK space	S
WRKSEC|NOWRKSEC OPTION WRKSEC|NOWRKSEC	OPTION WRKSEC| NOWRKSEC	NO	WRKSEC	SORTWK secondary allocation	S
Y2PAST OPTION Y2PAST SORT|MERGE Y2PAST	OPTION Y2PAST SORT|MERGE Y2PAST	SORT|MERGE Y2PAST	Y2PAST	Set century window	S,M,C
ZDPRINT|NZDPRINT OPTION ZDPRINT|NZDPRINT	OPTION ZDPRINT|NZDPRINT	NO	ZDPRINT	ZD SUM results	S,M

Notes to 24-bit list table

1

Does not request dynamic allocation; only supplies defaults.

2

Not used in SORTCNTL.

3

DFSORT terminates if the exit is specified via the parameter list entry and the user exit is specified in a MODS statement.

4

All functions do not apply to all user exits. See Table 66 on page 470 and Table 67 on page 470 for applicable user exits.

5

Not used if Blockset is selected and IGNCKPT=YES was specified.

6

Not used if MSGPRT=NONE or MSGPRT=CRITICAL is in effect; in this case control statements are not printed.

7

Not used unless MAINSIZE=MAX is in effect.

8

Overrides SORTDD for the sort input ddname.

9

Overrides SORTDD for the sort output ddname.

10

Override is at the ddname level.

Appendix C. Data format descriptions

DFSORT data formats

DFSORT supports a large number of data formats as described below.

DFSORT data formatsFormat	Description
CH	(character EBCDIC, unsigned). Each character is represented by its 8-bit EBCDIC code. Example: AB7 becomes C1 C2 F7 Hexadecimal 11000001 11000010 11110111 Binary **Note:** 1. If CHALT is in effect, a format CH field collates according to the ALTSEQ (alternate collating sequence) table in effect. AQ format can be used for the same purpose. 2. If locale processing is in effect, a format CH field collates according to the collating rules of the active locale.
UTF8	Unicode Transformation Format, an 8-bit encoding form designed for ease of use with existing ASCII-based systems. UTF-8 can encode any of the Unicode characters. A UTF-8 character is 1, 2, 3, or 4 bytes in length. A UTF-8 data string can contain any combination of SBCS and MBCS data, including supplementary characters. **Character** A **Unicode** U+0041 **UTF-8 (Half bytes in hexadecimal)** 01000001 4 1 => X'41' **Character** § **Unicode** U+00A7 **UTF-8 (Half bytes in hexadecimal)** 11000010 10100111 C 2 A 7 => X'C2A7' **Character** ᇖ **Unicode** U+1156 **UTF-8 (Half bytes in hexadecimal)** 11100001 10000101 10010110 E 1 8 5 9 6 => x'E18596' **Note:** 1. Refer to the Unicode Character Code Charts (www.unicode.org/charts) for all the code point representations. 2. When sorting/merging, a format UTF8 field collates according to the collating version of the active COLLKEY. UCA600 is the default collation version.

DFSORT data formatsFormat	Description
UTF16	Unicode Transformation Format, a 16-bit encoding form designed to provide code values for over a million characters. UTF-16 can encode any of the Unicode characters. In UTF-16 encoding, characters are 2 bytes in length, except for supplementary characters, which take two 2 byte string units per character. **Character** A **Unicode** U+0041 **UTF16 (hexadecimal)** 00000000 01000001 0 0 4 1 => X'0041' **Character** § **Unicode** U+00A7 **UTF16 (hexadecimal)** 00000000 10100111 0 0 A 7 => X'00A7' **Character** ᄖ **Unicode** U+1156 **UTF16 (hexadecimal)** 00010001 01010110 1 1 5 6 => x'1156' **Note:** 1. Refer to the Unicode Character Code Charts (www.unicode.org/charts) for all the code point representations. 2. When sorting/merging, a format UTF16 field collates according to the collating version of the active COLLKEY. UCA600 is the default collation version.

DFSORT data formatsFormat	Description
UTF32	Unicode Transformation Format, a 32-bit encoding form with a fixed-length encoding, in contrast to all other Unicode transformation formats, which are variable-length encodings. Each 32-bit value in UTF32 represents one Unicode code point and is exactly equal to that code point's numerical value. **Character** A **Unicode** U+0041 **UTF32 (hexadecimal)** 00000000 00000000 01000001 0 0 0 0 4 1 => X'00000041' **Character** § **Unicode** U+00A7 **UTF32 (hexadecimal)** 00000000 00000000 10100111 0 0 0 0 A 7 => X'00A7' **Character** 𑅖 **Unicode** U+1156 **UTF32 (hexadecimal)** 00000000 00000000 00010001 01010110 0 0 0 0 1 1 5 6 => X'1156' **Note:** 1. Refer to the Unicode Character Code Charts (www.unicode.org/charts) for all the code point representations. 2. When sorting/merging, a format UTF32 field collates according to the collating version of the active COLLKEY. UCA600 is the default collation version.
ZD	(zoned decimal, signed). Each digit of the decimal number is converted into its 8-bit EBCDIC representation. The sign indicator replaces the first four bits of the low order byte of the number. Example: -247 becomes 2 4 - 7 Decimal F2 F4 D7 Hexadecimal 11110010 11110100 11010111 Binary The number +247 becomes F2 F4 C7 11110010 11110100 11000111 **Note:** 1. The following are treated as positive sign indicators: F, E, C, A, 8, 6, 4, 2, 0. 2. The following are treated as negative sign indicators: D, B, 9, 7, 5, 3, 1. 3. For SUM processing, 0 through 9 for the sign or A through F for a digit results in a data exception (0C7 ABEND). For example, a ZD value such as 3.5 (X'F34BF5') results in an 0C7 because B is treated as an invalid digit. ICETOOL's DISPLAY or VERIFY operator can be used to identify ZD values with invalid digits. ICETOOL's VERIFY operator can be used to identify ZD values with invalid signs. 4. The first four bits of the last digit is the sign indicator. The first four bits of each other digit is ignored. Thus the EBCDIC strings '0025' and ' 25' are both treated as 25 because a leading blank (X'40') is equivalent to a 0 digit (X'F0').

DFSORT data formatsFormat	Description
PD	(packed decimal, signed). Each digit of the decimal number is converted into its 4-bit binary equivalent. The sign indicator is put into the rightmost four bits of the number. Example: -247 becomes 2 4 7- Decimal 2 4 7D Hexadecimal 00100100 01111101 Binary The number +247 becomes 247C in hexadecimal. **Note:** 1. The following are treated as positive sign indicators: F, E, C, A, 8, 6, 4, 2, 0. 2. The following are treated as negative sign indicators: D, B, 9, 7, 5, 3, 1. 3. For SUM processing, 0 through 9 for the sign or A through F for a digit results in a data exception (0C7 ABEND). For example, a PD value such as X' 0123BF' results in an 0C7 because B is treated as an invalid digit. ICETOOL's DISPLAY or VERIFY operator can be used to identify PD values with invalid digits. ICETOOL's VERIFY operator can be used to identify PD values with invalid signs.
PD0	(packed decimal, with sign and first digit ignored) The PD0 format can be represented as follows: `xddd...ds` x is hexadecimal 0-F and is ignored. d is hexadecimal 0-9 and represents a decimal digit. s is hexadecimal 0-F and is ignored. PD0 can be used for parts of PD fields. For example, in the PD field P'mmddyy' (hexadecimal 0mmddyyC), PD0 can be used separately for 0mmd (mm), mddy (dd) and dyyC (yy).

FI	(fixed-point, signed). The complete number is represented by its binary equivalent with the sign indicator placed in the most significant bit position. 0 for + or 1 for -. Negative numbers are in 2's complement form.
	
	`Example: +247 becomes in halfword form`
	`00F7 Hexadecimal`
	`0000000011110111 Binary`
	`-247 becomes in halfword form`
	`FF09 Hexadecimal`
	`1111111100001001 Binary`

BI	(binary unsigned). Any bit pattern.

FL	(hexadecimal floating-point, signed). The specified number is in the two-part format of characteristic and fraction with the sign indicator in bit position 0.
	
	`Example: +247 becomes`
	`42F70000... Hexadecimal`
	`0 1000010 111101110000000..... Binary`
	`+ chara. Fraction`
	`-247 is identical, except that the sign bit is changed to 1.`
	`C2F70000... Hexadecimal`
	`1 1000010 111101110000000..... Binary`
	`- chara. Fraction`

AQ	(character EBCDIC, with alternate collating sequence, unsigned). This is similar to format CH, but the characters collate according to the ALTSEQ (alternate collating sequence) table in effect.
AC	(character EBCDIC, with ASCII collating sequence, unsigned). This is similar to format CH, but the characters collate according to the ASCII collating sequence.
D1	(EFS type). User-defined data type (requires an EFS program)
D2	(EFS type). User-defined data type (requires an EFS program)

DFSORT data formatsFormat	Description
CSF or FS	(signed numeric with optional leading floating sign). The floating sign format can be represented as follows: <s>d . . .d s is an optional sign immediately to the left of the digits d . . .d. If s is a -, the number is treated as negative, otherwise it is treated as positive. Thus, - must be used for a minus sign, but any other character (for example, + or blank) can be used for a plus sign. The first non-decimal digit (that is, not 0-9) going from right to left is treated as the sign and anything to the left of the sign is ignored. Examples: Value: Treated as: 34 +34 +34 +34 00034 +34 -003 -3 --1234 -1234 1234 +1234 +01234 +1234 0 +0 The types of data handled by the CSF or FS format encompass those produced by several different FORTRAN, PL/I and COBOL formats, such as those shown below (using a width of 4 for purposes of illustration): * FORTRAN: I4 ; G4.0 ; SP,I4 ; SP,I4.3 ; S,I4.3 * PL/I: F(4) ; P'S999' ; P'SSS9' ; P'---9' * COBOL: PIC ++9 ; PIC +999 ; PIC ++++ ; PIC ---9 ; PIC ---- ; PIC ZZZZ
UFF	(unsigned free form numeric). This format extracts decimal digits (0-9) from right to left anywhere in the field to form a positive number. Any combination of characters is valid, but characters other than 0-9 are ignored. Examples: Value: Treated as: $58,272,300.10 +5827230010 $58,272,300.1 +582723001 $58,272,300 +58272300 12-31-2004 +12312004 (402)-125-3721XXX +4021253721 G1*** 52 $ 21 R +15221 000128637.240 +128637240 +400.52 +40052 +400.1 +4001 173/821/9072/@3 +17382190723 ABC +0
SFF	(signed free form numeric). This format extracts decimal digits (0-9) from right to left anywhere in the field to form a positive or negative number. If - or) is found anywhere in the field, the number is treated as negative, otherwise it is treated as positive. Any combination of characters is valid, but characters other than 0-9, - and) are ignored. Examples: Value: Treated as: 358,272,300.10 +35827230010 358,272,300.1 +3582723001 -358,272,300 -358272300 (82,316.90) -8231690 12-31-2004 -12312004 G1*** 52 $ 21 R +15221 G1***) 52 $ 21 R -15221 000128637.240 +128637240 400.52- -40052 ($400.5) -4005 173/821/9072/@3 +17382190723 X,Y,Z +0

DFSORT data formatsFormat	Description
CSL or LS	(signed number, leading separate sign). This format refers to decimal data as punched into cards, and then assembled into EBCDIC code. Example: +247 punched in a card becomes + 2 4 7 Punched numeric data 4E F2 F4 F7 Hexadecimal 01001110 11110010 11110100 11110111 Binary EBCDIC code -247 becomes - 2 4 7 Punched numeric data 60 F2 F4 F7 Hexadecimal 01100000 11110010 11110100 11110111 Binary EBCDIC code **Note:** A value with '-' as the leading sign character is treated as a negative value. A value with any leading sign character other than '-' (for example, '+' (plus) or blank) is treated as a positive value.
CST or TS	(signed numeric, trailing separate sign). This has the same representation as the CSL format, except that the sign indicator is punched after the number. Example: 247+ punched on the card becomes F2 F4 F7 4E Hexadecimal **Note:** A value with '-' as the trailing sign character is treated as a negative value. A value with any trailing sign character other than '-' (for example, '+' (plus) or blank) is treated as a positive value.
CLO[1] or OL[1]	(signed numeric, leading overpunch sign). This format again refers to decimal data punched into cards and then assembled into EBCDIC code. The sign indicator is, however, overpunched with the first decimal digit of the number. Example: +247 with + overpunched on 2 becomes +2 4 7 Punched numeric data C2 F4 F7 Hexadecimal 11000010 11110100 11110111 Binary EBCDIC code Similarly -247 becomes D2 F4 F7
CTO or OT	(signed numeric, trailing overpunch sign). This format has the same representation as for the CLO format, except that the sign indicator is overpunched on the last decimal digit of the number. Example: +247 with + overpunched on 7 becomes F2 F4 C7 hexadecimal
ASL	(signed numeric, ASCII, leading separate sign). Similar to the CSL format but with decimal data assembled into ASCII code. Example: +247 punched into card becomes + 2 4 7 Punched numeric data 2B 32 34 37 Hexadecimal 0101011 00110010 00110100 00110111 Binary ASCII code Similarly -247 becomes 2D 32 34 37 hexadecimal **Note:** A value with '-' as the leading sign character is treated as a negative value. A value with any leading sign character other than '-' (for example, '+' (plus) or blank) is treated as a positive value.
AST	(signed numeric, ASCII, trailing separate sign). This gives the same bit representation as the ASL format, except that the sign is punched after the number. Example: 247+ becomes 32 34 37 2B hexadecimal **Note:** A value with '-' as the trailing sign character is treated as a negative value. A value with any trailing sign character other than '-' (for example, '+' (plus) or blank) is treated as a positive value.

DFSORT data formatsFormat	Description

AUF	(ASCII unsigned free form numeric). This format extracts decimal digits 0-9 specified in ASCII code (X'30'-X'39') from right to left anywhere in the field to form a positive number. Any combination of characters is valid, but characters other than 0-9 are ignored.
	Examples:
	`Value: Treated as:`
	`$58,272,300.10 +5827230010`
	`$58,272,300.1 +582723001`
	`$58,272,300 +58272300`
	`12-31-2004 +12312004`
	`(402)-125-3721XXX +4021253721`
	`G1*** 52 $ 21 R +15221`
	`000128637.240 +128637240`
	`+400.52 +40052`
	`+400.1 +4001`
	`173/821/9072/@3 +17382190723`
	`ABC +0`

ASF	(ASCII signed free form numeric). This format extracts decimal digits 0-9 in ASCII code (X'30'-X'39') from right to left anywhere in the field to form a positive or negative number. If '-' or ')' in ASCII code (X'2D' or X'29') is found anywhere in the field, the number is treated as negative, otherwise it is treated as positive. Any combination of characters is valid, but characters other than 0-9, '-' and ')' are ignored.
	Examples:
	`Value: Treated as:`
	`358,272,300.10 +35827230010`
	`358,272,300.1 +3582723001`
	`-358,272,300 -358272300`
	`(82,316.90) -8231690`
	`12-31-2004 -12312004`
	`G1*** 52 $ 21 R +15221`
	`G1*** ) 52 $ 21 R -15221`
	`000128637.240 +128637240`
	`400.52- -40052`
	`($400.5) -4005`
	`173/821/9072/@3 +17382190723`
	`X,Y,Z +0`

In the date formats below, unless specified otherwise, yy represents the two-digit year (00-99), ddd represents the day of the year (001-366), ccyy represents the four-digit year, mm represents the month (01-12), dd represents the day (01-31), x represents a decimal digit (0-9), s is hexadecimal 0-F and is ignored, and c represents the century indicator (c=0 is transformed to 19, c=1 is transformed to 20 and c>1 is transformed to 21).

In the date formats below, unless specified otherwise, yy represents the two-digit year (00-99), ddd represents the day of the year (001-366), ccyy represents the four-digit year, mm represents the month (01-12), dd represents the day (01-31), x represents a decimal digit (0-9), s is hexadecimal 0-F and is ignored, and c represents the century indicator (c=0 is transformed to 19, c=1 is transformed to 20 and c>1 is transformed to 21).Format	Description
Y2T, Y2W, Y4T, Y4W	(character or zoned decimal date format with special indicators). The date field can be represented as follows: 3,Y2T: C'yyx' or Z'yyx' 4,Y2T: C'yyxx' or Z'yyxx' 5,Y2T: C'yyddd' or Z'yyddd' 6,Y2T: C'yymmdd' or Z'yymmdd' 7,Y4T: C'ccyyddd' or Z'ccyyddd' 8,Y4T: C'ccyymmdd' or Z'ccyymmdd' 3,Y2W: C'xyy' or Z'xyy' 4,Y2W: C'xxyy' or Z'xxyy' 5,Y2W: C'dddyy' or Z'dddyy' 6,Y2W: C'mmddyy' or Z'mmddyy' 7,Y4W: C'dddccyy' or Z'dddccyy' 8,Y4W: C'mmddccyy' or Z'mmddccyy' The special indicators are X'00...00' (BI zeros), X'40...40' (blanks), C'0...0' (CH zeros), Z'0...0' (ZD zeros), C'9...9' (CH nines), Z'9...9' (ZD nines) and X'FF...FF' (BI ones).
Y2U, Y2V, Y2X, Y2Y, Y4U, Y4V, Y4X, Y4Y	(packed decimal date format with special indicators). The date field can be represented as follows: 2,Y2U: P'yyx' (X'yyxs') 3,Y2V: P'yyxx' (X'0yyxxs') 3,Y2U: P'yyddd' (X'yyddds') 4,Y2V: P'yymmdd' (X'0yymmdds') 4,Y4U: P'ccyyddd' (X'ccyyddds') 5,Y4V: P'ccyymmdd' (X'0ccyymmdds') 2,Y2X: P'xyy' (X'xyys') 3,Y2Y: P'xxyy' (X'0xxyys') 3,Y2X: P'dddyy' (X'dddyys') 4,Y2Y: P'mmddyy' (X'0mmddyys') 4,Y4X: P'dddccyy' (X'dddccyys') 5,Y4Y: P'mmddccyy' (X'0mmddccyys') The special indicators are P'0...0' (PD zeros) and P'9...9' (PD nines).

In the date formats below, unless specified otherwise, yy represents the two-digit year (00-99), ddd represents the day of the year (001-366), ccyy represents the four-digit year, mm represents the month (01-12), dd represents the day (01-31), x represents a decimal digit (0-9), s is hexadecimal 0-F and is ignored, and c represents the century indicator (c=0 is transformed to 19, c=1 is transformed to 20 and c>1 is transformed to 21).Format	Description
Y2C or Y2Z	(two-digit, two-byte character or zoned-decimal year data). The two-digit year data can be represented as follows: xyxy y is hexadecimal 0-9 and represents a year digit. x is hexadecimal 0-F and is ignored. Thus, 96 might be represented as hexadecimal F9F6 (character 96) or as hexadecimal F9C6 or 0906 (zoned decimal 96).
Y2P	(two-digit, two-byte packed-decimal year data). The two-digit year data can be represented as follows: xyyx y is hexadecimal 0-9 and represents a year digit. x is hexadecimal 0-F and is ignored. Thus, 96 might be represented as hexadecimal 096F or 896C (packed decimal 96).
Y2D	(two-digit, one-byte decimal year data). The two-digit year data can represented as follows: yy y is hexadecimal 0-9 and represents a year digit. Thus, 96 would be represented as hexadecimal 96 (decimal 96).
Y2S	(two-digit, two-byte character or zoned-decimal year data with special indicators). The two-digit year data can represented as follows: xyxy y is hexadecimal 0-9 and represents a year digit. x is hexadecimal 0-F and is ignored. Thus, 96 might be represented as hexadecimal F9F6 (character 96) or as hexadecimal F9C6 or 0906 (zoned decimal 96). The special indicators can be represented as follows: qxzx qx is hexadecimal 00, 40 or FF. zx is hexadecimal 00-FF (although typically 00, 40 and FF). Thus, special indicators might be hexadecimal 0000, 0005, 4040, FFFF, FF85 and so on.

In the date formats below, unless specified otherwise, yy represents the two-digit year (00-99), ddd represents the day of the year (001-366), ccyy represents the four-digit year, mm represents the month (01-12), dd represents the day (01-31), x represents a decimal digit (0-9), s is hexadecimal 0-F and is ignored, and c represents the century indicator (c=0 is transformed to 19, c=1 is transformed to 20 and c>1 is transformed to 21).Format	**Description**

Y2B	(two-digit, one-byte binary year data). The binary year data can be represented as follows:
	hh
	hh is the hexadecimal equivalent of a decimal yy value as follows:
	`Binary Values  Decimal Values  yy`
	`X'00'-X'63'  00-99  00-99`
	`X'64'-X'C7'  100-199  00-99`
	`X'C8'-X'FF'  200-255  00-55`
	Thus, 96 might be represented as hexadecimal 60 (decimal 96) or C4 (decimal 196).

DT1	(SMF date interpreted as Z'yyyymmdd'). A 4-byte SMF date value in the form P'cyyddd' (X'0cyydddF') is converted to a Z'yyyymmdd' value.
DT2	(SMF date interpreted as Z'yyyymm'). A 4-byte SMF date value in the form P'cyyddd' (X'0cyydddF') is converted to a Z'yyyymm' value.
DT3	(SMF date interpreted as Z'yyyyddd'). A 4-byte SMF date value in the form P'cyyddd' (X'0cyydddF') is converted to a Z'yyyyddd' value.
DC1	(TOD date interpreted as Z'yyyymmdd'). The 8 bytes of an input clock value, in the basic time-of-day (TOD) format, is converted to a Z'yyyymmdd' value. The STCKCONV macro is used to do the conversion.
DC2	(TOD date interpreted as Z'yyyymm'). The 8 bytes of an input clock value, in the basic time-of-day (TOD) format, is converted to a Z'yyyymm' value. The STCKCONV macro is used to do the conversion.
DC3	(TOD date interpreted as Z'yyyyddd'). The 8 bytes of an input clock value, in the basic time-of-day (TOD) format, is converted to a Z'yyyyddd' value. The STCKCONV macro is used to do the conversion.
DE1	(ETOD date interpreted as Z'yyyymmdd'). The first 8 bytes of an input clock value, in the extended time-of-day (ETOD) format, is converted to a Z'yyyymmdd' value. The STCKCONV macro is used to do the conversion.
DE2	(ETOD date interpreted as Z'yyyymm'). The first 8 bytes of an input clock value, in the extended time-of-day (ETOD) format is converted to a Z'yyyymm' value. The STCKCONV macro is used to do the conversion.
DE3	(ETOD date interpreted as Z'yyyyddd'). The first 8 bytes of an input clock value, in the extended time-of-day (ETOD) format is converted to a Z'yyyyddd' value. The STCKCONV macro is used to do the conversion.

In the time formats below, unless specified otherwise, hh represents the hour (00-23), mm represents the minutes (00-59), ss represents the seconds (00-59), and xx represents hundredths of a second (00-99).

In the time formats below, unless specified otherwise, hh represents the hour (00-23), mm represents the minutes (00-59), ss represents the seconds (00-59), and xx represents hundredths of a second (00-99).Format	Description
TM1	(SMF time interpreted as Z'hhmmss'). A 4-byte binary SMF time value in hundredths of a second is converted to a Z'hhmmss' value.
TM2	(SMF time interpreted as Z'hhmm'). A 4-byte binary SMF time value in hundredths of a second is converted to a Z'hhmm' value.
TM3	(SMF time interpreted as Z'hh'). A 4-byte binary SMF time value in hundredths of a second is converted to a Z'hh' value.
TM4	(SMF time interpreted as Z'hhmmssxx'). A 4-byte binary SMF time value in hundredths of a second is converted to a Z'hhmmssxx' value.
TC1	(TOD time interpreted as Z'hhmmss'). The 8 bytes of an input clock value, in the basic time-of-day (TOD) format, is converted to a Z'hhmmss' value. The STCKCONV macro is used to do the conversion.
TC2	(TOD time interpreted as Z'hhmm'). The 8 bytes of an input clock value, in the basic time-of-day (TOD) format, is converted to a Z'hhmm' value. The STCKCONV macro is used to do the conversion.
TC3	(TOD time interpreted as Z'hh'). The 8 bytes of an input clock value, in the basic time-of-day (TOD) format, is converted to a Z'hh' value. The STCKCONV macro is used to do the conversion.
TC4	(TOD time interpreted as Z'hhmmssxx'). The 8 bytes of an input clock value, in the basic time-of-day (TOD) format, is converted to a Z'hhmmssxx' value. The STCKCONV macro is used to do the conversion.
TE1	(ETOD time interpreted as Z'hhmmss'). The first 8 bytes of an input clock value, in the extended time-of-day (ETOD) format, is converted to a Z'hhmmss' value. The STCKCONV macro is used to do the conversion.
TE2	(ETOD time interpreted as Z'hhmm'). The first 8 bytes of an input clock value, in the extended time-of-day (ETOD) format, is converted to a Z'hhmm' value. The STCKCONV macro is used to do the conversion.
TE3	(ETOD time interpreted as Z'hh'). The first 8 bytes of an input clock value, in the extended time-of-day (ETOD) format, is converted to a Z'hh' value. The STCKCONV macro is used to do the conversion.
TE4	(ETOD time interpreted as Z'hhmmssxx'). The first 8 bytes of an input clock value, in the extended time-of-day (ETOD) format, is converted to a Z'hhmmssxx' value. The STCKCONV macro is used to do the conversion.

1

The overpunch sign bit is always 'C' for positive and 'D' for negative.

Where DFSORT formats can be used

The following tables show the statements, operands, and operators allowed with each of the various data formats.

Table 105. Allowed with Frequently Used Data Types

Statement, Operand, or Operator	CH	BI or FI	PD or ZD	FS or CSF	UFF or SFF	DTn, DCn, DEn, TMn, TCn, or TEn
DFSORT statements						
INCLUDE	X	X	X	X	X	
MERGE	X	X	X	X	X	
OMIT	X	X	X	X	X	
SORT	X	X	X	X	X	

Table 105. Allowed with Frequently Used Data Types (continued)

Statement, Operand, or Operator	CH	BI or FI	PD or ZD	FS or CSF	UFF or SFF	DTn, DCn, DEn, TMn, TCn, or TEn
SUM		X	X			
INREC statement operands						
IFTHEN WHEN=(logexp)	X	X	X	X	X	
IFTHEN BEGIN=(logexp)	X	X	X	X	X	
IFTHEN END=(logexp)	X	X	X	X	X	
FIELDS		X	X	X	X	X
BUILD		X	X	X	X	X
OVERLAY		X	X	X	X	X
IFTHEN BUILD		X	X	X	X	X
IFTHEN OVERLAY		X	X	X	X	X
OUTREC statement operands						
IFTHEN WHEN=(logexp)	X	X	X	X	X	
IFTHEN BEGIN=(logexp)	X	X	X	X	X	
IFTHEN END=(logexp)	X	X	X	X	X	
FIELDS		X	X	X	X	X
BUILD		X	X	X	X	X
OVERLAY		X	X	X	X	X
IFTHEN BUILD		X	X	X	X	X
IFTHEN OVERLAY		X	X	X	X	X
OUTFIL statement operands						
INCLUDE	X	X	X	X	X	
OMIT	X	X	X	X	X	
IFTHEN WHEN=(logexp)	X	X	X	X	X	
IFTHEN BEGIN=(logexp)	X	X	X	X	X	
IFTHEN END=(logexp)	X	X	X	X	X	
OUTREC		X	X	X	X	X
BUILD		X	X	X	X	X
OVERLAY		X	X	X	X	X
IFTHEN BUILD		X	X	X	X	X
IFTHEN OVERLAY		X	X	X	X	X
TRAILERx		X	X	X	X	
IFTRAIL TRLID=(logexp)	X	X	X	X	X	
IFTRAIL TRLUPD		X	X	X	X	
ICETOOL operators						
DISPLAY (ON, BREAK)	X	X	X	X	X	X

Table 105. Allowed with Frequently Used Data Types (continued)

Statement, Operand, or Operator	CH	BI or FI	PD or ZD	FS or CSF	UFF or SFF	DTn, DCn, DEn, TMn, TCn, or TEn
OCCUR (ON)	X	X	X	X	X	X
RANGE (ON)		X	X	X	X	
SELECT (ON)	X	X	X	X	X	
SPLICE (ON)	X	X	X	X	X	
STATS (ON)		X	X	X	X	
UNIQUE (ON)		X	X	X	X	
VERIFY (ON)			X			

Table 106. Allowed with Other Data Types

Allowed with Other Data TypesStatement or Operand	AQ	AC	FL	LS or CSL	TS or CST	OL or CLO	OT or CTO	ASL	AST	D1	D2	PD0	Y2x	Y4x
DFSORT statements														
INCLUDE	X	X		X	X	X	X	X	X		X	X	X	
MERGE	X	X	X	X	X	X	X	X	X	X		X	X	
OMIT	X	X		X	X	X	X	X	X		X	X	X	
SORT	X	X	X	X	X	X	X	X	X	X		X	X	
SUM			X											
INREC statement operands														
IFTHEN WHEN=(exp)	X	X		X	X	X	X	X	X			X	X	
IFTHEN BEGIN=(exp)	X	X		X	X	X	X	X	X			X	X	
IFTHEN END=(exp)	X	X		X	X	X	X	X	X			X	X	
FIELDS			X									X	X	X
BUILD			X									X	X	X
OVERLAY			X									X	X	X
IFTHEN BUILD			X									X	X	X
IFTHEN OVERLAY			X									X	X	X
OUTREC statement operands														
IFTHEN WHEN=(exp)	X	X		X	X	X	X	X	X			X	X	
IFTHEN BEGIN=(exp)	X	X		X	X	X	X	X	X			X	X	
IFTHEN END=(exp)	X	X		X	X	X	X	X	X			X	X	
FIELDS			X									X	X	X
BUILD			X									X	X	X
OVERLAY			X									X	X	X
IFTHEN BUILD			X									X	X	X
IFTHEN OVERLAY			X									X	X	X
OUTFIL statement operands														

Table 106. Allowed with Other Data Types (continued)

Allowed with Other Data TypesStatement or Operand	AQ	AC	FL	LS or CSL	TS or CST	OL or CLO	OT or CTO	ASL	AST	D1	D2	PD0	Y2x	Y4x
INCLUDE	X	X		X	X	X	X	X	X			X	X	
OMIT	X	X		X	X	X	X	X	X			X	X	
IFTHEN WHEN=(exp)	X	X		X	X	X	X	X	X			X	X	
IFTHEN BEGIN=(exp)	X	X		X	X	X	X	X	X			X	X	
IFTHEN END=(exp)	X	X		X	X	X	X	X	X			X	X	
IFTRAIL TRLID=(exp)	X	X		X	X	X	X	X	X			X	X	
OUTREC			X									X	X	X
BUILD			X									X	X	X
OVERLAY			X									X	X	X
IFTHEN BUILD			X									X	X	X
IFTHEN OVERLAY			X									X	X	X
ICETOOL operators														
DISPLAY (ON)			X											

Table 107. ***UTF8/UTF16/UTF32***

	UTF8/UTF16/UTF32
DFSORT statements	
INCLUDE	X
MERGE	X
OMIT	X
SORT	X
SUM	
INREC statement operands	
IFTHEN WHEN=(exp)	
IFTHEN BEGIN=(exp)	
IFTHEN END=(exp)	
FIELDS	
BUILD	
OVERLAY	
IFTHEN BUILD	
IFTHEN OVERLAY	
OUTREC statement operands	
IFTHEN WHEN=(exp)	
IFTHEN BEGIN=(exp)	
IFTHEN END=(exp)	

*Table 107. **UTF8/UTF16/UTF32** (continued)*	
	UTF8/UTF16/UTF32
FIELDS	
BUILD	
OVERLAY	
IFTHEN BUILD	
IFTHEN OVERLAY	
OUTFIL statement operands	
INCLUDE	
OMIT	
IFTHEN WHEN=(exp)	
IFTHEN BEGIN=(exp)	
IFTHEN END=(exp)	
IFTRAIL TRLID=(exp)	
OUTREC	
BUILD	
OVERLAY	
IFTHEN BUILD	
IFTHEN OVERLAY	

DFSORT formats for COBOL data types

Both DFSORT and COBOL support a large number of data types. COBOL describes these data types in one way, and DFSORT describes them in another way. If you SORT or MERGE with COBOL, the compiler automatically generates a SORT or MERGE control statement for you with the correct DFSORT descriptions for the COBOL fields you specify. But to take full advantage of DFSORT, you will often want to describe your fields in your own DFSORT control statements (for example, SORT, MERGE, INCLUDE, OMIT, INREC, OUTREC, OUTFIL, SUM) either outside of COBOL or in a DFSPARM data set used with COBOL. The table below will show you what DFSORT length and format to use for the various commonly used COBOL data types.

For example, say you want to separate out records in a very large file into two data sets based on the values in a PIC S9(4) COMP field starting in position 21. In the first data set, you want records with values in the field that are greater than or equal to +5000. In the second data set, you want records with values in the field that are less than -1000. You could use the table below to determine that a PIC S9(4) COMP field is equivalent to a DFSORT field with a length of 2 and a format of FI, allowing you to code your DFSORT statements as follows:

```
  OPTION COPY
  OUTFIL FNAMES=OUT1,INCLUDE=(21,2,FI,GE,+5000)
  OUTFIL FNAMES=OUT2,INCLUDE=(21,2,FI,LT,-1000)
```

Table 108. Equivalent DFSORT formats for various COBOL data types		
Equivalent DFSORT formats for various COBOL data typesCOBOL data type	**DFSORT Length**	**DFSORT Format**
PIC X(n) USAGE DISPLAY	n	CH

Table 108. Equivalent DFSORT formats for various COBOL data types (continued)

Equivalent DFSORT formats for various COBOL data typesCOBOL data type	DFSORT Length	DFSORT Format
GROUP DATA ITEMS with n bytes	n	CH
PIC 9(n) DISPLAY	n	ZD
PIC S9(n) DISPLAY <TRAILING>	n	ZD
PIC S9(n) DISPLAY LEADING	n	CLO
PIC S9(n) DISPLAY SEPARATE <TRAILING>	n+1	CST
PIC S9(n) DISPLAY LEADING SEPARATE	n+1	CSL or FS
PIC 9(n) COMP|BINARY|COMP-4|COMP-5		
n = 1 to 4	2	BI
n = 5 to 9	4	BI
n >= 10	8	BI
PIC S9(n) COMP|BINARY|COMP-4|COMP-5		
n = 1 to 4	2	FI
n = 5 to 9	4	FI
n >= 10	8	FI
PIC 9(n) COMP-3|PACKED-DECIMAL	(n/2)+1	PD
PIC S9(n) COMP-3|PACKED-DECIMAL	(n/2)+1	PD
COMP-1	4	FL
COMP-2	8	FL

Note:

1. PIC 9(x)V9(y) can be treated like PIC 9(n) where n=x+y. (COBOL does NOT store the decimal point internally.)
2. PIC S9(x)V9(y) can be treated like PIC S9(n) where n=x+y. (COBOL does NOT store the decimal point internally.)

Appendix D. EBCDIC and ASCII collating sequences

EBCDIC

Table 109 on page 839 shows the collating sequence for EBCDIC character and unsigned decimal data. The collating sequence ranges from low (00000000) to high (11111111). The bit configurations which do not correspond to symbols (that is, 0 through 73, 81 through 89, and so forth) are not shown. Some of these correspond to control commands for the printer and other devices.

ALTSEQ, CHALT, and LOCALE can be used to select alternate collating sequences for character data.

Packed decimal, zoned decimal, fixed-point, and normalized floating-point data are collated algebraically, that is, each quantity is interpreted as having a sign.

Table 109. EBCDIC Collating Sequence

EBCDIC Collating SequenceCollating Sequence	**Bit Configuration**	**Symbol**	**Meaning**
0	00000000		
.			
.			
64	01000000	SP	Space
.			
.			
74	01001010	θ	Cent sign
75	01001011	.	Period, decimal point
76	01001100	<	Less than sign
77	01001101	(	Left parenthesis
78	01001110	+	Plus sign
79	01001111	I	Vertical bar, Logical OR
80	01010000	&	Ampersand
.			
.			
90	01011010	!	Exclamation point
91	01011011	$	Dollar sign
92	01011100	*	Asterisk
93	01011101	)	Right parenthesis
94	01011110	;	Semicolon
95	01011111		Logical not
96	01100000	—	Minus, hyphen
97	01100001	/	Slash

Table 109. EBCDIC Collating Sequence (continued)

EBCDIC Collating SequenceCollating Sequence	Bit Configuration	Symbol	Meaning
107	01101011	,	Comma
108	01101100	%	Percent sign
109	01101101	_	Underscore
110	01101110	>	Greater than sign
111	01101111	?	Question mark
.			
.			
122	01111010	:	Colon
123	01111011	#	Number sign
124	01111100	@	Commercial At
125	01111101	'	Apostrophe, prime
126	01111110	=	Equal sign
127	01111111	"	Quotation marks
.			
.			
129	10000001	a	
130	10000010	b	
131	10000011	c	
132	10000100	d	
133	10000101	e	
134	10000110	f	
135	10000111	g	
136	10001000	h	
137	10001001	i	
.			
.			
145	10010001	j	
146	10010010	k	
147	10010011	l	
148	10010100	m	
149	10010101	n	
150	10010110	0	
151	10010111	p	

Table 109. EBCDIC Collating Sequence (continued)

EBCDIC Collating SequenceCollating Sequence	Bit Configuration	Symbol	Meaning
152	10011000	q	
153	10011001	r	
.			
.			
162	10100010	s	
163	10100011	t	
164	10100100	u	
165	10100101	v	
166	10100110	w	
167	10100111	x	
168	10101000	y	
169	10101001	z	
193	11000001	A	
194	11000010	B	
195	11000011	C	
196	11000100	D	
197	11000101	E	
198	11000110	F	
199	11000111	G	
200	11001000	H	
201	11001001	I	
.			
.			
209	11010001	J	
210	11010010	K	
211	11010011	L	
212	11010100	M	
213	11010101	N	
214	11010110	O	
215	11010111	P	
216	11011000	Q	
217	11011001	R	
.			

Table 109. EBCDIC Collating Sequence (continued)			
EBCDIC Collating SequenceCollating Sequence	**Bit Configuration**	**Symbol**	**Meaning**
.			
226	11100010	S	
227	11100011	T	
228	11100100	U	
229	11100101	V	
230	11100010	W	
231	11100111	X	
232	11101000	Y	
233	11101001	Z	
.			
.			
240	11110000	0	
241	11110001	1	
242	11110010	2	
243	11110011	3	
244	11110100	4	
245	11110101	5	
246	11110110	6	
247	11110111	7	
248	11111000	8	
249	11111001	9	
.			
.			
255	11111111		

ASCII

Table 110 on page 843 shows the collating sequence for ASCII, character, and unsigned decimal data. The collating sequence ranges from low (00000000) to high (01111111). Bit configurations that do not correspond to symbols are not shown.

Packed decimal, zoned decimal, fixed-point normalized floating-point data, and the signed numeric data formats are collated algebraically; that is, each quantity is interpreted as having a sign.

Table 110. ASCII Collating Sequence

ASCII Collating SequenceCollating Sequence	**Bit Configuration**	**Symbol**	**Meaning**
0	00000000	Null	
.			
.			
32	00100000	SP	Space
33	00100001	!	Exclamation point
34	00100010	"	Quotation mark
35	00100011	#	Number sign
36	00100100	$	Dollar sign
37	00100101	%	Percent
38	00100110	&	Ampersand
39	00100111	'	Apostrophe, prime
.			
.			
40	00101000	(	Opening parenthesis
41	00101001	)	Closing parenthesis
42	00101010	*	Asterisk
43	00101011	+	Plus
44	00101100	,	Comma
45	00101101	—	Hyphen, minus
46	00101110	.	Period, decimal point
47	00101111	/	Slash
48	00110000	0	
49	00110001	1	
50	00110010	2	
51	00110011	3	
52	00110100	4	
53	00110101	5	
54	001101100	6	
55	00110111	7	
56	00111000	8	
57	00111001	9	
58	00111010	:	Colon
59	00111011	;	Semicolon

Table 110. ASCII Collating Sequence (continued)

ASCII Collating SequenceCollating Sequence	Bit Configuration	Symbol	Meaning
60	00111100	<	Less than
61	00111101	=	Equals
62	00111110	>	Greater than
63	00111111	?	Question mark
64	01000000	@	Commercial At
65	01000001	A	
66	01000010	B	
67	01000011	C	
68	01000100	D	
69	01000101	E	
70	01000110	F	
71	01000111	G	
72	01001000	H	
73	01001001	I	
74	01001010	J	
75	01001011	K	
76	01001100	L	
77	01001101	M	
78	01001110	N	
79	01001111	O	
80	01010000	P	
81	01010001	Q	
82	01010010	R	
83	01010011	S	
84	01010100	T	
85	01010101	U	
86	01010110	V	
87	01010111	W	
88	01011000	X	
89	01011001	Y	
90	01011010	Z	
91	01011011	[	Opening bracket
92	01011100	\	Reverse slash

Table 110. ASCII Collating Sequence (continued)

ASCII Collating SequenceCollating Sequence	**Bit Configuration**	**Symbol**	**Meaning**
93	01011101	]	Closing bracket
94	01011110	^	Circumflex, Logical NOT
95	01011111	_	Underscore
96	01100000	`	Grave Accent
97	01100001	a	
98	01100010	b	
99	01100011	c	
100	01100100	d	
101	01100101	e	
102	01100110	f	
103	01100111	g	
104	01101000	h	
105	01101001	i	
106	01101010	j	
107	01101011	k	
108	01101100	l	
109	01101101	m	
110	01101110	n	
111	01101111	o	
112	01110000	p	
113	01110001	q	
114	01110010	r	
115	01110011	s	
116	01110100	t	
117	01110101	u	
118	01110110	v	
119	01110111	w	
120	01111000	x	
121	01111001	y	
122	01111010	z	
123	01111011	{	Opening Brace
124	01111100	I	Vertical Line
125	01111101	}	Closing Brace

Table 110. ASCII Collating Sequence (continued)

ASCII Collating SequenceCollating Sequence	**Bit Configuration**	**Symbol**	**Meaning**
126	01111110	~	Tilde

Appendix E. DFSORT abend processing

This appendix explains how DFSORT processes an abend. It is intended to help you get the dump you need to diagnose the error causing the abend.

All abend dumps produced by DFSORT are system abend dumps that can be processed by standard dump analysis programs. A dump will be generated if you have included a SYSUDUMP, SYSABEND, or SYSMDUMP DD statement in your application. The actual output of the system dump depends on the system parameters specified in the IEADMP00, IEAABD00 or IEADMR00 members of SYS1.PARMLIB by your installation.

At the beginning of each run, DFSORT establishes an ESTAE recovery routine to trap system or user abends for Blockset and Peer/Vale applications. You can delete the routine by specifying installation option ESTAE=NO, or by specifying NOESTAE on the DEBUG control statement. We recommend that you always run with ESTAE in effect so that in the event of an abend the following benefits are available:

- In general, you get dumps closer to the time of the abend.
- You get additional information useful in diagnosing the problem causing the abend.
- If you have activated SMF, an ICETEXIT routine, or an EFS program, DFSORT attempts to continue processing. That is, an SMF record is created, the termination exit is called, or Major Calls 4 and 5 are made to the EFS program before the application terminates processing. Of course, if one of these functions caused the abend, that function will not complete successfully.

At the end of its recovery routine, DFSORT always returns control to the system to allow termination to continue. The system will then invoke the next higher level ESTAE recovery routine.

Checkpoint/restart

Checkpoint/Restart is a facility of the operating system that allows information about an application to be recorded so that same application can be restarted after abnormal termination or after some portion of the application has been completed. Restart can take place immediately or be deferred until the application is resubmitted.

DFSORT takes checkpoints when requested during a sort that uses the Peerage or Vale techniques.

To have DFSORT record checkpoints you must code a SORTCKPT DD statement and ensure the Peerage or Vale technique is selected. See “SORTCKPT DD statement” on page 73 and “OPTION control statement” on page 173 for more information on the SORTCKPT and CKPT options, respectively.

In general, no checkpoints are taken if the following conditions exist:

- No work data set is specified.
- The application is a copy or merge.
- Blockset is selected.

Note:

1.
2. No ANSI Standard Label tape files can be open during Checkpoint/Restart processing.
3. Do not specify CHKPT=EOV on any DFSORT DD statement.

For more information on the Checkpoint/Restart facility, see *z/OS DFSMSdfp Checkpoint/Restart*.

DFSORT abend categories

There are two categories of abends for DFSORT: unexpected abends and user abends issued by DFSORT.

- Unexpected abends

These are system abends or user abends not issued by DFSORT. The abend code in these cases is the system abend code or the user abend code. See *z/OS DFSORT Messages, Codes and Diagnosis Guide* for information about detecting common user errors and reporting DFSORT program failures.

- User abends issued by DFSORT

  DFSORT will issue user abends under the following circumstances:

  –
  – The ABEND or ABSTP option is in effect and DFSORT encounters an error that prevents completion of the run.
  – DFSORT detects an error in its internal logic.

  See *z/OS DFSORT Messages, Codes and Diagnosis Guide* for complete information about user abends issued by DFSORT.

Abend recovery processing for unexpected abends

DFSORT normally has an ESTAE recovery routine established to trap system or user exit routine abends for Blockset and Peer/Vale applications. If an abend occurs, the system will pass control to this routine. The DFSORT ESTAE recovery routine functions are as follows:

- Abend dump

  The recovery routine will first have the system issue an abend dump to capture the environment at the time the error occurred.

- Termination functions

  DFSORT tries to accomplish the following tasks when they are specified, whether the program terminates normally or abnormally.

  – Calls 4 and 5 to an EFS program
  – Create the SMF record
  – Call the ICETEXIT routine

  The DFSORT recovery routine runs any of the previous functions specified if they have not already been run at the time of the abend.

- Abend information message

  For unexpected system or user exit routine abends, the DFSORT recovery routine issues message ICE185A giving information about when the abend occurred. The description of this message is in *z/OS DFSORT Messages, Codes and Diagnosis Guide*.

- Snap dumps

  The DFSORT recovery routine provides a snap dump of the system diagnostic work area (SDWA). The snap dumps are written to a dynamically allocated data set whether or not a SYSUDUMP (or SYSABEND or SYSMDUMP) DD statement is included in the application.

- Copy system diagnostic work area

  If an invoking program passes the address of an SDWA area in the 24-bit or extended parameter list, DFSORT will copy the first 104 or 112 bytes of the system diagnostic work area into the user SDWA area. See Chapter 6, “Invoking DFSORT from a program,” on page 517 for more information.

- Continuation of an application after successful SORTOUT output

  If an unexpected abend occurs after the sort, merge, or copy application writes the SORTOUT data set successfully, DFSORT issues message ICE186A and completes its normal cleanup and termination functions. The SORTOUT data set written by DFSORT is closed. The run is successful except for the function causing the abend. Message ICE186A says that the SORTOUT data set is usable even though the run has abended. You can then decide to use the SORTOUT data set or rerun the application.

- DFSORT returns control to the system at the end of its abend recovery processing so that recovery routines can be invoked.

The DFSORT abend recovery routine functions described previously may not be performed after an abend if NOESTAE is in effect. The DFSORT ESTAE recovery routine is always established at the beginning of a run. It is deleted early in DFSORT processing if NOESTAE is in effect.

Processing of error abends with A-type messages

When DFSORT encounters a critical error, it issues an A-type message and terminates. You can specify that DFSORT is to terminate the application with an abend through the ABEND or ABSTP options.

If abend termination is in effect and DFSORT encounters a critical error, DFSORT first causes an abend dump to capture the environment at the time of the error. Then, it issues the A-type message. It also runs the termination functions described earlier before terminating with an abend. The abend code will be the error message number, or a number between 1-99, as determined during installation with the ABCODE installation option.

With NOESTAE and ABEND in effect, the abend dump is produced after the A-type message is printed and other termination functions are run. As a result, the dump produced might not reflect the conditions at the time of the error. It may not include the module that encountered the error.

With NOESTAE and ABSTP in effect, the correct module will be dumped but the A-type message will not be issued.

CTRx abend processing

The CTRx option can be used to diagnose a problem by requesting that DFSORT abend during record input or output processing. See the DEBUG control statement in Chapter 3, “Using DFSORT program control statements,” on page 77. DFSORT will cause an 0C1 abend when the CTRx count is satisfied. The DFSORT ESTAE recovery routine will process the abend in the same way as it does for an unexpected abend described earlier.

The DFSORT ESTAE recovery routine will return control to the system, which will pass control to any ESTAE recovery routines established by invoking programs.

As described earlier, the DFSORT ESTAE recovery routine will save the first 104 or 112 bytes of the system diagnostic work area in the invoking program's SDWA area if the address of the area is passed to DFSORT.

Since PL/I normally has an ESPIE in effect to intercept program checks (0Cx abend codes), the DFSORT ESTAE recovery routine is not entered after these errors unless you have specified NOSPIE. DFSORT abend recovery processing will occur for all other types of abends.

Invocations from COBOL programs or use of COBOL exits can result in more than one abend dump.

Appendix F. Accessibility

Accessible publications for this product are offered through IBM Documentation (www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the Contact the z/OS team web page (www.ibm.com/systems/campaignmail/z/zos/contact_z) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information 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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com®/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, JES3, and MVS™, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming interface information

This publication primarily documents information that is NOT intended to be used as Programming Interfaces of DFSORT.

This publication also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of DFSORT. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at Copyright and Trademark information (www.ibm.com/legal/copytrade.shtml).

Index

Special Characters

Numerics

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

Product Number: 5650-ZOS

SC23-6878-50