z/OS

DFSORT: Getting Started

Version 2 Release 3

SC23-6880-30

Chapter 1. What is DFSORT?

DFSORT is IBM's high-performance sort, merge, copy, analysis, and reporting product for z/OS.

With DFSORT, you can sort, merge, and copy data sets. You can use DFSORT to do simple tasks such as alphabetizing a list of names, or you can use it to aid complex tasks such as taking inventory or running a billing system. DFSORT gives you versatile data handling capabilities at the record, field and bit level.

DFSORT on the World Wide 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 by anonymous FTP to:

`ftp.software.ibm.com/storage/dfsort/mvs/`

Data sets, records and fields

The information you manipulate with DFSORT is contained in *data sets*. The term *data set* refers to a file that contains one or more records. Any named group of records is called a *data set*. The terms *data set* and *file* are synonymous, and are used interchangeably in this document.

A data set contains the information that you want to sort, copy, or merge. For most of the processing done by DFSORT, the whole data set is affected. However, some forms of DFSORT processing involve only certain individual records in that data set.

Data sets can be *cataloged*, which permits the data set to be referred to by name without specifying where the data set is stored. A cataloged data set should not be confused with a cataloged *procedure*. A cataloged procedure is a named collection of JCL stored in a data set, and a cataloged data set is a data set whose name is recorded by the system.

Throughout this document, the term *record* refers to a collection of related information used as a unit, such as one item in a data base or personnel data about one member of a department. The term *field* refers to a specific portion of a record used for a particular category of data, such as an employee's name or department.

DFSORT can sort, copy or merge fixed-length or variable-length records. The type and length of a data set is defined by its record format (RECFM) and logical record length (LRECL). Fixed-length data sets have a RECFM of F, FB, FBS, and so on. Variable-length data sets have a RECFM of V, VB, VBS, and so on. For simplicity in this document, the terms "FB data set" and "FB records" are used as short-hand for fixed-length data sets and records, respectively, and the terms "VB data set" and "VB records" are used as short-hand for variable-length record data sets and variable-length records, respectively.

A data set with RECFM=FB and LRECL=25 is a fixed-length (FB) data set with a record length of 25-bytes (the B is for blocked). For an FB data set, the LRECL tells you the length of each record in the data set; all of the records are the same length. The first data byte of an FB record is in position 1. A record in an FB data set with LRECL=25 might look like this:

```
Positions 1-3:  Country Code = 'USA'
Positions 4-5:  State Code = 'CA'
Positions 6-25:  City = 'San Jose' padded with 12 blanks on the right
```

A data set with RECFM=VB and LRECL=25 is a variable-length (VB) data set with a maximum record length of 25-bytes. For a VB data set, different records can have different lengths. The first four bytes of each record contain the Record Descriptor Word or RDW, and the first two bytes of the RDW contain the length of that record (in binary). The first data byte of a VB record is in position 5, after the 4-byte RDW in positions 1-4. A record in a VB data set with LRECL=25 might look like this:

```
Positions 1-2:    Length in RDW = hex 000E = decimal 14
Positions 3-4:    Zeros in RDW = hex 0000 = decimal 0
Positions 5-7:    Country Code = 'USA'
Positions 8-9:    State Code =  'CA'
Positions 10-17:  City = 'San Jose'
```

Unless otherwise noted, the examples in this document process FB data sets, which are easier to work with and describe. However, special considerations for processing VB data sets are discussed throughout this document whenever appropriate.

Sorting data sets

You can use DFSORT to rearrange the records in your data sets. *Sorting* is arranging records in either ascending or descending order within a file. Table 2 shows a sample data set of names, first sorted in ascending order, then in descending order.

Table 2. DFSORT Arranges Information in Ascending and Descending Order

Unsorted Data Set	Sorted Ascending	Sorted Descending
Andy	Andy	Edward
Edward	Betty	Dan
Carol	Carol	Carol
Dan	Dan	Betty
Betty	Edward	Andy

You can sort data in many different formats. Table 3 on page 5 shows the most commonly used DFSORT data formats and the format identifiers you use to specify them.

Table 3. Commonly Used Data Formats

Data Format	Format Identifier
EBCDIC (Character)	CH
Binary (Unsigned Numeric)	BI
Fixed-point (Signed Numeric)	FI
Zoned Decimal (Signed Numeric)	ZD
Packed Decimal (Signed Numeric)	PD
Floating Sign (Signed Numeric)	FS
Free Form (Unsigned Numeric)	UFF
Free Form (Signed Numeric)	SFF

Refer to *z/OS DFSORT Application Programming Guide* for complete details of the available formats.

Merging data sets

You can also use DFSORT to merge data sets. DFSORT *merges* data sets by combining two or more files of sorted records to form a single data set of sorted records.

Table 4. DFSORT Merges Two Data Sets into One Data Set

Data Set 1	Data Set 2	Merged Data Set
Andy	Amy	Amy
Betty	Chris	Andy
Carol	Sue	Betty
Dan		Carol
Edward		Chris
		Dan
		Edward
		Sue

The data sets you merge must be previously sorted into the same order (ascending or descending).

Copying data sets

DFSORT can also copy data sets without any sorting or merging taking place. You copy data sets in much the same way that you sort or merge them.

Joining data sets

DFSORT can perform various "join" operations on two data sets by one or more keys. You can 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 data sets 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.

What else can you do with DFSORT?

While sorting, merging, or copying data sets, you can also perform other tasks such as the following:

- Select a subset of records from an input data set. You can include or omit records that meet specified criteria. For example, when sorting an input data set

containing records of course documents from many different school departments, you can sort the documents for only one department.

- Reformat records in a variety of ways. You can build your records one item at a time, only overlay specific columns, or reformat different records in different ways. You can edit, change, add or delete fields. You can convert date fields of one type to another type and perform date field arithmetic. You can perform find and replace operations on your records. You can perform various operations on groups of records. You can work with fixed position/length fields directly or convert variable position/length fields (such as comma separated values) to fixed parsed fields for further processing. You can also insert blanks, zeros, strings, current date, future date, past date, current time, sequence numbers, decimal constants, and the results of arithmetic instructions before, between, and after input fields. For example, you can create an output data set that contains character strings and only certain edited fields from the input data set, arranged differently.
- Sum the values in selected records while sorting or merging (but not while copying). In the example of a data set containing records of course books, you can use DFSORT to add up the dollar amounts of books for one school department.
- Create multiple output data sets and simple or complex reports from a single pass over an input data set. For example, you can create a different output data set for the records of each department.
- Convert VB data sets to FB data sets, or convert FB data sets to VB data sets.
- Sample or repeat records.
- Sort, merge, include or omit records according to the collating rules defined in a selected locale.
- Alter the collating sequence when sorting or merging records (but not while copying). For example, you can have the lowercase letters collate after the uppercase letters.
- Sort, merge, or copy Japanese data if the IBM Double Byte Character Set Ordering Support (DBCS Ordering) (5665-360 Licensed Program, Release 2.0 or an equivalent product) is used with DFSORT to process the records.
- Sort, merge of Unicode data format records according to the collating rules defined in a selected collation version.

Creating and running DFSORT jobs

Processing data sets with DFSORT involves two steps:

1. Creating a DFSORT job
2. Running a DFSORT job

You can run a DFSORT job by invoking processing in a number of ways:

- With a JCL EXEC statement, using the name of the program or the name of the cataloged procedure
- Within programs written in COBOL, PL/I, or basic Assembler language

In this document, the phrases *directly* or *JCL-invoked* mean that the DFSORT program is initiated by a JCL EXEC statement with PGM=SORT or PGM=ICEMAN. The phrases *called by a program* or *dynamically invoked* mean that the DFSORT program is initiated from another program.

Writing jobs

You can use DFSORT by writing JCL and DFSORT control statements no matter how your site has installed DFSORT. Part 1 contains instructions on writing JCL and DFSORT program control statements.

You must prepare JCL statements and DFSORT program control statements to invoke DFSORT processing. JCL statements are processed by your operating system. They describe your data sets to the operating system, and initiate DFSORT processing. DFSORT program control statements are processed by DFSORT. They describe and initiate the processing you want to do.

Summary of DFSORT control statements

The functions of the most important DFSORT control statements can be summarized briefly as follows:

SORT Describes the fields for a sort application, or requests a copy application.

MERGE
Describes the fields for a merge application, or requests a copy application.

OPTION
Overrides installation defaults, or requests optional features or a copy application.

INCLUDE
Describes the criteria to be used to include records before they are sorted, copied or merged.

OMIT Describes the criteria to be used to omit records before they are sorted, copied or merged.

INREC
Describes how records are to be reformatted before they are sorted, copied or merged.

OUTREC
Describes how records are to be reformatted after they are sorted, copied or merged.

SUM Describes how fields are to be summed after sorting or merging.

OUTFIL
Describes various types of processing to be performed for one or more output data sets after sorting, copying or merging.

JOINKEYS, JOIN, REFORMAT
Describes a "joinkeys" application for joining two files on one or more keys.

The functions of the less important DFSORT control statements can be summarized briefly as follows:

ALTSEQ
Describes changes to the normal translation table.

MODS
Describes user exit routines.

RECORD
Supplies data set record type and length information when needed.

DEBUG
Requests diagnostic features.

END Marks the end of the control statements.

Running jobs

You can run DFSORT jobs directly with a JCL EXEC statement that uses PGM=SORT or PGM=ICEMAN. Or, you can call DFSORT dynamically from a COBOL, Assembler, PL/I, or other type of program.

Creating and using the sample data sets

Many of the examples in this document refer to the sample data sets SORT.SAMPIN, SORT.SAMPADD, SORT.BRANCH and SORT.SAMPOUT. Appendix A, "Creating the Sample Data Sets" shows you how to create your own copies of these data sets, using a program called ICESAMP shipped with DFSORT, if you want to try the examples in this document that use them.

Note: Some of the examples use data sets other than SORT.SAMPIN, SORT.SAMPOUT, SORT.SAMPADD, and SORT.BRANCH. You can either create data sets from scratch to match the ones used in the text, or else perform a similar exercise on data sets you already have.

Before you begin, turn to Appendix B, “Descriptions of the sample data sets,” on page 189. Many of the examples in this document refer to the sample bookstore data sets as the input data sets, so you should become familiar with them. The input data sets contain the data that you want arranged or sorted. You must specify an input data set for every DFSORT job you run. The sample bookstore data set is named **SORT.SAMPIN** and the additional bookstore data set is named **SORT.SAMPADD**.

Each record in the bookstore data sets has 12 fields (book title, author’s last name, and so on). A record can be represented by one horizontal row on the page. A field can be represented by one vertical column on the page.

To sort a data set, you choose one or more fields that you want to use to order the records (arrange in ascending or descending order). These fields are called *control fields* (or, in COBOL, *keys*).

As you work through the exercises on the following pages, remember that each *entire* record is sorted, not just the control field. However, for the sake of simplicity, the figures in the text show only the control fields being discussed. The sorted records actually contain all of the fields, but one page is not wide enough to show them. Appendix B, “Descriptions of the sample data sets,” on page 189, shows all of the fields in each record. It is also arranged with headings and numbers that show the byte positions of each field. The numeric fields are in binary format (see Table 3 on page 5) and therefore will not appear on most displays as they do in this document. Methods you can use to arrange and view the data are explained in the chapters on DFSORT functions that follow.

Table 5 on page 9 shows an example of sorted fields. Notice the line of numbers above the sorted fields. These numbers represent the byte positions of those fields. You use byte positions to identify fields to DFSORT. The examples show the byte positions to help you while you are learning to use DFSORT. The byte positions do not actually appear in any of your processed data sets.

In Table 5, the first two records, which show nothing in the course department fields, are general purpose books not required for a particular course. For this example, the control field is the Course Department field.

Table 5. Sample Bookstore Data Set Sorted by Course Department in Ascending Order

Book Title	Course Department	Price
1 75	110 114	170 173
LIVING WELL ON A SMALL BUDGET		9900
PICK'S POCKET DICTIONARY		295
INTRODUCTION TO BIOLOGY	BIOL	2350
SUPPLYING THE DEMAND	BUSIN	1925
STRATEGIC MARKETING	BUSIN	2350
COMPUTER LANGUAGES	COMP	2600
VIDEO GAME DESIGN	COMP	2199
COMPUTERS: AN INTRODUCTION	COMP	1899
NUMBERING SYSTEMS	COMP	360
SYSTEM PROGRAMMING	COMP	3195
INKLINGS: AN ANTHOLOGY OF YOUNG POETS	ENGL	595
EDITING SOFTWARE MANUALS	ENGL	1450
MODERN ANTHOLOGY OF WOMEN POETS	ENGL	450
THE COMPLETE PROOFREADER	ENGL	625
SHORT STORIES AND TALL TALES	ENGL	1520
THE INDUSTRIAL REVOLUTION	HIST	795
EIGHTEENTH CENTURY EUROPE	HIST	1790
CRISIS OF THE MIDDLE AGES	HIST	1200
INTRODUCTION TO PSYCHOLOGY	PSYCH	2200
ADVANCED TOPICS IN PSYCHOANALYSIS	PSYCH	2600

Also notice that records in Table 5 with *equally collating control fields* (in this case,the same department) appear in their original order. For example, within the Computer Science department (COMP), the title *Video Game Design* still appears before *Computers: An Introduction*.

You can control whether records with equally collating control fields appear in their original order or whether DFSORT orders them randomly. The system programmer sets defaults at installation time that you can change with some DFSORT options at run time. The examples in this document assume that the default is for records with equally collating control fields to appear in their original order.

Summary

So far in Getting Started you covered the following concepts:

- You can sort, copy, or merge data sets using DFSORT.
- You can write JCL and DFSORT program control statements to create and process DFSORT jobs.
- You can run DFSORT jobs directly or call DFSORT from a program.

In addition, this chapter covered how to use and read the sample data sets provided with DFSORT. Now continue with tutorials on how to write DFSORT control statements.

Part 2. Learning to write JCL and DFSORT control statements

Chapter 2. Sorting, merging, and copying data sets

This tutorial shows you how to sort, merge, and copy data sets by writing DFSORT program control statements that are processed with JCL.

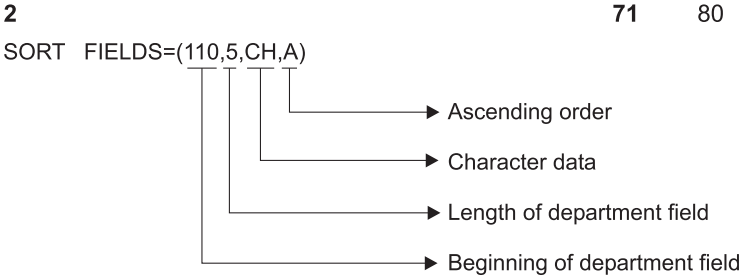
DFSORT program control statements are input in the JCL used to run DFSORT. To keep the instructions simple, the program control statements are covered first and the related JCL statements are explained afterward. For most of the tutorials you will concentrate on JCL-invoked DFSORT, that is, running DFSORT with JCL. Information on calling DFSORT from a program (dynamic invocation) is presented in Chapter 9, “Calling DFSORT from a program,” on page 117.

Sorting data sets

To use DFSORT directly (JCL-invoked), write a SORT control statement to describe the control fields, and the order in which you want them sorted. The control statements you write are part of the SYSIN data set in the JCL. The SYSIN data set is typically specified as //SYSIN DD * followed by "inline" control statements (as shown in the examples in this document). However, a sequential data set, or a member of a partitioned data set, with the control statements as records can also be used for the SYSIN data set.

You can use SORT with all of the other DFSORT control statements.

A SORT statement that sorts the bookstore records by the course department field (as shown in Table 7 on page 14) looks like this:



Make sure that the statement is coded between columns 2 and 71.

Here are the steps for writing this SORT statement:

Table 6. Steps to Create the SORT Statement to Sort by Department

Step	Action
1	Leave at least one blank, and type **SORT**
2	Leave at least one blank and type **FIELDS=**

Table 6. Steps to Create the SORT Statement to Sort by Department (continued)

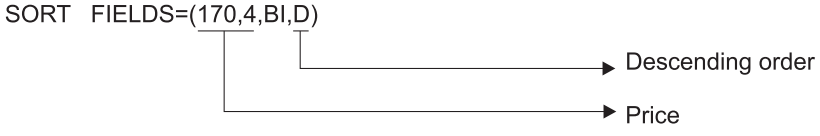
Step	Action
3	Type, in parenthesis and separated by commas: 1. Where the course department field begins, relative to the beginning of the record in the bookstore data set (the first position is byte 1). The course department field begins at byte **110**. 2. The length of the department field in bytes. The department field is **5** bytes long. 3. A format identifier for the data format. The department field contains character data, which you specify as **CH**. (Table 3 on page 5 shows the codes for the most commonly used data formats.) 4. The letter **A**, for ascending order.

Remember that although Table 7 shows only certain fields, the displayed fields are not the only ones in the output data set. Your output data set will more closely resemble the fold-out of the sample bookstore data set.

Table 7. Sample Bookstore Data Set Sorted by Course Department in Ascending Order

Book Title	Course Department
1 75	110 114
LIVING WELL ON A SMALL BUDGET	
PICK'S POCKET DICTIONARY	
INTRODUCTION TO BIOLOGY	BIOL
SUPPLYING THE DEMAND	BUSIN
STRATEGIC MARKETING	BUSIN
COMPUTER LANGUAGES	COMP
VIDEO GAME DESIGN	COMP
COMPUTERS: AN INTRODUCTION	COMP
NUMBERING SYSTEMS	COMP
SYSTEM PROGRAMMING	COMP
INKLINGS: AN ANTHOLOGY OF YOUNG POETS	ENGL
EDITING SOFTWARE MANUALS	ENGL
MODERN ANTHOLOGY OF WOMEN POETS	ENGL
THE COMPLETE PROOFREADER	ENGL
SHORT STORIES AND TALL TALES	ENGL
THE INDUSTRIAL REVOLUTION	HIST
EIGHTEENTH CENTURY EUROPE	HIST
CRISES OF THE MIDDLE AGES	HIST
INTRODUCTION TO PSYCHOLOGY	PSYCH
ADVANCED TOPICS IN PSYCHOANALYSIS	PSYCH

To sort the records in descending order, specify **D** instead of A. For example, to sort the prices for each book in descending order, type:



The sort order is bytes 170 through 173 as binary data in descending sequence. Table 8 on page 15 shows the results of the sort in descending order.

Table 8. Sample Bookstore Data Set Sorted by Price in Descending Order

Book Title	Price
1 75	170 173
LIVING WELL ON A SMALL BUDGET	9900
SYSTEM PROGRAMMING	3195
COMPUTER LANGUAGES	2600
ADVANCED TOPICS IN PSYCHOANALYSIS	2600
STRATEGIC MARKETING	2350
INTRODUCTION TO BIOLOGY	2350
INTRODUCTION TO PSYCHOLOGY	2200
VIDEO GAME DESIGN	2199
SUPPLYING THE DEMAND	1925
COMPUTERS: AN INTRODUCTION	1899
EIGHTEENTH CENTURY EUROPE	1790
SHORT STORIES AND TALL TALES	1520
EDITING SOFTWARE MANUALS	1450
CRISES OF THE MIDDLE AGES	1200
THE INDUSTRIAL REVOLUTION	795
THE COMPLETE PROOFREADER	625
INKLINGS: AN ANTHOLOGY OF YOUNG POETS	595
MODERN ANTHOLOGY OF WOMEN POETS	450
NUMBERING SYSTEMS	360
PICK'S POCKET DICTIONARY	295

Sorting by multiple fields

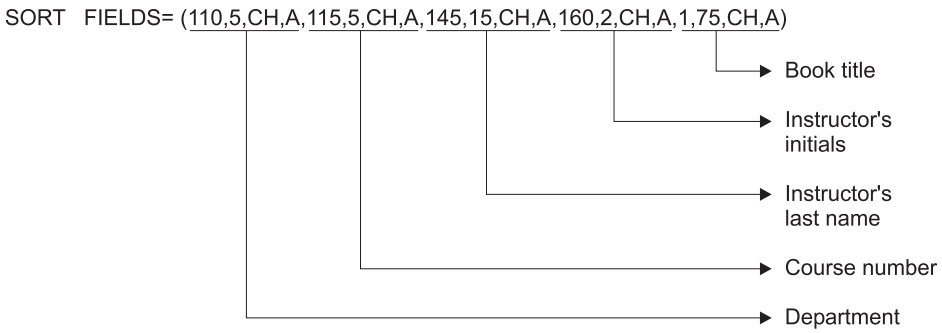
You can further sort the records in the bookstore data set by specifying multiple control fields. When you specify two or more control fields, you specify them in the order of greater to lesser priority. Note that control fields might overlap or be contained within other control fields.

Table 9 on page 16 shows how the records would be sorted if you specified the following control fields in the order they are listed:

1. Course department
2. Course number
3. Instructor's last name
4. Instructor's initials
5. Book title.

So, if two records have the same department, they are sorted by course number. If they also have the same course number, they are sorted by instructor's last name. If they also have the same last name, they are sorted by initials. Finally, if they also have the same initials, they are sorted by title.

Specify the location, length, data format, and order for each of the control fields, as follows:

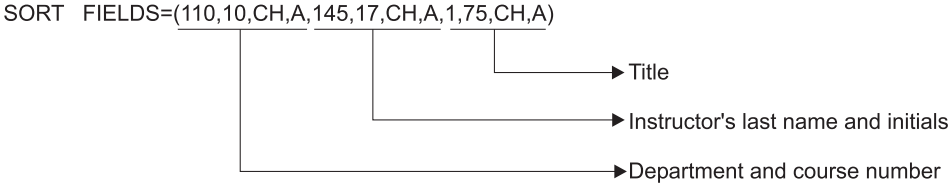


The records are sorted as shown in Table 9.

Table 9. Sample Bookstore Data Set Sorted by Multiple Fields

Book Title	**Course Department**	**Course Number**	**Instructor's Last Name**	**Instructor's Initials**
1 75	110 114	115 119	145 159	160 161
LIVING WELL ON A SMALL BUDGET				
PICK'S POCKET DICTIONARY				
INTRODUCTION TO BIOLOGY	BIOL	80521	GREENBERG	HC
STRATEGIC MARKETING	BUSIN	70124	LORCH	HH
SUPPLYING THE DEMAND	BUSIN	70251	MAXWELL	RF
NUMBERING SYSTEMS	COMP	00032	CHATTERJEE	AN
COMPUTER LANGUAGES	COMP	00032	CHATTERJEE	CL
COMPUTERS: AN INTRODUCTION	COMP	00032	CHATTERJEE	CL
SYSTEM PROGRAMMING	COMP	00103	SMITH	DC
VIDEO GAME DESIGN	COMP	00205	NEUMANN	LB
SHORT STORIES AND TALL TALES	ENGL	10054	BUCK	GR
EDITING SOFTWARE MANUALS	ENGL	10347	MADRID	MM
THE COMPLETE PROOFREADER	ENGL	10347	MADRID	MM
INKLINGS: AN ANTHOLOGY OF YOUNG POETS	ENGL	10856	FRIEDMAN	KR
MODERN ANTHOLOGY OF WOMEN POETS	ENGL	10856	FRIEDMAN	KR
THE INDUSTRIAL REVOLUTION	HIST	50420	GOODGOLD	ST
CRISES OF THE MIDDLE AGES	HIST	50521	WILLERTON	DW
EIGHTEENTH CENTURY EUROPE	HIST	50632	BISCARDI	HR
INTRODUCTION TO PSYCHOLOGY	PSYCH	30016	ZABOSKI	RL
ADVANCED TOPICS IN PSYCHOANALYSIS	PSYCH	30975	NAKATSU	FL

You can often shorten the length of control statements. You can specify fields together whenever they are next to each other in the record and have the same data format and order. You can shorten this last statement by specifying the department and course number together as one field, and the instructor's last name and initials together as one field.



Also, if all of the control fields have the same data format, you can specify the data format just once, using the FORMAT=f parameter. For example:

```
SORT FORMAT=CH,FIELDS=(110,10,A,145,17,A,1,75,A)
```

If some of the control fields have the same data format, but others don't, you can specify the FORMAT=f parameter along with the individual data formats. For example:

```
SORT FORMAT=CH,FIELDS=(110,10,A,170,4,BI,D,145,17,A,1,75,A)
```

is equivalent to:

```
SORT FIELDS=(110,10,CH,A,170,4,BI,D,145,17,CH,A,1,75,CH,A)
```

example : Sorting UTF16 data and Character 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.

Continuing a statement

If you cannot fit your SORT statement (or any other DFSORT control statement) between columns 2 through 71, you can continue it on the next line. If you end a line with a comma followed by a blank, DFSORT treats the next line as a continuation. The continuation can begin anywhere between columns 2 through 71.

For example:

```
SORT FORMAT=CH,FIELDS=(110,10,A,145,17,A,
   1,75,A)
```

Comment statements

You can mix comment statements with your control statements by starting them with an asterisk (*) in column 1. DFSORT prints comment statements, but otherwise ignores them.

For example:

```
* Sort by department and course number
  SORT FIELDS=(110,10,CH,A)
```

JCL for sorting data sets directly

The job control language (JCL) you need to do a sort depends on whether you run DFSORT directly or call DFSORT from a program. For now, concentrate on running DFSORT directly. Information on calling DFSORT from a program is presented in Chapter 9, “Calling DFSORT from a program,” on page 117.

Your operating system uses the 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 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:

- Invoke DFSORT with an EXEC statement in the input job stream, or with a system macro instruction within another program
- Choose to use EXEC statement cataloged procedures to invoke DFSORT
- Choose to specify PARM options on the EXEC statement
- Choose to specify PARM options or control statements in a DFSPARM data set
- Choose to specify control statements in a SYSIN data set
- Want to use program exits to activate routines of your own

Information on when you would choose each of the previous options is explained in *z/OS DFSORT Application Programming Guide*.

The JCL statements you need for most jobs are as follows.

//jobname JOB
: Signals the beginning of a job. At your site, you might be required to specify information such as your name and account number on the JOB statement.

//stepname EXEC
: Signals the beginning of a job step and tells the operating system what program to run. To run DFSORT, write the EXEC statement like this:

```
//stepname  EXEC  PGM=SORT
```

//STEPLIB DD
: The DFSORT program would typically be in a library known to the system, so the //STEPLIB DD statement would **not** be needed. However, if DFSORT is not in a library known to the system, the //STEPLIB DD statement defines the library containing the DFSORT program

//SYSOUT DD
: Defines the data set in which DFSORT messages and control statements are listed.

//SORTIN DD
: Defines the input data set or concatenated input data sets.

//SORTWKdd DD
: Defines a work data set for a sort. Typically not needed, because DFSORT can allocate work data sets for a sort dynamically.

//SORTOUT DD
: Defines the output data set.

//SYSIN DD
Precedes or contains the DFSORT program control statements.

The following is a typical example of JCL to run DFSORT.

```
//EXAMP    JOB  A492,PROGRAMMER
//SORT     EXEC PGM=SORT
//SYSOUT   DD   SYSOUT=A
//SORTIN   DD   DSN=A123456.SORT.SAMPIN,DISP=SHR
//SORTOUT  DD   DSN=A123456.SORT.SAMPOUT,DISP=OLD
//SYSIN    DD   *
    SORT FORMAT=CH,
         FIELDS=(110,10,A,145,17,A,1,75,A)
/*
```

z/OS DFSORT Application Programming Guide contains additional information about running DFSORT directly.

So far
So far in this chapter you covered how to write a SORT program control statement and how to run that sort with JCL statements. The next tutorial explains how to use the MERGE program control statement to merge two data sets.

Merging data sets

Generally, the reason for merging data sets is to add more records to a data set that is already sorted.

For example, assume that the bookstore data set is already sorted by course department and book title (as shown in Table 10), and you want to update it by merging it with a data set that contains five new records, also sorted by course department and book title.

Table 10. Sample Bookstore Data Set Sorted by Course Department and Book Title

Book Title	**Course Department**
1 75	110 114
LIVING WELL ON A SMALL BUDGET	
PICK'S POCKET DICTIONARY	
INTRODUCTION TO BIOLOGY	BIOL
STRATEGIC MARKETING	BUSIN
SUPPLYING THE DEMAND	BUSIN
COMPUTER LANGUAGES	COMP
COMPUTERS: AN INTRODUCTION	COMP
NUMBERING SYSTEMS	COMP
SYSTEM PROGRAMMING	COMP
VIDEO GAME DESIGN	COMP
EDITING SOFTWARE MANUALS	ENGL
INKLINGS: AN ANTHOLOGY OF YOUNG POETS	ENGL
MODERN ANTHOLOGY OF WOMEN POETS	ENGL
SHORT STORIES AND TALL TALES	ENGL
THE COMPLETE PROOFREADER	ENGL
CRISES OF THE MIDDLE AGES	HIST
EIGHTEENTH CENTURY EUROPE	HIST
THE INDUSTRIAL REVOLUTION	HIST
ADVANCED TOPICS IN PSYCHOANALYSIS	PSYCH
INTRODUCTION TO PSYCHOLOGY	PSYCH

For this example, use a new data set such as the one shown in Table 11.

Table 11. Five New Records Sorted by Course Department and Book Title

Book Title		Course Department	
1	75	110	114
INTERNATIONAL COOKBOOK			
WORLD JOURNEYS BY TRAIN			
ARTS AND CRAFTS OF ASIA		ART	
BIOCHEMISTRY		BIOL	
BEHAVIORAL ANALYSIS		PSYCH	

To merge data sets, you write a MERGE control statement and several JCL statements. Whenever you merge data sets, you must make sure that their records have the same format and that they have been previously sorted by the same control fields. You can merge up to 100 data sets at a time.

You can use MERGE with all of the other DFSORT control statements.

Writing the MERGE control statement

The format of the MERGE statement is the same as that of the SORT statement. To merge the bookstore master data set with the data set containing the five new records, write:

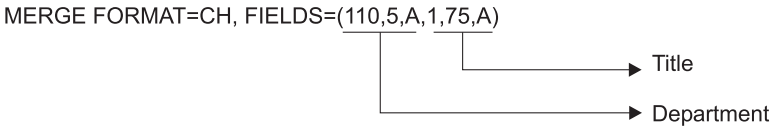


Table 12 shows the merged output.

Table 12. Sample Bookstore Data Set Merged with Five New Records

Book Title		Course Department	
1	75	110	114

Table 12. Sample Bookstore Data Set Merged with Five New Records (continued)

Book Title	Course Department
INTERNATIONAL COOKBOOK	
LIVING WELL ON A SMALL BUDGET	
PICK'S POCKET DICTIONARY	
WORLD JOURNEYS BY TRAIN	
ARTS AND CRAFTS OF ASIA	**ART**
BIOCHEMISTRY	**BIOL**
INTRODUCTION TO BIOLOGY	BIOL
STRATEGIC MARKETING	BUSIN
SUPPLYING THE DEMAND	BUSIN
COMPUTER LANGUAGES	COMP
COMPUTERS: AN INTRODUCTION	COMP
NUMBERING SYSTEMS	COMP
SYSTEM PROGRAMMING	COMP
VIDEO GAME DESIGN	COMP
EDITING SOFTWARE MANUALS	ENGL
INKLINGS: AN ANTHOLOGY OF YOUNG POETS	ENGL
MODERN ANTHOLOGY OF WOMEN POETS	ENGL
SHORT STORIES AND TALL TALES	ENGL
THE COMPLETE PROOFREADER	ENGL
CRISES OF THE MIDDLE AGES	HIST
EIGHTEENTH CENTURY EUROPE	HIST
THE INDUSTRIAL REVOLUTION	HIST
ADVANCED TOPICS IN PSYCHOANALYSIS	PSYCH
BEHAVIORAL ANALYSIS	**PSYCH**
INTRODUCTION TO PSYCHOLOGY	PSYCH

JCL for merging data sets directly

As in a sort, the JCL you need depends on whether you run DFSORT directly or call it from a program. This chapter only discusses running DFSORT directly.

The JCL needed for a merge is the same as that for a sort, with the following exceptions:

- You do *not* need dynamic allocation of work data sets or SORTWKdd DD statements.
- Instead of the SORTIN DD statement, you use SORTINnn DD statements to define the input data sets. The SORTINnn DD statements name the input data sets, and tell how many data sets will be merged. You need one SORTINnn DD statement for each data set being merged. *nn* in SORTIN*nn* is a number from 00 to 99. Thus, if you wanted to merge 5 data sets, you would typically use DD statements for SORTIN01, SORTIN02, SORTIN03, SORTIN04 and SORTIN05.

To merge the pre-sorted bookstore data set and the data set containing the new records, code the following JCL statements for this example. The new data set is called A123456.NEW and the sorted version of the bookstore data set is called A123456.MASTER. For this example, it is assumed that the input data sets are cataloged and that the output data set will be cataloged.

```
//EXAMP    JOB  A492,PROGRAMMER
//MERGE    EXEC PGM=SORT
//SYSOUT   DD   SYSOUT=A
//SORTIN01 DD   DSN=A123456.MASTER,DISP=SHR
//SORTIN02 DD   DSN=A123456.NEW,DISP=SHR
//SORTOUT  DD   DSN=A123456.SORT.SAMPOUT,DISP=OLD
//SYSIN    DD   *
  MERGE  FIELDS=(110,5,CH,A,1,75,CH,A)
/*
```

example : Merging UTF16 data with format

```
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.

In Chapter 9, “Calling DFSORT from a program,” on page 117, you learn how to merge data sets when calling DFSORT from a program.

So far
So far in this chapter you covered how to write both the SORT and MERGE program control statements and how to process those control statements using JCL statements. Now you continue with the tutorial on COPY.

VB data set considerations

A record in a VB data set looks like this:

A record in a VB data set looks like this:

VB data set record		
RDW	Fixed data	Variable data

The RDW (Record Descriptor Word) is a 4-byte binary field with the length of the record in the first two bytes. Fixed data consists of data bytes that are present in every record. Variable data consists of one or more data bytes that may or may not be present in every record, so different records can have different lengths up to the maximum logical record length (LRECL) for the data set.

Starting positions

For FB data sets, the first data byte starts in position 1. However, for VB data sets, the RDW is in positions 1-4, so the first data byte starts in position 5. So when you code your control fields for sorting or merging VB data sets, remember to add 4 to

the starting position to account for the 4-byte RDW. For example, the following SORT statement specifies a CH control field in the third through fifth data bytes of a VB record:

```
SORT FIELDS=(7,3,CH,A)
```

Short control fields

Because VB records have a fixed part and a variable part, it is possible for part of a control field to be missing. Consider this SORT statement:

```
SORT FIELDS=(21,12,CH,A)
```

The control field is in positions 21-32. If your VB records have 25 fixed data bytes and LRECL=45, the records can vary in length from 29 bytes (4-byte RDW, 25 bytes of fixed data, and 0 bytes of variable data) to 45 bytes (4-byte RDW, 25 bytes of fixed data, and 16 bytes of variable data). Records 32 bytes or longer include the entire control field. But records less than 32 bytes have "short" control fields, that is, they are missing some of the bytes at the end of the control field. You cannot validly sort or merge on control fields with missing bytes because missing bytes have no values.

If you know you have VB records with short control fields, you can specify the VLSHRT option, if appropriate, to prevent DFSORT from terminating. For example:

```
OPTION VLSHRT
SORT FIELDS=(21,12,CH,A)
```

VLSHRT tells DFSORT that you want to temporarily replace any missing control field bytes with binary zeros (the zeros are not kept for the output record), thus allowing DFSORT to validly sort or merge on the short control fields.

Attention: If NOVLSHRT is in effect, DFSORT terminates if it finds a short control field in any VB record.

For more information on DFSORT's VLSHRT option, see *z/OS DFSORT Application Programming Guide*.

Copying data sets

With DFSORT you can copy data sets directly without performing a sort or merge.

You can use COPY with all of the other DFSORT control statements except SUM. DFSORT can select and reformat the specific data sets you want to copy by using the control statements covered in later chapters.

You write a copy statement by specifying COPY on the SORT, MERGE, or OPTION statement.

Specifying COPY on the SORT, MERGE, or OPTION statement

The SORT and MERGE statements change very little when you specify COPY. Just replace the information you usually put in parentheses with the word COPY:

```
SORT FIELDS=COPY
MERGE FIELDS=COPY
```

You can also specify COPY on the OPTION statement:

```
OPTION COPY
```

All three of these statements have identical results.

JCL for copying data sets directly

The JCL for a copy application is the same as for a sort, except that you do not need dynamic allocation of work data sets or SORTWKdd DD statements.

This sample JCL will copy the SORT.SAMPIN data set to a temporary output data set using the OPTION COPY statement:

```
//EXAMP    JOB  A492,PROGRAMMER
//COPY     EXEC PGM=SORT
//SYSOUT   DD   SYSOUT=A
//SORTIN   DD   DSN=A123456.SORT.SAMPIN,DISP=SHR
//SORTOUT DD DSN=&&TEMP,DISP=(,PASS),SPACE=(CYL,(5,5)),UNIT=SYSDA
//SYSIN    DD   *
  OPTION COPY
/*
```

You can use SORT FIELDS=COPY or MERGE FIELDS=COPY instead of OPTION COPY to produce the same results.

Summary
In this chapter, you covered the following concepts: • Writing the SORT, COPY, or MERGE program control statements • Using JCL statements to process your sort, copy, or merge As you continue with the tutorials, you will cover two methods of obtaining subsets of your input data set for your output data set. Chapter 3, “Including or omitting records,” on page 25 covers allowable comparison operators, various types of constants, substring search, and padding and truncation rules for INCLUDE and OMIT.

Chapter 3. Including or omitting records

Often, you need only a subset of the records in a data set for an application. This chapter explains how to include or omit only specific records from the input data set for sorting, copying or merging to the output data set.

By removing unneeded records with an INCLUDE or OMIT statement **before sorting, copying or merging,** you can increase the speed of the sort, copy or merge. The fewer the records, the less time it takes to process them.

You select a subset of the records in an input data set by:

- Using an INCLUDE control statement to collect wanted records
- Using an OMIT control statement to exclude unwanted records
- Using an INCLUDE or OMIT parameter on an OUTFIL statement to collect wanted records or exclude unwanted records, respectively. Different INCLUDE and OMIT parameters can be used on different OUTFIL statements.

Your choice of an INCLUDE or OMIT statement depends on which is easier and more efficient to write for a given application. *You cannot use both statements together*.

The information presented in this chapter for the INCLUDE and OMIT statements also applies to the INCLUDE and OMIT parameters of the OUTFIL statement, except that:

- OUTFIL is processed **after** sorting, copying or merging
- The FORMAT=f parameter cannot be used for OUTFIL

OUTFIL is discussed later in Chapter 7, “Creating multiple output data sets and reports,” on page 83.

You select the records you want included or omitted by either:

1. Comparing the contents of a field with one of the following:

   Another field
   For example, you can select records for which the author’s last name is the same as the instructor’s last name.

   A constant
   The constant can be a character string, a decimal number, a hexadecimal string, or the current date, a future date or a past date. For example, you can select records that have the character string “ HIST” in the department field.
2. Testing a field for “numerics”, “alphanumerics”, “non-numerics”or “non-alphanumerics”. For example, you can select records that have non-numerics in the Employees field or Revenue field, or you can select records that have only uppercase (A-Z) and lowercase (a-z) characters in a specific field.

You can also combine two or more conditions with logical ANDs and ORs. For example, you can select records that have either “HIST” or “PSYCH” in the department field.

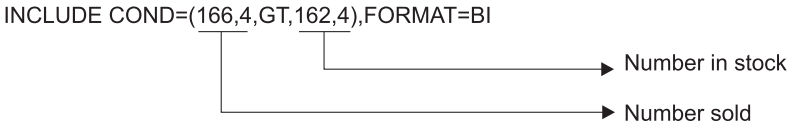
INCLUDE and OMIT both offer powerful substring search capabilities.

In addition, INCLUDE and OMIT allow you to select records based on the results of bit logic tests and two-digit year date comparisons. These two features are not discussed in this document, but details can be found in *z/OS DFSORT Application Programming Guide*.

Writing the INCLUDE statement

Suppose it is the end of the year and you want to sort, by title, only the books that you need to reorder for the coming year. If the number of copies sold this year for a particular book is greater than the number in stock, you can assume you need to order more copies.

An INCLUDE statement that selects only the books you need to order looks like



this:

Here are the steps for writing this INCLUDE statement:

Table 13. Steps to Create the INCLUDE Statement for Books You Need to Order

Step	Action
1	Leave at least one blank and type **INCLUDE**
2	Leave at least one blank and type **COND=**
3	Type, in parentheses, and separated by commas: 1. The location, length, and data format of the number sold field 2. The comparison operator **GT** (comparison operators are shown in Figure 1) for greater than 3. The location, length, and data format of the number in stock field.

You can select from the following comparison operators:

Comparison Operator	Meaning
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

Figure 1. Comparison Operators

You can place the SORT statement either before or after the INCLUDE statement. Control statements do not have to be in any specific order. However, it is good documentation practice to code them in the order in which they are processed. For a flowchart showing the order in which all the control statements are processed, see Appendix C, “Processing order of control statements,” on page 193.

```
INCLUDE COND=(166,4,BI,GT,162,4,BI)
SORT FIELDS=(1,75,CH,A)
```

This sorts the selected subset of the input records by title in ascending order. Table 14 shows the sorted data set.

Table 14. Books for which Number Sold is greater than Number in Stock

Book Title	Number In Stock	Number Sold
1 75	162 165	166 169
ADVANCED TOPICS IN PSYCHOANALYSIS	1	12
COMPUTER LANGUAGES	5	29
COMPUTERS: AN INTRODUCTION	20	26
CRISES OF THE MIDDLE AGES	**14**	**17**
EDITING SOFTWARE MANUALS	13	32
INKLINGS: AN ANTHOLOGY OF YOUNG POETS	**2**	**32**
INTRODUCTION TO BIOLOGY	6	11
MODERN ANTHOLOGY OF WOMEN POETS	**1**	**26**
NUMBERING SYSTEMS	6	27
STRATEGIC MARKETING	3	35
SUPPLYING THE DEMAND	**0**	**32**
SYSTEM PROGRAMMING	4	23
THE COMPLETE PROOFREADER	7	19

Suppose you want to reduce the subset of input records even further, to sort only the books you need to order from COR publishers. In this case, two conditions must be true:

- The number sold is greater than the number in stock.
- The book is published by COR.

To add the second condition, expand the INCLUDE statement by adding a logical AND, and compare the contents of the publisher field to the character string " COR" (see "Writing constants" on page 30 for details how to specify constants). Because the publisher field is 4 bytes long, "COR" will be padded on the right with one blank.

```
INCLUDE COND=(166,4,BI,GT,162,4,BI,AND,106,4,CH,EQ,C'COR')
SORT FIELDS=(1,75,CH,A)
```

Table 15 shows the result.

Table 15. COR Books for which Number Sold is greater than Number in Stock

Book Title	Publisher	Number In Stock	Number Sold
1 75	106 109	162 165	166 169
CRISES OF THE MIDDLE AGES	COR	14	17
INKLINGS: AN ANTHOLOGY OF YOUNG POETS	COR	2	32
MODERN ANTHOLOGY OF WOMEN POETS	COR	1	26
SUPPLYING THE DEMAND	COR	0	32

As another example, you might sort only the books for courses 00032 and 10347 by writing the INCLUDE and SORT statements as follows:

```
INCLUDE COND=(115,5,CH,EQ,C'00032',OR,115,5,CH,EQ,C'10347')
SORT FIELDS=(115,5,CH,A)
```

Note: In the previous example, you cannot substitute C'32' for C'00032', because character constants are padded on the right with blanks. DFSORT uses the following rules for padding and truncation:

Padding
adds fillers in data, usually zeros or blanks

Truncation
deletes or omits a leading or trailing portion of a string

In comparisons, the following rules apply:

- In a field-to-field comparison, the shorter field is padded as appropriate (with blanks or zeros).
- In a field-to-constant comparison, the constant is padded or truncated to the length of the field. Decimal constants are padded or truncated on the left. Character and hexadecimal constants are padded or truncated on the right.

Writing the OMIT statement

Suppose that you want to sort, by title, all the books used for courses but not those for general reading. In this case, you can use an OMIT statement that excludes records containing a blank in the course department field.

The format of the OMIT statement is the same as that of the INCLUDE statement. To exclude the general reading books, write:

```
OMIT COND=(110,5,CH,EQ,C' ')
SORT FIELDS=(1,75,CH,A)
```

Table 16 shows the sorted data set.

Table 16. Sorted Data Set without Books Not Required for Classes

Book Title	Course Department
1 75	110 114
ADVANCED TOPICS IN topCHOANALYSIS	PSYCH
COMPUTER LANGUAGES	COMP
COMPUTERS: AN INTRODUCTION	COMP
CRISES OF THE MIDDLE AGES	HIST
EDITING SOFTWARE MANUALS	ENGL
EIGHTEENTH CENTURY EUROPE	HIST
INKLINGS: AN ANTHOLOGY OF YOUNG POETS	ENGL
INTRODUCTION TO BIOLOGY	BIOL
INTRODUCTION TO PSYCHOLOGY	PSYCH
MODERN ANTHOLOGY OF WOMEN POETS	ENGL
NUMBERING SYSTEMS	COMP
SHORT STORIES AND TALL TALES	ENGL
STRATEGIC MARKETING	BUSIN
SUPPLYING THE DEMAND	BUSIN
SYSTEM PROGRAMMING	COMP
THE COMPLETE PROOFREADER	ENGL
THE INDUSTRIAL REVOLUTION	HIST
VIDEO GAME DESIGN	COMP

Allowable comparisons for INCLUDE and OMIT

Table 17 and Table 18 show the allowable field-to-field and field-to-constant comparisons for the data formats most commonly used with INCLUDE and OMIT. Refer to *z/OS DFSORT Application Programming Guide* for complete details of all of the data formats you can use with INCLUDE and OMIT.

Table 17. Allowable Field-to-Field Comparisons

Field Format	BI	FI	CH	ZD	PD	FS	UFF	SFF
BI	✓		✓					
FI		✓						
CH	✓		✓					
ZD				✓	✓			
PD				✓	✓			
FS						✓	✓	✓
UFF						✓	✓	✓
SFF						✓	✓	✓

Table 18. Allowable Field-to-Constant Comparisons

Field Format	Character String	Hexadecimal String	Decimal Number
BI	✓	✓	✓
FI			✓
CH	✓	✓	
ZD			✓
PD			✓
FS			✓
UFF			✓
SFF			✓

For example, if you want to sort by author's name and include only those books whose author's last name begins with "M," you can compare the contents of byte 76 (the first byte of the author's last name), which is in character format, with either a character or hexadecimal string:

```
INCLUDE COND=(76,1,CH,EQ,C'M')
SORT FIELDS=(76,15,CH,A)
```

or

```
INCLUDE COND=(76,1,CH,EQ,X'4D')
SORT FIELDS=(76,15,CH,A)
```

Also, if you want to sort by number in stock only the books for which the number in stock is less than 10, you can compare the contents of the number in stock field, which is in binary format, to a decimal constant or a hexadecimal string:

```
INCLUDE COND=(162,4,BI,LT,10)
SORT FIELDS=(162,4,BI,A)
```

or

```
INCLUDE COND=(162,4,BI,LT,X'0000000A')
SORT FIELDS=(162,4,BI,A)
```

For the hexadecimal constant, remember the padding and truncation rules. If you specify X'0A', the string is padded on the right instead of the left. For the decimal constant, you can use 10 or +10, and you do not have to worry about padding or truncation.

Writing constants

The formats for writing character strings, hexadecimal strings, and decimal numbers are shown later in this section.

Character strings

The format for writing a character string is:

```
C'x...x'
```

where *x* is an EBCDIC character. For example, C'FERN'.

If you want to include a single apostrophe in the string, you must specify it as two single apostrophes. For example, *O'NEILL* must be specified as C'O''NEILL'.

You can use special keywords to specify a character string for the current date of the run in various forms, as detailed in *z/OS DFSORT Application Programming Guide*. For example, if you want to select records in which a 10-character date in the form C'yyyy/mm/dd' starting in position 42 equals today's date, write:

```
INCLUDE COND=(42,10,CH,EQ,DATE1(/))
```

You can also use special keywords to specify a character string for a future or past date (relative to the current date of the run) in various forms, as detailed in *z/OS DFSORT Application Programming Guide*. For example, if you want to select records in which a 10-character date in the form C'yyyy/mm/dd' starting in position 42 is between 30 days in the past and 30 days in the future, write:

```
INCLUDE COND=(42,10,CH,GE,DATE1(/)-30,AND,42,10,CH,LE,DATE1(/)+30)
```

Hexadecimal strings

The format for writing a hexadecimal string is:

```
X'yy...yy'
```

where *yy* is a pair of hexadecimal digits. For example, X'7FB0'.

Decimal numbers

The format for writing a decimal number is:

```
n...n    or    ±n...n
```

where n is a decimal digit. Examples are 24, +24, and -24.

Decimal numbers must not contain commas or decimal points.

You can use special keywords to specify a decimal number for the current date of the run in various forms, as detailed in *z/OS DFSORT Application Programming Guide*. For example, if you want to select records in which a 4-byte packed decimal date of P'yyyyddd' (hex yyyydddC) starting in position 28 equals today's date, write:

```
INCLUDE COND=(28,4,PD,EQ,DATE3P)
```

You can use special keywords to specify a decimal number for a future or past date (relative to the current date of the run) in various forms, as detailed in *z/OS DFSORT Application Programming Guide*. For example, if you want to select records in which a 4-byte packed decimal date of P'yyyyddd' (hex yyyydddC) starting in position 28 is between 30 days in the past and 30 days in the future, write:

```
INCLUDE COND=(28,4,PD,GE,DATE3P-30,AND,28,4,PD,LE,DATE3P+30)
```

Numeric tests for INCLUDE and OMIT

Suppose you think that some of the values in the Employees field might contain invalid numeric data, and you want to select the records with those values, if any. Each byte of the 4-byte Employees field should contain '0' through '9'; you would consider any other character, such as 'A' or '.' to be invalid. '1234' is a valid numeric value; it contains all numerics. '12.3' is an invalid numeric value; it contains a non-numeric. You can use one of the numeric test capabilities of the INCLUDE statement to collect the records you want as follows:

```
INCLUDE COND=(18,4,FS,NE,NUM)
```

If the value in the field (18,4,FS) is not equal (NE) to numerics (NUM), the record is included. The records in the output data set will be those in which the field has non-numerics (a character other than '0'-'9' appears somewhere in the field).

Use NUM to indicate a test for numerics or non-numerics.

Use EQ to test for numerics, or NE to test for non-numerics.

Use FS format for the field if you want to test for character numerics ('0'-'9' in every byte).

Use ZD format for the field if you want to test for zoned decimal numerics ('0'-'9' in all non-sign bytes; X'F0'-X'F9', X'D0'-X'D9' or X'C0'-X'C9' in the sign byte).

Use PD format for the field if you want to test for packed decimal numerics (0-9 for all digits; F, D or C for the sign).

Here's an INCLUDE statement that only includes records in which the Revenue field and Profit field have packed decimal numerics (that is, there are no invalid packed decimal values in these fields).

```
INCLUDE COND=(22,6,PD,EQ,NUM,AND,28,6,PD,EQ,NUM)
```

Alphanumeric Tests for INCLUDE and OMIT

Testing for alphanumerics or non-alphanumerics is similar to testing for numerics or non-numerics. Use BI for the format. Use EQ to test for alphanumerics or NE to test for non-alphanumerics. Instead of NUM, use one of the following corresponding to the set of alphanumeric characters you need:

- 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)

Here is an INCLUDE statement that only includes records which have uppercase or numeric characters in positions 11 to 15:

```
INCLUDE COND=(11,5,BI,EQ,UN)
```

If every position from 11 to 15 in a record has A-Z or 0-9, that record is included. If any position from 11 to 15 in a record does not have A-Z or 0-9, that record is not included. So a record with 'B03RS' in 11 to 15 would be included, whereas records with 'B03rS' or 'B,ABC' would not be included.

Here's an INCLUDE statement that removes any record that has only A-Z or a-z in positions 1 to 8:

```
INCLUDE COND=(1,8,BI,NE,MC)
```

So a record with 'RsTUVxyz' in 1-8 would not be included, whereas a record with 'Rs$UVxyz' in 1-8 would be included.

Substring search for INCLUDE and OMIT

Suppose you want to select only the books for the Biology, History, Business and Psychology departments. Based on what you learned earlier, you can select those books using this INCLUDE statement:

```
INCLUDE COND=(106,5,CH,EQ,C'BIOL',OR,
              106,5,CH,EQ,C'HIST',OR,
              106,5,CH,EQ,C'BUSIN',OR,
              106,5,CH,EQ,C'PSYCH')
```

But the more departments you want to include, the more typing you have to do. Instead, you can use one of the substring search capabilities of INCLUDE and OMIT to write the statement in a simpler form as:

```
INCLUDE COND=(106,5,SS,EQ,C'BIOL ,HIST ,BUSIN,PSYCH')
```

With substring search (SS format), you only write the field once and write the character constant so it includes all of the strings you want to search for. If the value in the field matches any of the strings (for example, "BUSIN"), the record is included. If the value in the field does not match any of the strings, the record is omitted.

The length of each string must match the length of the field. Because the Department field is 5 characters, you must add a blank at the end of "BIOL" and "HIST", which are each four characters, but not for "BUSIN" and "PSYCH", which are each five characters.

The other way to use substring search is by searching for a constant within a field. For example, if you wanted to select only the books with "INTRODUCTION" in their Title, you could use the following INCLUDE statement:

```
INCLUDE COND=(1,75,SS,EQ,C'INTRODUCTION')
```

The books selected for output would be:

```
COMPUTERS: AN INTRODUCTION
INTRODUCTION TO PSYCHOLOGY
INTRODUCTION TO BIOLOGY
```

VB data set considerations

The same VB data set considerations you learned about previously for the SORT and MERGE statements also apply to the INCLUDE and OMIT statements.

Starting positions

When you code your compare fields for including or omitting VB records, remember to add 4 to the starting position to account for the 4-byte RDW. For example, the following INCLUDE statement compares a PD field in the third through fifth data bytes of a VB record to a PD field in the sixth through eighth bytes of a VB record.

```
INCLUDE COND=(7,3,PD,EQ,10,3,PD)
```

Short control fields

If you know you have VB records with short compare fields, you can specify the VLSCMP option, if appropriate, to prevent DFSORT from terminating. For example:

```
OPTION COPY,VLSCMP
INCLUDE COND=(21,8,CH,EQ,C'Type 200')
```

VLSCMP tells DFSORT that you want to temporarily replace any missing compare field bytes with binary zeros, thus allowing the short fields to be validly compared (the zeros are not kept for the output records). In this example, records less than 28 bytes are not included because the binary zeros added for the missing bytes in the field prevent it from being equal to 'Type 200'.

Another way you can prevent DFSORT from terminating for VB records with short compare fields, if appropriate, is by specifying the VLSHRT option. For example:

```
OPTION COPY,VLSHRT
INCLUDE COND=(21,8,CH,EQ,C'Type 200')
```

VLSHRT tells DFSORT to treat any comparison involving a short field as false. In this example, any records less than 28 bytes are not included.

Attention: If NOVLSCMP and NOVLSHRT are in effect, DFSORT terminates if it finds a short compare field in any VB record.

For more information on DFSORT's VLSCMP and VLSHRT options, see *z/OS DFSORT Application Programming Guide*.

Summary

This chapter covered three ways to select only a subset of the input records to make processing more efficient. You wrote INCLUDE and OMIT statements and learned about allowable comparison operators, various types of constants, numeric tests, and substring search.

Chapter 4. Summing records

Suppose that the English department wants to know the total price of books for all its courses. You can include just the records for the English department by using the INCLUDE statement, and add the book prices together by using the SORT and SUM statements.

On the SUM control statement, you specify one or more **numeric** fields that are to be summed whenever records have equally collating control fields (control fields are specified on the SORT statement). The data formats you can specify on the SUM statement are binary (BI), fixed-point (FI), packed decimal (PD), zoned decimal (ZD) and floating-point (FL).

To sum the prices for just the records for the English department, specify the price field on the SUM statement and the department field on the SORT statement. The INCLUDE statement selects just the records for the English department before SUM and SORT are processed, making the department field equal for all of the included records, and allowing the prices to be summed. (For a flowchart showing the order in which the INCLUDE, SUM, and SORT statements are processed, see Appendix C, “Processing order of control statements,” on page 193.)

When you sum records, keep in mind that two types of fields are involved:

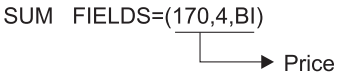
Control fields
specified on the SORT statement

Summary fields
specified on the SUM statement

The contents of the summary fields are summed for groups of records with the same control fields (often called "duplicate" records).

Writing the SUM statement

A SUM statement that sums the prices would look like this:



Here are the steps for writing this SUM statement:

Table 19. Steps to Create the SUM Statement for Prices

Step	Action
1	Leave at least one blank and type **SUM**
2	Leave at least one blank and type **FIELDS=**
3	Type, in parentheses and separated by commas, the location, length, and data format of the price field.

The INCLUDE, SORT, and SUM statements are as follows:

```
INCLUDE COND=(110,5,CH,EQ,C'ENGL')
SORT FIELDS=(110,5,CH,A)
SUM FIELDS=(170,4,BI)
```

When the prices are summed, the final sum appears in the price field of one record, and the other records are deleted. Therefore, the result (shown in Table 20) is only one record, containing the sum. You can control which record appears if you specify that records keep their original order. For the examples, the default is for records with equally collating control fields to appear in their original order (EQUALS in effect). When summing records keeping the original order, DFSORT chooses the first record to contain the sum.

Table 20. Sum of Prices for English Department

Book Title	Course Department	Price
1 75	110 114	170 173
INKLINGS: AN ANTHOLOGY OF YOUNG POETS[1]	ENGL[2]	4640[3]

Note:

[1] Some of the fields in your summation record might not be meaningful, such as the book title field in Table 20. In the next chapter, you will learn two ways to leave out fields that are not meaningful.

[2] Specified as a control field.

[3] Specified as a summary field.

Suppose now that the English department wants to know the total price of books for *each* of its courses. In this case, you still select only the English department's records using INCLUDE, and specify the price field on the SUM statement, but you specify the *course number* on the SORT statement.

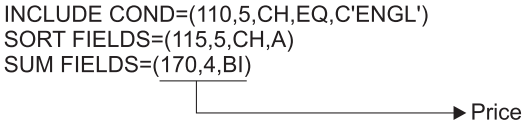


Table 21 shows the result, one record per course.

Table 21. Sum of Prices for English Department

Book Title	Course Number	Price
1 75	115 119	170 173
SHORT STORIES AND TALL TALES	10054	1520
EDITING SOFTWARE MANUALS	10347	2075
INKLINGS: AN ANTHOLOGY OF YOUNG POETS	10856	1045

For an example using two summary fields, assume that for inventory purposes you want to sum separately the number of books in stock and the number sold for each of the publishers.

For this application, specify the publisher as the control field on the SORT statement and the number in stock and number sold as summary fields on the SUM statement. You want to use all of the records in the input data set this time,

so you don't need to code an INCLUDE or OMIT statement. (SORT and SUM can be used with or without an INCLUDE or OMIT statement.)

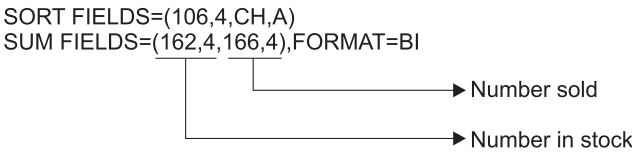


Table 22 shows the result, one record per publisher.

Table 22. Sum of Number in Stock and Number Sold for Each Publisher

Book Title 1 75	Publisher 106 109	Number In Stock 162 165	Number Sold 166 169
LIVING WELL ON A SMALL BUDGET	COR	103	161
COMPUTER LANGUAGES	FERN	19	87
VIDEO GAME DESIGN	VALD	42	97
COMPUTERS: AN INTRODUCTION	WETH	62	79

Suppressing records with duplicate control fields

Apart from summing values, you can also use SUM to delete records with duplicate control fields (often called "duplicate records").

For example, you might want to list the publishers in ascending order, with each publisher appearing only once. If you use only the SORT statement, COR appears seven times (because seven books in the file are published by COR), FERN appears four times, VALD five times, and WETH four times.

By specifying FIELDS=NONE on the SUM statement, DFSORT writes only one record per publisher, as follows:

```
SORT FIELDS=(106,4,CH,A)
SUM FIELDS=NONE
```

Table 23 shows the result.

Table 23. List of Publishers, Deleting Duplicates

Book Title 1 75	Publisher 106 109
LIVING WELL ON A SMALL BUDGET	COR
COMPUTER LANGUAGES	FERN
VIDEO GAME DESIGN	VALD
COMPUTERS: AN INTRODUCTION	WETH

In Chapter 12, “Using the ICETOOL utility,” on page 135, you will learn how to use ICETOOL's powerful SELECT, OCCUR and SPLICE operators to perform many more functions involving duplicate and non-duplicate records.

Handling overflow

When a sum becomes larger than the space available for it, *overflow* occurs. For example, if a 2-byte binary field (unsigned) contains X' FFFF' and you add X'0001' to it, overflow occurs, because the sum requires more than two bytes.

```
  FFFF
  0001
 10000
```

If overflow occurs, the summary fields in the two records involved are not added together. That is, the records are kept unchanged; neither record is deleted.

In some cases, you can correct overflow by padding the summary fields with zeros, using the INREC control statement. “Preventing overflow when summing values” on page 71 shows you how to do this.

VB data set considerations

The same VB data set considerations you learned about previously for the SORT, MERGE, INCLUDE and OMIT statements also apply to the SUM statement.

Starting positions

When you code your summary fields for VB records, remember to add 4 to the starting position to account for the 4-byte RDW. For example, the following SUM statement specifies a PD summary field in the third through fifth data bytes of a VB record:

```
SUM FIELDS=(7,3,PD)
```

Short summary fields

If you know you have VB records with short summary fields, you can specify the VLSHRT option, if appropriate, to prevent DFSORT from terminating. For example:

```
OPTION VLSHRT
SORT FIELDS=(6,2,CH,A)
SUM FIELDS=(21,8,ZD)
```

VLSHRT tells DFSORT to leave records with short summary fields unsummed. That is, when one of the two records involved in a summary operation has a short summary field, the records are kept unchanged; neither record is deleted.

Attention: If NOVLSHRT is in effect, DFSORT terminates if it finds a short summary field in any VB record.

For more information on DFSORT's VLSHRT option, see *z/OS DFSORT Application Programming Guide*.

Summary
This chapter covered summing records in your data set. It explained how to use the SUM statement to sum records with equal control fields, and how to suppress any records with duplicate control fields. Now, you continue with tutorials about using OUTREC and INREC to reformat your data sets.