

z/OS
3.2

DFSORT: Getting Started



Note

Before using this information and the product it supports, read the information in [“Notices” on page 181.](#)

This edition applies to IBM® z/OS® 3.2 (5655-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2025-09-30

© **Copyright International Business Machines Corporation 1983, 2025.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures.....	ix
Tables.....	xi
About this document.....	xv
How to use this document.....	xv
Required product knowledge.....	xv
How to provide feedback to IBM.....	xvii
Summary of changes.....	xix
Summary of changes for z/OS 3.2.....	xix
Summary of changes for z/OS 3.1.....	xix
Part 1. Introduction.....	1
Chapter 1. What is DFSORT?.....	3
DFSORT on the World Wide Web.....	3
Data sets, records and fields.....	3
Sorting data sets.....	4
Merging data sets.....	4
Copying data sets.....	5
Joining data sets.....	5
What else can you do with DFSORT?.....	5
Creating and running DFSORT jobs.....	6
Writing jobs.....	6
Summary of DFSORT control statements.....	6
Running jobs.....	7
Creating and using the sample data sets.....	7
Part 2. Learning to write JCL and DFSORT control statements.....	9
Chapter 2. Sorting, merging, and copying data sets.....	11
Sorting data sets.....	11
Sorting by multiple fields.....	13
FIELDS.....	15
Example: Sorting Character data and ASCII Unsigned free format numeric data	15
Example: Merging ASCII Signed free format numeric data	15
Continuing a statement.....	15
Comment statements.....	15
JCL for sorting data sets directly.....	16
Merging data sets.....	17
Writing the MERGE control statement.....	18
JCL for merging data sets directly.....	19
FIELDS.....	20
FORMAT.....	20
VB data set considerations.....	20
Starting positions.....	20
Short control fields.....	20

Copying data sets.....	21
Specifying COPY on the SORT, MERGE, or OPTION statement.....	21
JCL for copying data sets directly.....	21
Chapter 3. Including or omitting records.....	23
Writing the INCLUDE statement.....	23
Writing the OMIT statement.....	26
Allowable comparisons for INCLUDE and OMIT.....	26
Writing constants.....	28
Character strings.....	28
Hexadecimal strings.....	28
Decimal numbers.....	28
Numeric tests for INCLUDE and OMIT.....	29
Alphanumeric Tests for INCLUDE and OMIT.....	29
Substring search for INCLUDE and OMIT.....	30
Regular expressions.....	30
Unicode comparisons.....	31
VB data set considerations.....	32
Starting positions.....	32
Short control fields.....	32
Summary.....	32
Chapter 4. Summing records.....	33
Writing the SUM statement.....	33
Suppressing records with duplicate control fields.....	35
Handling overflow.....	36
VB data set considerations.....	36
Starting positions.....	36
Short summary fields.....	36
Chapter 5. Reformatting records with fixed fields.....	37
Reformatting records after sorting with BUILD or FIELDS.....	38
Reordering fields to reserve space.....	39
Inserting binary zeros.....	39
Inserting blanks.....	40
Inserting strings.....	41
Character strings.....	41
Hexadecimal strings.....	42
Setting up a basic report.....	42
Translating uppercase to lowercase, EBCDIC to ASCII, and more.....	43
Converting numeric fields to different formats.....	44
Editing numeric fields.....	45
Displaying data in hexadecimal.....	48
Displaying data as bits.....	48
Performing arithmetic with numeric fields and constants.....	48
Converting date fields.....	49
Performing arithmetic with date fields.....	50
Doing lookup and change.....	51
Left-justifying and right-justifying data.....	52
Left-squeezing and right-squeezing data.....	54
Reformatting records with OVERLAY.....	58
Extending records with OVERLAY.....	59
Reformatting records with FINDREP.....	60
Reformatting records with IFTHEN.....	61
Reformatting records before sorting.....	65
Using other statements with INREC.....	65
Preventing overflow when summing values.....	66
Inserting sequence numbers.....	67

VB data set considerations.....	69
RDW.....	69
Starting positions and columns.....	70
Variable data.....	70
Summary.....	71
Chapter 6. Reformatting records with variable fields.....	73
Using %nnn, %nn and %n parsed fields with BUILD and OVERLAY.....	73
Using %nnn, %nn and %n Parsed Fields with IFTHEN.....	74
Where you can use %nnn, %nn and %n fields in BUILD and OVERLAY.....	76
PARSE parameters.....	76
Chapter 7. Creating multiple output data sets and reports.....	79
Creating multiple identical copies.....	79
Selecting and sampling by relative record number.....	80
Including, omitting, and saving discards.....	82
Reformatting.....	84
Repeating.....	85
Splitting.....	86
Creating reports: OUTFIL vs ICETOOL.....	88
Creating reports with OUTFIL.....	88
Data.....	88
Headers.....	89
Trailers and statistics.....	91
No data or carriage control characters.....	93
VB data set considerations for headers and trailers.....	93
Sections.....	94
Updating counts and totals in trailer with OUTFIL.....	96
Converting FB to VB.....	98
Converting VB to FB.....	99
Chapter 8. Joining records.....	101
Chapter 9. Calling DFSORT from a program.....	109
Passing control statements.....	109
Calling DFSORT from a COBOL program.....	109
Sorting records.....	109
Merging records.....	111
Sorting with COBOL FASTSRT.....	112
Calling DFSORT from a PL/I program.....	113
Chapter 10. Overriding installation defaults	115
Specifying PARM parameters on a JCL EXEC statement.....	115
Specifying an OPTION control statement in DFSPARM.....	116
Chapter 11. Using DFSORT efficiently.....	117
Be generous with main storage.....	117
Allow memory object sorting, Hipersorting and dataspace sorting.....	117
Use high-speed disks for work data sets.....	117
Eliminate unnecessary fields with INREC.....	118
Eliminate unnecessary records with INCLUDE or OMIT.....	118
Eliminate unnecessary records with STOPAFT and SKIPREC.....	118
Consolidate records with SUM.....	118
Create multiple output data sets with OUTFIL.....	119
Replace program logic with DFSORT control statements.....	119
Use FASTSRT with COBOL.....	119
Avoid options that might degrade performance.....	119

Part 3. Learning to use ICETOOL.....	121
Chapter 12. Using the ICETOOL utility.....	123
ICETOOL operators.....	123
Sample input data sets.....	124
Writing required JCL statements.....	124
ICETOOL comment and blank statements.....	125
Printing statistics for numeric fields.....	126
Continuing an operator statement.....	127
Statistics for VB data set record lengths.....	127
Creating identical sorted data sets.....	128
Creating different subsets of a sorted data set.....	130
Creating multiple unsorted data sets.....	133
Counting values in a range.....	133
Printing simple reports.....	135
Printing tailored reports.....	136
Using formatting items.....	138
Edit masks.....	138
Number of digits.....	140
Leading zeros.....	140
Edit patterns.....	140
No statistics.....	141
Division.....	141
Leading, floating and trailing characters.....	142
Printing sectioned reports.....	142
Printing how many times fields occur.....	144
Selecting records by field occurrences.....	146
Joining fields from different data sets.....	150
Matching records from different data sets.....	152
Sorting records between headers and trailers.....	155
Keeping or removing headers, trailers and relative records.....	156
Merging previously sorted data sets.....	157
Complete ICETOOL job and TOOLMSG output.....	157
Part 4. Learning to use symbols.....	163
Chapter 13. Defining and using symbols.....	165
Creating the SYMNAMES data set.....	165
Defining symbols for fields.....	165
Using symbols for fields in DFSORT statements.....	166
Using symbols for fields in ICETOOL operators.....	167
Defining and using symbols for constants.....	168
Appendix A. Creating the sample data sets.....	171
Appendix B. Descriptions of the sample data sets.....	173
Appendix C. Processing order of control statements.....	177
Appendix D. Accessibility.....	179
Notices.....	181
Terms and conditions for product documentation.....	182
IBM Online Privacy Statement.....	183
Policy for unsupported hardware.....	183

Minimum supported hardware.....	183
Programming interface information.....	184
Trademarks.....	184
Index.....	185

Figures

1. Comparison Operators..... 24

2. Sample COBOL Program with MERGE Commands.....112

3. Sample PL/I Program with SORT Commands..... 114

4. Complete TOOLMSG Data Set. (Part 1 of 2)..... 160

5. Complete TOOLMSG Data Set. (Part 2 of 2)..... 161

6. Processing Order of Control Statements.....177

Tables

1. Related documents.....	xvi
2. DFSORT arranges information in ascending and descending order.....	4
3. Commonly used data formats.....	4
4. DFSORT merges two data sets into one data set.....	5
5. Sample bookstore data set sorted by course department in ascending order.....	8
6. Steps to Create the SORT Statement to Sort by Department.....	11
7. Sample Bookstore Data Set Sorted by Course Department in Ascending Order.....	12
8. Sample Bookstore Data Set Sorted by Price in Descending Order.....	12
9. Sample Bookstore Data Set Sorted by Multiple Fields.....	14
10. Sample Bookstore Data Set Sorted by Course Department and Book Title.....	17
11. Five New Records Sorted by Course Department and Book Title.....	17
12. Sample Bookstore Data Set Merged with Five New Records.....	18
13. Steps to Create the INCLUDE Statement for Books You Need to Order.....	24
14. Books for which Number Sold is greater than Number in Stock.....	25
15. COR Books for which Number Sold is greater than Number in Stock.....	25
16. Sorted Data Set without Books Not Required for Classes.....	26
17. Allowable Field-to-Field Comparisons.....	27
18. Allowable Field-to-Constant Comparisons.....	27
19. Steps to Create the SUM Statement for Prices.....	33
20. Sum of Prices for English Department.....	34
21. Sum of Prices for English Department.....	34
22. Sum of Number in Stock and Number Sold for Each Publisher.....	35
23. List of Publishers, Deleting Duplicates.....	35

24. Steps to Create the OUTREC Statement for Reformatting Records.....	38
25. Writing Only Publisher, Number In Stock, and Number Sold Fields.....	38
26. Reordering the Fields.....	39
27. Inserting Binary Zeros.....	40
28. Output after inserting blanks.....	40
29. Output of a Report.....	43
30. Converting from BI to ZD.....	44
31. Converting from BI to FS.....	45
32. Input records with PD values.....	45
33. Edit Mask Patterns.....	46
34. Books with Course Department and Price Changes.....	58
35. New and old prices.....	59
36. Proposed discounts for books.....	62
37. Using INREC to Write Only Publisher, Number in Stock, and Number Sold.....	65
38. Padding summary fields (example 1).....	66
39. Padding summary fields (example 2).....	67
40. Total book prices by course.....	67
41. Total Book Prices by Course with Sequence Numbers.....	68
42. Creating the UTFIL Statement for the Multiple Output Data Set Job.....	80
43. Relative Records Numbers for Copy.....	81
44. Relative Records Numbers for Sort.....	81
45. Output records in ENGLOUT data set.....	82
46. Output records in HISTOUT data set.....	83
47. Output records in PSYCHOUT data set.....	83
48. Output records in RESTOUT data set.....	83

49. REGION.IN1 data set for JOINKEYS application.....	101
50. REGION.IN2 data set for JOINKEYS application.....	101
51. REGION.OUT data set for JOINKEYS application.....	101
52. REGION.OUT data set Sorted by Headquarters and Office.....	103
53. CITIES.IN1 data set.....	103
54. CITIES.IN2 data set.....	104
55. CITIES.OUT data set.....	104
56. Joined City records.....	105
57. BOTH.OUT data set.....	106
58. F1ONLY.OUT data set.....	107
59. F2ONLY.OUT data set.....	107
60. Control statement and corresponding records with INREC.....	118
61. Steps to Create a Blank Statement and a Comment Statement.....	125
62. Steps to Create the STATS Operator.....	126
63. Steps to Create the SORT Operator.....	128
64. Steps to Create JCL Statements for the SORT Operator.....	129
65. Books from publishers VALD and WETH.....	130
66. Steps to Create the SORT Operator.....	131
67. Steps to Create JCL Statements for the SORT Operator.....	131
68. Records for California Sorted by City.....	132
69. Records for Colorado Sorted by City.....	132
70. Edit Mask Patterns.....	138
71. Books from Publishers with More than Four Books in Use.....	148
72. REGION.IN1 data set for join.....	150
73. REGION.IN2 data set for join.....	150

74. REGION.OUT data set for join.....	151
75. T1 data set fields from REGION.IN1.....	152
76. T1 data set fields from REGION.IN2.....	152
77. T1 data set fields from SORT.SAMPIN.....	153
78. T1 data set fields from SORT.SAMPADD.....	153
79. COURSE.MATCH output.....	154
80. COURSE.INONLY output.....	154
81. COURSE.ADDONLY output.....	154
82. Steps to Define Symbols for Fields.....	166
83. Steps to define symbols for constants.....	169
84. SORT.SAMPIN and SORT.SAMPADD Field Descriptions.....	173
85. SORT.BRANCH Field Descriptions.....	173

About this document

z/OS DFSORT: Getting Started is a user's guide and tutorial for DFSORT (Data Facility Sort). You should read it if you are not familiar with DFSORT and would like to learn the many ways you can use DFSORT and DFSORT's ICETOOL utility to process data sets. Both new and experienced DFSORT users can use this document as a general guide to the many features available with DFSORT and ICETOOL. This document introduces you to the JCL, control statements and features of DFSORT and ICETOOL with numerous examples. *z/OS DFSORT: Getting Started* can help you get the most out of *z/OS DFSORT Application Programming Guide*, which has the complete details on all of the topics introduced in this document.

The chapters in this document assume that you have used job control language (JCL) and understand how to work with data sets. You should also know what data sets are available at your site.

How to use this document

This document gives you all of the information and instructions you need to build and submit DFSORT jobs. You use DFSORT by writing JCL and DFSORT program control statements.

New users should work through *z/OS DFSORT: Getting Started* from cover to cover. Each task explained in this document builds on knowledge gained in previous tasks. The Table of Contents lists the main tasks, and summaries are included in the chapters. If you have previous experience with these tasks, you can proceed from here directly to the tutorials that begin in [Chapter 2, "Sorting, merging, and copying data sets,"](#) on page 11.

[Chapter 1, "What is DFSORT?,"](#) on page 3 is an overview of the basic principles of sorting, merging, and copying, and explains how to create and use the sample data sets for the examples in this document.

[Chapter 2, "Sorting, merging, and copying data sets,"](#) on page 11 through [Chapter 11, "Using DFSORT efficiently,"](#) on page 117 show you how to create and process DFSORT jobs by writing JCL and DFSORT program control statements (for example, SORT, MERGE, OPTION, INCLUDE, OMIT, INREC, OUTREC, SUM, OUTFIL, JOINKEYS, JOIN and REFORMAT) to create sorted, merged or copied output data sets while performing various operations on records such as subsetting, reformatting, summing, reporting, and joining.

[Chapter 12, "Using the ICETOOL utility,"](#) on page 123 shows you how to create and process ICETOOL jobs by writing JCL and ICETOOL statements. ICETOOL is a multipurpose DFSORT utility that uses the capabilities of DFSORT to perform multiple operations on one or more data sets in a single step. ICETOOL's 17 operators (COPY, COUNT, DATASORT, DEFAULTS, DISPLAY, MERGE, MODE, OCCUR, RANGE, RESIZE, SELECT, SORT, SPLICE, STATS, SUBSET, UNIQUE and VERIFY) expand DFSORT's capabilities significantly.

[Chapter 13, "Defining and using symbols,"](#) on page 165 shows you how to define symbols for fields and constants and use them in your DFSORT control statements and ICETOOL operators.

Several appendixes and an index follow the chapters.

Required product knowledge

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

- Job control language (JCL)
- Data management

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

<i>Table 1. Related documents</i>
Document
<i>z/OS DFSORT Application Programming Guide</i>
<i>z/OS MVS JCL Reference</i>
<i>z/OS MVS JCL User's Guide</i>
<i>z/OS DFSMS Using Data Sets</i>

[*z/OS DFSORT: Getting Started*](#) is a part of a more extensive DFSORT library. These documents can help you work with DFSORT more effectively.

These documents can help you work with DFSORT more effectively.	
Task	Publication Title
Application Programming	<i>z/OS DFSORT Application Programming Guide</i>
Planning For and Customizing DFSORT	<i>z/OS DFSORT Installation and Customization</i>
Diagnosing Failures and Interpreting Messages	<i>z/OS DFSORT Messages, Codes and Diagnosis Guide</i>
Tuning DFSORT	<i>z/OS DFSORT Tuning Guide</i>

How to provide feedback to IBM

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. For more information, see [How to send feedback to IBM](#).

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Note: IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under [IBM z/OS Product Documentation Update Policy](http://www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy) (www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy).

Summary of changes for z/OS 3.2

The following content is new, changed, or no longer included in z/OS 3.2.

New

The following content is new.

September 2025 release

- None.

Changed

The following content is changed.

September 2025 release

- None.

Deleted

The following content is deleted.

September 2025 release

- None.

Summary of changes for z/OS 3.1

The following content is new, changed, or no longer included in z/OS 3.1.

New

The following content is new.

September 2023 release

- None.

Changed

The following content is changed.

October 2024 refresh

- Pointers to examples are added in [“Where you can use %nnn, %nn and %n fields in BUILD and OVERLAY”](#) on page 76.

September 2023 release

- None.

Deleted

The following content was deleted.

September 2023 release

- Removed all references to the DFSORT FTP site.

Part 1. Introduction

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](http://www.ibm.com/storage/dfsor) (www.ibm.com/storage/dfsor).

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'
```

What is DFSORT?

```
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 on [page 4](#) 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 4](#) 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.

What is DFSORT?

- 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 173](#). 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 173](#), 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 4](#)) 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 8](#) 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 on page 8](#), 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.

What is DFSORT?

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 on page 8 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

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, you learned 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 109.

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.

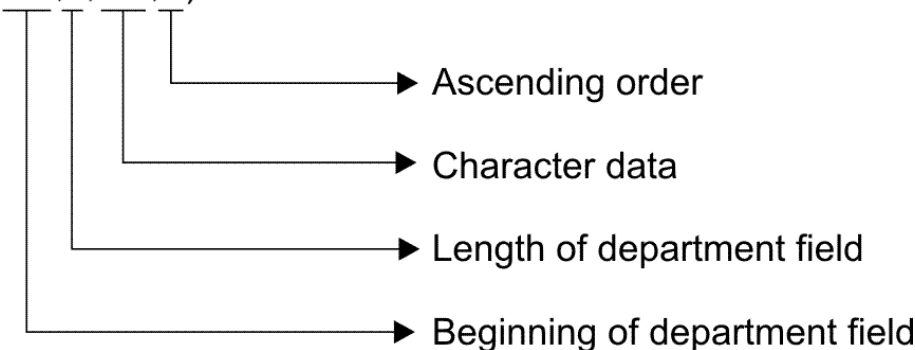
You can use SORT with all of the other DFSORT control statements.

A SORT statement that sorts the bookstore records by the course department field (as shown in [Table 7](#) on page 12) looks like this:

1 2

71 80

SORT FIELDS=(110,5,CH,A)



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=
3	Type, in parenthesis and separated by commas: <ol style="list-style-type: none">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 4 shows the codes for the most commonly used data formats.)4. The letter A, for ascending order.

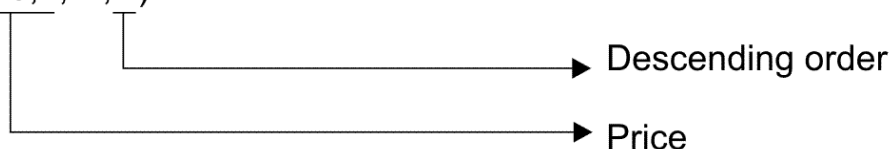
Remember that although [Table 7 on page 12](#) shows only certain fields, the displayed fields are not the only ones in the output data set. Your output data set will more closely resemble the fold-out of the sample bookstore data set.

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:

SORT FIELDS=(170,4,BI,D)



The sort order is bytes 170 through 173 as binary data in descending sequence. [Table 8 on page 12](#) 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

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

Book Title	Price
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 14 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:

SORT FIELDS= (110,5,CH,A,115,5,CH,A,145,15,CH,A,160,2,CH,A,1,75,CH,A)

Book title

Instructor's initials

Instructor's last name

Course number

Department

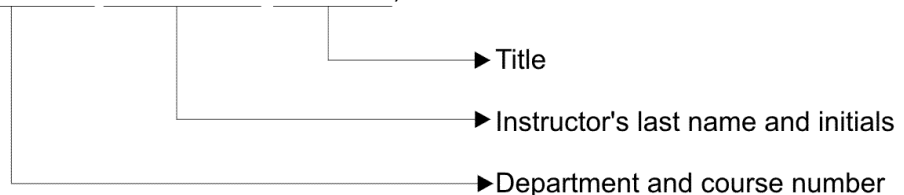
The records are sorted as shown in [Table 9 on page 14](#).

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.

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



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.

Example: Sorting Character data and ASCII Unsigned free format numeric data

```
SORT FIELDS=(10,16,CH,A,40,8,AUF,D)
```

FIELDS

The first four values describe the major control field. It begins on byte 10 of each record, is 16 bytes long, contains character (EBCDIC) data, and is to be sorted in ascending order.

The next four values describe the second control field. It begins on byte 40, is 8 bytes long, contains an ASCII Unsigned free numeric format (AUF), and is to be sorted in descending order.

Example: Merging ASCII Signed free format numeric data

```
MERGE FIELDS=(31,24,A),FORMAT=ASF
```

FIELDS

The first three values describe the major control field. It begins on byte 31 of each record, is 24 bytes long, and is to be merged in ascending order.

FORMAT

FORMAT=ASF is used to supply an ASCII signed free format numeric data format for the p,m,s fields and is equivalent to specifying p,m,ASF,s for these fields.

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 109](#).

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 on page 17](#)), 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 on page 17](#).

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

Book Title		Course Department
1	75	110 114

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

Book Title	Course Department
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:

```
MERGE FORMAT=CH, FIELDS=(110,5,A,1,75,A)
```

```

graph LR
    A["110,5,A,1,75,A"] --> B["Title"]
    A --> C["Department"]
  
```

Table 12 on page 18 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 SORTINnn 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 109, 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 topic, 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 23](#) 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 79.

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.

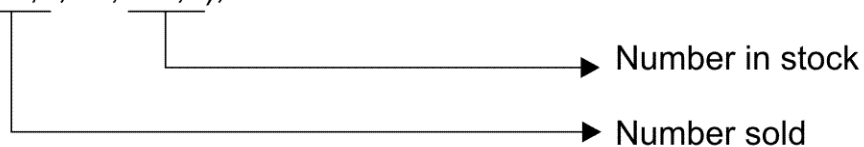
In addition, INCLUDE and OMIT allow you to select records based on the results of bit logic tests and two-digit year date comparisons. These two features are not discussed in this document, but details can be found in [z/OS DFSORT Application Programming Guide](#).

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:

```
INCLUDE COND=(166,4,GT,162,4),FORMAT=BI
```



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: <ol style="list-style-type: none"> 1. The location, length, and data format of the number sold field 2. The comparison operator GT (comparison operators are shown in Figure 1 on page 24) 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
RE	Regular expressions
REH	Regular expressions with hexadecimal string

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 177.

```
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 on page 25](#) 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 28 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 on page 25](#) 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 on page 26 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 on page 27 and Table 18 on page 27 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	AUF	ASF
BI	✓		✓							
FI		✓								
CH	✓		✓							
ZD				✓	✓					
PD				✓	✓					
FS						✓	✓	✓	✓	✓
UFF						✓	✓	✓	✓	✓
SFF						✓	✓	✓	✓	✓
AUF						✓	✓	✓	✓	✓
ASF						✓	✓	✓	✓	✓

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
```

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.

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 exactly with "ibm".

Example 1: Include all records that contain string 'ER' (case-insensitive) 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 ALBUQUERQUE NEW 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 the above job is the following:

```
HOLMER CA 90201
JERRY 10029 NEW YORK
```

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.

The following are examples of Unicode comparisons:

```
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.

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 177.)

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:

SUM FIELDS=(170,4,BI)



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)
```

Summing Records

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 on page 34) 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 on page 34. 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.

INCLUDE COND=(110,5,CH,EQ,C'ENGL')
SORT FIELDS=(115,5,CH,A)
SUM FIELDS=(170,4,BI)




Table 21 on page 34 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.)

```

SORT FIELDS=(106,4,CH,A)
SUM FIELDS=(162,4,166,4),FORMAT=BI

```

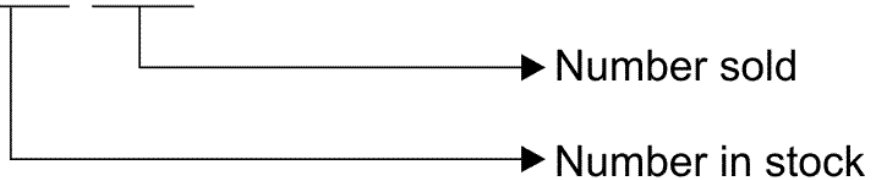


Table 22 on page 35 shows the result, one record per publisher.

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

Book Title		Publisher	Number In Stock	Number Sold
1	75	106 109	162 165	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 on page 35 shows the result.

Table 23. List of Publishers, Deleting Duplicates

Book Title		Publisher
1	75	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 123, 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 66 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.

Chapter 5. Reformatting records with fixed fields

You can reformat records in your data sets by using the INREC, OUTREC, and OUTFIL control statements. You can use these statements separately or together.

This chapter describes how you can reformat records with **fixed fields**, that is, fields that start in the same position and have the same length in every record. The next chapter describes how you can reformat records in similar ways with **variable fields**, that is, fields that have different starting positions and lengths in different records, such as comma separated values (CSV).

With INREC, OUTREC, and OUTFIL, you can perform a wide variety of tasks while sorting, copying, or merging, including the following:

- Delete fields.
- Reorder fields.
- Insert separators (blanks, zeros, strings, current date, future date, past date, and current time).
- Translate data from uppercase to lowercase, lowercase to uppercase, EBCDIC to ASCII and ASCII to EBCDIC.
- Translate data to hexadecimal and bit representation.
- Convert numeric values to many different types of printable formats with or without thousands separators, decimal point, leading or suppressed zeros, and signs.
- Convert numeric values from one format to another format.
- Perform arithmetic using numeric values and decimal constants.
- Convert date fields from one type to another type.
- Perform arithmetic using date fields.
- Change values using a lookup table.
- Replace or remove data using find and replace operations.
- Insert sequence numbers.
- Left-justify data, right-justify data, left-squeeze data, and right-squeeze data.
- Perform various operations on groups of records.

You can reformat records in one of the following three ways:

- **BUILD:** 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 records and the order in which they appear. You can delete, rearrange and insert fields and constants.

Note: For INREC and OUTREC, you can use either the BUILD parameter or the FIELDS parameter. For OUTFIL, you can use either the BUILD parameter or the OUTREC parameter.
- **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.
- **FINDREP:** Reformat each record by doing various types of find and replace operations.
- **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.

The INREC, OUTREC, and OUTFIL statements can all perform the same functions. However:

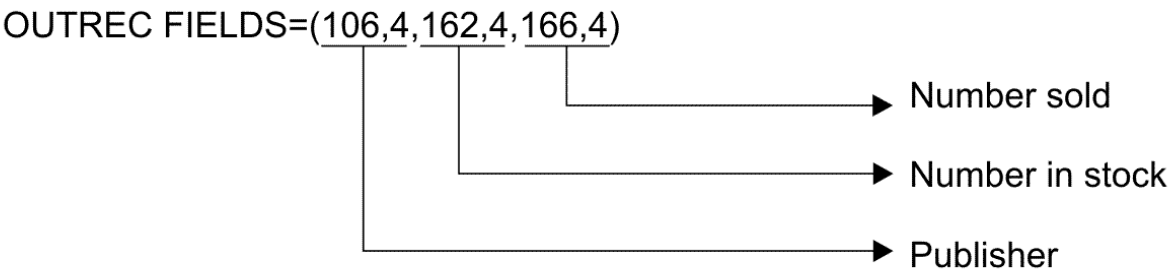
- The INREC statement reformats records **before** they are sorted, copied or merged. See [“Using other statements with INREC” on page 65](#) for information about how INREC affects other control statements.
- The OUTREC statement reformats records **after** they are sorted, copied or merged.

- The OUTFIL statement can reformat records for OUTFIL output data sets **after** they are sorted, copied or merged. Different reformatting parameters can be used for different OUTFIL data sets. The information presented in this Chapter for the INREC and OUTREC statements also applies to the OUTFIL statement. OUTFIL is discussed later in [Chapter 7, “Creating multiple output data sets and reports,”](#) on [page 79](#).

Reformatting records after sorting with BUILD or FIELDS

In the last chapter, you used the SUM statement to sum the price of the books in stock and the books sold for each publisher. Now, using the FIELDS or BUILD parameter of the OUTREC statement, you can delete all the fields that are not needed for the application; in other words, fields whose contents are not meaningful in a summation record. Only the publisher, number-in-stock, and number-sold fields are written, reducing the output record length to 12 bytes.

The OUTREC statement looks like this:



Here are the steps for writing this OUTREC statement:

Table 24. Steps to Create the OUTREC Statement for Reformatting Records	
Step	Action
1	Leave at least one blank, and type OUTREC
2	Leave at least one blank, and type FIELDS= (or BUILD=)
3	Type, in parentheses, and separated by commas: 1. The location and length of the publisher field 2. The location and length of the number in stock field 3. The location and length of the number sold field.

The SORT, SUM and OUTREC statements are as follows:

```
SORT FIELDS=(106,4,CH,A)
SUM FIELDS=(162,4,BI,166,4,BI)
OUTREC FIELDS=(106,4,162,4,166,4)
```

[Table 25 on page 38](#) shows the output.

Table 25. Writing Only Publisher, Number In Stock, and Number Sold Fields		
Publisher	Number In Stock	Number Sold
1 4	5 8	9 12

Table 25. Writing Only Publisher, Number In Stock, and Number Sold Fields (continued)

Publisher	Number In Stock	Number Sold
COR	103	161
FERN	19	87
VALD	42	97
WETH	62	79

The number in stock and number sold fields are binary values which would actually be unreadable if you printed or displayed the output records shown in [Table 25 on page 38](#). Later in this chapter, you'll learn how binary and other unreadable numeric values can be converted to various readable forms.

Suppose you use this SORTOUT DD statement for your output data set:

```
//SORTOUT DD DSN=MY.OUTPUT,DISP=(NEW,CATLG,DELETE),
// UNIT=SYSDA,SPACE=(CYL,(5,5))
```

DFSORT automatically sets the LRECL of your new output data set (MY.OUTPUT) to the reformatted output record length, so LRECL=12 would be set in this case. You do not need to specify the LRECL on your SORTOUT DD statement, because DFSORT will set it appropriately for you.

If your SORTOUT data set already has an LRECL value (OLD data set), or you choose to specify an LRECL value on the SORTOUT DD statement, that value overrides the value DFSORT would set for the LRECL. This can result, intentionally or unintentionally, in padding or truncating your output records if the calculated length of the reformatted record does not match your specified LRECL.

In general, it is best to let DFSORT set the RECFM, LRECL and BLKSIZE for your NEW SORTOUT data sets, unless you have a particular reason to override these values. For OLD SORTOUT data sets, ensure that the existing RECFM, LRECL and BLKSIZE values are appropriate for the control statements you use.

Reordering fields to reserve space

The fields always appear in the order in which you specify them. Therefore, if you want the number sold to appear before the number in stock, as shown in [Table 26 on page 39](#), you reverse their order on the OUTREC statement.

```
SORT FIELDS=(106,4,CH,A)
SUM FIELDS=(162,4,BI,166,4,BI)
OUTREC FIELDS=(106,4,166,4,162,4)
```

Table 26. Reordering the Fields

Publisher	Number Sold	Number In Stock
1 4	5 8	9 12
COR	161	103
FERN	87	19
VALD	97	42
WETH	79	62

Inserting binary zeros

Building on the last example, assume you want to reformat the records to include a new 4-byte binary field after the number in stock (beginning at byte 13). In this case, you can insert binary zeros as place holders for the new field (to be filled in with data at a later date). You can use Z or 1Z to specify a single binary zero. You can use nZ to specify n binary zeros.

Reformatting Records

To insert four binary zeros, write 4Z after the last field:

```
SORT FIELDS=(106,4,CH,A)
SUM FIELDS=(162,4,BI,166,4,BI)
OUTREC FIELDS=(106,4,166,4,162,4,4Z)
```

Table 27 on page 40 shows the result.

Table 27. Inserting Binary Zeros

Publisher	Number Sold	Number In Stock	X'0...0'
1 4	5 8	9 12	13 16
COR	161	103	0...0
FERN	87	19	0...0
VALD	97	42	0...0
WETH	79	62	0...0

Inserting blanks

If an output data set contains only character data, you can print it by writing the SORTOUT DD statement as follows:

```
//SORTOUT DD SYSOUT=A
```

You can make the printout more readable by using the OUTREC statement to separate the fields with blanks and to create margins. You can insert blanks before, between, or after fields. You can use X or 1X to specify a single blank. You can use nX to specify n blanks.

For example, assume you want to print just the publisher and title fields, with the publisher field appearing first. Because most of the publishers' names fill up the entire 4-byte publisher field, the publishers' names will run into the titles if you do not separate the two fields with blanks. Also, without a margin, the publishers' names will begin at the edge of the paper.

The printout can be made more readable by creating a left margin of 20 blanks and by separating the fields with 10 blanks.

To insert 20 blanks, write 20X before the first field. To insert 10 blanks, write 10X between the two fields. The SORT statement sorts the records by title in ascending order (remember that SORT, COPY, or MERGE is always required).

```
SORT FIELDS=(1,75,CH,A)
OUTREC BUILD=(20X,106,4,10X,1,75)
```

Table 28 on page 40 shows the result.

Table 28. Output after inserting blanks

Publisher				Book Title	
1	20	21 24	25 34	35	190

Table 28. Output after inserting blanks (continued)

Publisher		Book Title	
(20 Blanks)		(10 Blanks)	
	FERN		ADVANCED TOPICS IN PSYCHOANALYSIS
	FERN		COMPUTER LANGUAGES
	WETH		COMPUTERS: AN INTRODUCTION
	COR		CRISES OF THE MIDDLE AGES
	VALD		EDITING SOFTWARE MANUALS
	WETH		EIGHTEENTH CENTURY EUROPE
	COR		INKLINGS: AN ANTHOLOGY OF YOUNG POETS
	VALD		INTRODUCTION TO BIOLOGY
	COR		INTRODUCTION TO PSYCHOLOGY
	COR		LIVING WELL ON A SMALL BUDGET
	COR		MODERN ANTHOLOGY OF WOMEN POETS
	FERN		NUMBERING SYSTEMS
	COR		PICK'S POCKET DICTIONARY
	VALD		SHORT STORIES AND TALL TALES
	VALD		STRATEGIC MARKETING
	COR		SUPPLYING THE DEMAND
	WETH		SYSTEM PROGRAMMING
	FERN		THE COMPLETE PROOFREADER
	WETH		THE INDUSTRIAL REVOLUTION
	VALD		VIDEO GAME DESIGN

If you know the columns you want your fields to start in, you can specify them directly instead of figuring out how many blanks to use. For example, if you want the publisher field to start in column 21 and the title field to start in column 35, as shown previously, you can write the OUTREC statement as:

```
OUTREC BUILD=(21:106,4,35:1,75)
```

c: causes DFSORT to fill in blanks from the last field with data (or column 1) up to the column before c. So 21: puts blanks in columns 1-20 and 35: puts blanks in column 25-34.

Specifying c:X as your last field is an easy way to increase the record length of your output records to c bytes. For example, if you want to create 80-byte output records containing the first 30 bytes of your input records padded with blanks, you can use this OUTREC statement:

```
OUTREC BUILD=(1,30,80:X)
```

DFSORT automatically sets LRECL=80 for the SORTOUT data set if you do not override the LRECL.

Inserting strings

In addition to making the printout more readable, OUTREC can also be used to set up a very basic report format by inserting strings. ICETOOL's DISPLAY operator or the OUTFIL control statement can be used to create complex reports, as you will see later in this document. The formats for writing strings are shown in the rest of this section.

Character strings

The format for repeating a character string is:

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

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

The format for repeating a character string is:

```
nC'x...x'
```

Reformatting Records

where n can be from 1 to 4095; n repetitions of the character string (C'x...x') are inserted into the reformatted records. If n is omitted, 1 is used instead.

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 insert a character string or packed decimal constant for the current date or time of the run in various forms, as detailed in [z/OS DFSORT Application Programming Guide](#). For example, suppose you want to insert the following at the end of each 80 byte record:

- the string 'Timestamp: '
- the date of the run in the form C'yyyy/mm/dd',
- a blank, and
- the time of the run in the form C'hh:mm'

Here's the OUTREC statement:

```
OUTREC FIELDS=(1,80,C'Timestamp: ',DATE1(/),X,TIME2(:))
```

You can also use special keywords to specify a character string or packed decimal constant 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, suppose you want to insert the following at the start of each 20-byte record:

- the current date - 7 days in the form C'yyyyddd',
- a blank, and
- the current date + 14 days in the form C'yyyyddd'.

Here's the OUTREC statement:

```
OUTREC FIELDS=(DATE3-7,X,DATE3+14,1,20)
```

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'.

The format for repeating a hexadecimal string is:

```
nX'yy...yy'
```

where n can be from 1 to 4095. n repetitions of the hexadecimal string (X'yy...yy') are inserted in the reformatted records. If n is omitted, 1 is used.

Setting up a basic report

To produce a very basic report of the publisher's names and author's names from the bookstore data set, you can use the following OUTREC statement to put in "Publisher is" and "Author is" as character separators.

```
OPTION COPY  
OUTREC FIELDS=(11:C'Publisher is ',106,4,  
31:C'Author is ',91,15,X,76,15)
```

The result is shown in [Table 29 on page 43](#).

Table 29. Output of a Report

				Publisher		Author's First Name		Author's Last Name			
1	10	11	22	24	27	31	39	41	55	57	71
(10 blanks)		Publisher is		FERN		Author is		ROBERT		MURRAY	
		Publisher is		COR		Author is		FRANK		DEWAN	
		Publisher is		COR		Author is		TOM		MILLER	
		Publisher is		VALD		Author is		LORI		RASMUSSEN	
		Publisher is		COR		Author is		KAREN		WILDE	
		Publisher is		WETH		Author is		JOKHI		DINSHAW	
		Publisher is		COR		Author is		CAROL		GUSTLIN	
		Publisher is		VALD		Author is		VICTOR		OJALVO	
		Publisher is		FERN		Author is		WILLIAM		BAYLESS	
		Publisher is		VALD		Author is		MARK		YAEGER	
		Publisher is		WETH		Author is		DON		GROSS	
		Publisher is		COR		Author is		PETER		COWARD	
		Publisher is		COR		Author is		LINDA		DUZET	
		Publisher is		FERN		Author is		ANN		GREEN	
		Publisher is		WETH		Author is		RAUL		CAUDILLO	
		Publisher is		VALD		Author is		LILIANA		AVRIL	
		Publisher is		VALD		Author is		CHIEN		WU	
		Publisher is		FERN		Author is		DIANNE		OSTOICH	
		Publisher is		WETH		Author is		ALICE		MUNGER	
		Publisher is		COR		Author is		GREG		BENDER	

So far

So far this chapter has covered how the BUILD or FIELDS parameter of the OUTREC statement can reformat the records of your output data set to contain only the fields and constants you specify. This chapter also covered inserting binary zeros as place holders, and using blanks and strings to make a printout of the output data set more readable. In the following sections, you will learn how to use other reformatting features to change input fields in various ways for output.

Translating uppercase to lowercase, EBCDIC to ASCII, and more

You could use the following statements to show all of the authors for the books used in computer courses:

```
INCLUDE COND=(110,5,CH,EQ,C'COMP')
SORT FIELDS=(76,15,CH,A,91,15,CH,A)
OUTREC FIELDS=(91,15,X,76,15)
```

The results produced for these statements are:

```
WILLIAM      BAYLESS
RAUL         CAUDILLO
JOKHI        DINSHAW
ROBERT       MURRAY
LORI         RASMUSSEN
```

The author's first and last names are displayed as uppercase letters, because they are stored that way in the bookstore data set. If you wanted to display these names with an initial uppercase letter and subsequent lower case letters, you could use the following OUTREC statement:

```
OUTREC FIELDS=(91,1,92,14,TRAN=UTOL,X,76,1,77,14,TRAN=UTOL)
```

The results produced for this OUTREC statement are:

```
William      Bayless
Raul         Caudillo
Jokhi        Dinshaw
```

Robert Lori	Murray Rasmussen
----------------	---------------------

p,m,TRAN=UTOL translates uppercase letter (A-Z) in the field to the equivalent lowercase letter (a-z). In our example, p,m,TRAN=UTOL translates uppercase letters to lowercase letters starting with the **second** letter of the author's first name and the **second** letter of the author's last name. The first letter of each name is not included in the TRAN=UTOL field; these uppercase letters are moved to the output record unchanged.

In addition to TRAN=UTOL, you can also use these other TRAN=keyword functions in a similar way:

- TRAN=LTOU translates lowercase letters (a-z) in a field to the equivalent uppercase letters (A-Z)
- TRAN=ETOA translates EBCDIC characters in a field to the equivalent ASCII characters.
- TRAN=ATOE translates ASCII characters in a field to the equivalent EBCDIC characters.
- TRAN=ALTSEQ translates characters in a field to other characters as specified by an ALTSEQ statement. For example, "null" characters (hex 00) can be changed to blank characters (hex 40)
- TRAN=HEX translates binary values in a field to their equivalent EBCDIC hexadecimal values ('0'-'F'). 2 output characters are produced for each input byte.
- TRAN=BIT translates binary values in a field to their equivalent EBCDIC bit values ('0' or '1'). 8 output characters are produced for each input byte.
- TRAN=UNHEX translates EBCDIC hexadecimal values ('0'-'F') in a field to their equivalent binary values. An output byte is produced for each 2 input characters.
- TRAN=UNBIT translates EBCDIC bit values ('0' or '1') in a field to their equivalent binary values. An output byte is produced for each 8 input characters.

For detailed information on all of the available TRAN=keyword functions, see [z/OS DFSORT Application Programming Guide](#).

Converting numeric fields to different formats

Suppose you want the summed binary values in [Table 25 on page 38](#) be readable so you can print or display them. You can use the following statements to convert the BI values to readable ZD values:

```
OPTION COPY
OUTREC FIELDS=(1,4,X,
                5,4,BI,TO=ZD,LENGTH=6,X,
                9,4,BI,TO=ZD,LENGTH=6)
```

p,m,BI,TO=ZD converts the BI values to ZD values. By default, a 4-byte BI value produces a 10-byte ZD value, but LENGTH=6 overrides the default length to produce a 6-byte ZD value.

[Table 30 on page 44](#) shows the output, which is readable:

Table 30. Converting from BI to ZD

Publisher	Number In Stock	Number Sold
1 4	6 11	13 18
COR	000103	000161
FERN	000019	000087
VALD	000042	000097
WETH	000062	000079

Alternatively, you can use the following OUTREC statement to convert the BI values to readable FS values:

```
OUTREC FIELDS=(1,4,X,
                5,4,BI,TO=FS,LENGTH=6,X,
                9,4,BI,TO=FS,LENGTH=6)
```


p,m,BI,TO=FS converts the BI values to FS values. By default, a 4-byte BI value produces an 11-byte FS value, but LENGTH=6 overrides the default length to produce a 6-byte FS value.

Table 31 on page 45 shows the output, which is readable:

Table 31. Converting from BI to FS

Publisher	Number In Stock	Number Sold
1 4	6 11	13 18
COR	103	161
FERN	19	87
VALD	42	97
WETH	62	79

You can use p,m,f,TO=f or p,m,f,f to convert from various numeric formats to various other numeric formats. You can use LENGTH=n to override the default output length.

For detailed information on DFSORT's numeric conversion parameters, see [z/OS DFSORT Application Programming Guide](#).

Editing numeric fields

Suppose you have input records with numeric values stored in PD format that you want to display or print so they can be easily interpreted. PD values are stored as integers in a compressed internal format that is actually unreadable if you print or display the values.

However, you know that:

- The first PD field represents a number with two decimal places.
- The second PD field represents a number with three decimal places.
- The third PD field represents a date in the form mmddyyyy.

You can use the edit features of the OUTREC statement to display the digits of the PD values as readable characters. Furthermore, you can use the edit features to insert signs, commas, decimal points and hyphens, as appropriate, to make the PD values easy to interpret.

Table 32 on page 45 shows the relevant input fields in the records. The PD fields are represented as readable numbers in Table 32 on page 45, but they are really stored in an unreadable format.

Table 32. Input records with PD values

CH field	First PD field	Second PD field	Third PD field
1 8	11 16	31 37	41 45
WEST	+1524900810	+0000000020000	+05122003
EAST	-0065781053	+0721500532006	+11292003
NORTH	+0000000000	-0000982630735	+02152004
SOUTH	-0000003562	-0003826254999	+12032003

You can use the following OUTREC statement to make the PD values meaningful:

```
OUTREC FIELDS=(1,8,
  5X,
  11,6,PD,M4,
  5X,
  31,7,PD,EDIT=(SI,III,III,IIT.TTT),SIGNS=(,-),
  5X,
  41,5,PD,EDIT=(TT-TT-TTTT))
```

M4 is one of DFSORT's 27 pre-defined edit masks (see [Table 33 on page 46](#)). It edits a numeric field according to the pattern SI,III,III,III,IIT.TT. EDIT=(SI,III,III,IIT.TTT) is a user-defined edit mask. It edits a numeric field according to the pattern SI,III,III,IIT.TTT. EDIT=(TT-TT-TTTT) is another user-defined edit mask. It edits a numeric field according to the pattern TT-TT-TTTT.

In the patterns:

- **I** indicates a leading insignificant digit to be displayed as 1-9, or as blank for a leading 0.
- **T** indicates a significant digit to be displayed as 0-9.
- **S** before the digits indicates a leading sign. **S** after the digits indicates a trailing sign. For M4, the leading sign is to be displayed as + for a positive value or as - for a negative value. For EDIT=(SI,III,III,IIT.TTT),SIGNS=(,-), the leading sign is to be displayed as blank for a positive value or as - for a negative value.
- Any other character (for example, comma, decimal point or hyphen) is just displayed as appropriate.

The results produced for the OUTREC statement are:

WEST	+15,249,008.10	20.000	05-12-2003
EAST	-657,810.53	3,721,500,532.006	11-29-2003
NORTH	+0.00	-982,630.735	02-15-2004
SOUTH	-35.62	-3,826,254.999	12-03-2003

Pre-defined and user-defined edit masks give you a great deal of flexibility in how you format your numeric fields for output. You can use p,m,f,Mnn or p,m,f,EDIT=(pattern) to edit various numeric fields according to various output patterns. You can use SIGNS=(a,b,c,d) to override or set the output signs (leading positive, leading negative, trailing positive and trailing negative). You can use LENGTH=n to override or set the output length.

See [z/OS DFSORT Application Programming Guide](#) for complete details about DFSORT's numeric editing parameters. For easy reference, [Table 33 on page 46](#) shows the patterns for the 27 pre-defined edit masks (M0-M26) as well as examples of the output for each mask.

Table 33. Edit Mask Patterns

Mask	Pattern	Examples	
		Value	Result
M0	IIIIIIIIIIIIIIIIIIIIIIIIITS	+01234	1234
		-00001	1-
M1	TTTTTTTTTTTTTTTTTTTTTTTTTTTS	-00123	00123-
		+00123	00123
M2	II,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,IIT.TTCR	-001234	12.34CR
		+123456	1,234.56
M4	SII,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,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

Table 33. Edit Mask Patterns (continued)

Mask	Pattern	Examples	
		Value	Result
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	IIIIIIIIIIIIIIIIIIIIIT	01234	1234
		00000	0
M11	TTTTTTTTTTTTTTTTTTTTTTTTTTTT	00010	00010
		01234	01234
M12	SI,III,III,III,III,III,III,III,III,III,IT	+1234567	1,234,567
		-0012345	-12,345
M13	SI.III.III.III.III.III.III.III.III.III.IT	+1234567	1.234.567
		-0012345	-12.345
M14	SI III III III III III III III III III ITS	+1234567	1 234 567
		-0012345	(12 345)
M15	I III III III III III III III III III ITS	+1234567	1 234 567
		-0012345	12 345-
M16	SI III III III III III III III III III IT	+1234567	1 234 567
		-0012345	-12 345
M17	SI'III'III'III'III'III'III'III'III'III'IT	+1234567	1'234'567
		-0012345	-12'345
M18	SII,III,III,III,III,III,III,III,III,III,TT	+0123456	1,234.56
		-1234567	-12,345.67
M19	SII.III.III.III.III.III.III.III.III.III,TT	+0123456	1.234,56
		-1234567	-12.345,67
M20	SI III III III III III III III III III IT,TTS	+0123456	1 234,56
		-1234567	(12 345,67)
M21	II III III III III III III III III III IT,TTS	+0123456	1 234,567
		-1234567	12 345,67-
M22	SI III III III III III III III III III IT,TT	+0123456	1 234,56
		-1234567	-12 345,67
M23	SII'III'III'III'III'III'III'III'III'III,TT	+0123456	1'234.56
		-1234567	-12'345.67
M24	SII'III'III'III'III'III'III'III'III'III,TT	+0123456	1'234,56
		-1234567	-12'345,67
M25	SIIIIIIIIIIIIIIIIIIIIIT	+01234	1234
		-00001	-1

Table 33. Edit Mask Patterns (continued)

Mask	Pattern	Examples	
		Value	Result
M26	STTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT	1234	+01234
		-1	-00001

Displaying data in hexadecimal

If you are curious about how the CH and PD values in Table 32 on page 45 are actually stored, you can use this OUTREC statement to display hexadecimal representations of these fields:

```
OUTREC  FIELDS=(1,8,TRAN=HEX,
                2X,
                11,6,TRAN=HEX,
                2X,
                31,7,TRAN=HEX,
                2X,
                41,5,TRAN=HEX)
```

Note: p,m,TRAN=HEX and p,m,HEX are equivalent.

The results produced for this OUTREC statement are:

```
E6C5E2E340404040 01524900810C 0000000020000C 005122003C
C5C1E2E340404040 00065781053D 3721500532006C 011292003C
D5D6D9E3C8404040 000000000000C 0000982630735D 002152004C
E2D6E4E3C8404040 000000003562D 0003826254999D 012032003C
```

Displaying data as bits

You can also use OUTREC to display bit representations of data. For example, if you had the following five input records:

```
A
Z
0
9
$
```

You could use this OUTREC statement to display the data in bit, hexadecimal and character form:

```
OUTREC  BUILD=(1,1,TRAN=BIT,X,1,1,TRAN=HEX,X,1,1)
```

The resulting output records would be:

```
11000001 C1 A
11101001 E9 Z
11110000 F0 0
11111001 F9 9
01011011 5B $
```

Performing arithmetic with numeric fields and constants

Suppose you want to display the results of subtracting the second PD field in Table 32 on page 45 from the first PD field in Table 32 on page 45. As discussed previously, the PD fields are stored as integers, but we know that:

- The first PD field is 6 bytes and represents a signed number with 11 digits including two decimal places.
- The second PD field is 7 bytes and represents a signed number with 13 digits including three decimal places.

In order to do arithmetic on these fields, you must ensure that the integer and decimal parts of the numbers match up correctly. For example, in the first record, the first PD value is +1524900810 representing +15249008.10 and the second PD value is +20000 representing +20.000. In order to subtract the second PD value from the first PD value, you need to put them in the following forms:

```
+15249008.100
+00000020.000
```

You could use the following OUTREC statement to handle the subtraction correctly:

```
OUTREC FIELDS=(1,8,
  5X,
  ((11,6,PD,MUL,+10),SUB,31,7,PD),
  EDIT=(SI,III,III,IIT.TTT),SIGNS=(,-))
```

11,6,PD,MUL,+10 multiplies the first PD field by 10 to match up the decimal places. SUB,31,7,PD subtracts the second PD field from the results of the 11,6,PD,MUL,+10 operation.

EDIT and SIGNS are used to make the results more meaningful.

The results produced for this OUTREC statement are:

```
WEST          15,248,988.100
EAST          -3,722,158,342.536
NORTH         982,630.735
SOUTH         3,826,219.379
```

You can do arithmetic with numeric fields and decimal constants (+n and -n) using the operators MIN (minimum), MAX (maximum), DIV (division), MUL (multiplication), MOD (modulus), ADD (addition) and SUB (subtraction). The order of evaluation precedence for the operators is as follows, but can be changed by using parentheses:

1. MIN and MAX
2. MUL, DIV and MOD
3. ADD and SUB

You can use p,m,f, MIN, MAX, MUL, DIV, MOD, ADD, SUB, +n, -n and parentheses to do a wide variety of arithmetic operations for various numeric formats. The result of an arithmetic operation is a 15-digit ZD value that can be converted to a different numeric format, or edited using pre-defined edit masks (M0-M26) or user-defined edit masks, as discussed earlier.

For complete details on arithmetic operations, see [z/OS DFSORT Application Programming Guide](#).

Converting date fields

Suppose you had input records that looked like this:

```
03212009 Sacramento
05172008 Los Angeles
12302008 Morgan Hill
08132009 Palo Alto
```

The first field is a gregorian date in 'mmddccyy' form, but you'd actually like to reformat it into a julian date in 'ccyy/ddd' form. You can use the following statements to convert the date field to the form you want:

```
OUTREC BUILD=(1,8,Y4W,TOJUL=Y4T(/),X,10,15)
```

The results produced for this OUTREC statement are:

```
2009/080 Sacramento
2008/138 Los Angeles
2008/365 Morgan Hill
2009/225 Palo Alto
```

Y4W and Y4T are two of DFSORT's 4-digit year date field formats. Y4W indicates a date value with the year (ccyy) last such as 'mmdccyy' or 'ddccyy' whereas Y4T indicates a date value with the year first such as 'ccymmdd' or 'ccyyddd'. TOJUL is one of the conversion keywords that can be used with date fields. You can use any of the following after p,m,Y4x or p,m,Y2x to convert a CH, ZD or PD 4-digit or 2-digit year date in one form to a corresponding CH, ZD or PD 4-digit or 2-digit year date in another form or to a corresponding day of the week:

- TOJUL=Yaa - converts to a julian date without a separator (for example, P'2009007').
- TOJUL=Yaa(s) - converts to a julian date with a separator (for example, C'325-2008').
- TOGREG=Yaa - converts to a gregorian date without separators (for example, Z'091121').
- TOGREG=Yaa(s) - converts to a gregorian date with separators (for example, C'2009.09.21').
- WEEKDAY=CHAR3 - converts to a 3 character day of the week (for example, C'WED' for Wednesday).
- WEEKDAY=CHAR9 - converts to a 9 character day of the week (for example, C'THURSDAY ' for Thursday).
- WEEKDAY=DIGIT1 - converts to a 1 digit indicator for the day of the week (for example, C'2' for Monday).
- WEEKNUM=USA - converts to an integer in the range of 1 to 54 that represents the week of the year.
- WEEKNUM=ISO - converts to an integer in the range of 1 to 53 that represents the week of the year.
- Age=YMD – converts to a 8 byte integer which has duration in years (0-9999), months (00-12), and days (00-31).
- Age=YM – converts to a 6 byte integer which has duration in years (0-9999), months (00-12).
- Age=YD – converts to a 7 integer which has duration in years (0-9999), days (00-366).

For complete details on converting from one type of date field to another, see [z/OS DFSORT Application Programming Guide](#).

Performing arithmetic with date fields

Suppose you had an input file with gregorian input dates in ccyyymmdd form like this:

```
20070305
20071213
20080219
20080901
20091122
20090115
20100915
20100630
```

You can use these statements:

```
OUTREC BUILD=(1,8,Y4T,ADDDAYS,+50,TOGREG=Y4T(/),2X,
              1,8,Y4T,SUBMONS,+7,TOGREG=Y4T(/),2X,
              1,8,Y4T,ADDYEARS,+2,TOGREG=Y4T(/))
```

to display the original date and perform arithmetic on these dates as follows:

- Add 50 days
- Subtract 7 months
- Add 2 years

The results produced for this OUTREC statement are:

```
2007/04/24 2006/08/05 2009/03/05
2008/02/01 2007/05/13 2009/12/13
2008/04/09 2007/07/19 2010/02/19
2008/10/21 2008/02/01 2010/09/01
2010/01/11 2009/04/22 2011/11/22
2009/03/06 2008/06/15 2011/01/15
```

```
2010/11/04 2010/02/15 2012/09/15
2010/08/19 2009/11/30 2012/06/30
```

You can perform various types of arithmetic on date fields in either julian or gregorian form, with 2-digit (Y2x) or 4-digit (Y4x) years, in CH, ZD or PD format. You can convert the result to the same form of date or another form, with or without separators.

You can use the following date arithmetic functions:

- ADDDAYS, ADDMONS and ADDYEARS can be used to add days, months or years to a date field.
- SUBDAYS, SUBMONS and SUBYEARS can be used to subtract days, months or years from a date field.
- DATEDIFF can be used to calculate the number of days between two date fields.
- NEXTDday can be used to calculate the next specified day of the week for a date field (where day can be SUN, MON, TUE, WED, THU, FRI or SAT).
- PREVDday can be used to calculate the previous specified day of the week for a date field (where day can be SUN, MON, TUE, WED, THU, FRI or SAT).
- LASTDAYW, LASTDAYM, LASTDAYQ and LASTDAYY can be used to calculate the last day of the week, month, quarter or year for a date field.

For complete details on using date field arithmetic functions, see [z/OS DFSORT Application Programming Guide](#).

Doing lookup and change

You can use the following statements to show all of the course departments and the publishers whose books they use:

```
SORT FIELDS=(110,5,CH,A,106,4,CH,A)
SUM FIELDS=NONE
OUTREC FIELDS=(110,5,X,106,4)
```

The results produced for these statements are:

```
COR
BIOL VALD
BUSIN COR
BUSIN VALD
COMP FERN
COMP VALD
COMP WETH
ENGL COR
ENGL FERN
ENGL VALD
HIST COR
HIST WETH
PSYCH COR
PSYCH FERN
```

This list is rather cryptic as everything is abbreviated. You can use the following OUTREC statement to make the list more meaningful by substituting descriptive words for the abbreviations.

```
OUTREC FIELDS=(110,5,CHANGE=(16,
  C'HIST',C'History',
  C'BUSIN',C'Business',
  C'COMP',C'Computer Science',
  C'ENGL',C'English',
  C'BIOL',C'Biology',
  C'PSYCH',C'Psychology'),
  NOMATCH=(C'Unaffiliated'),
  C'| ',
  106,4,CHANGE=(20,
  C'FERN',C'Fernal Brothers',
  C'COR',C'Cornish Limited',
  C'VALD',C'Valdean and Co.',
  C'WETH',C'Wethman, Inc.'))
```

The results produced for this OUTREC statement are:

Unaffiliated		Cornish Limited
Biology		Valdern and Co.
Business		Cornish Limited
Business		Valdern and Co.
Computer Science		Fernall Brothers
Computer Science		Valdern and Co.
Computer Science		Wethman, Inc.
English		Cornish Limited
English		Fernall Brothers
English		Valdern and Co.
History		Cornish Limited
History		Wethman, Inc.
Psychology		Cornish Limited
Psychology		Fernall Brothers

Table lookup and change allows you to find an input field value in a table of character, hexadecimal or bit constants, and set a corresponding character constant, hexadecimal constant, or input field in the output record.

You use `p, m, CHANGE=(v,find,set,...)` to give the position and length of the input field, the length of the output field, and the "table" consisting of pairs of find constants, and set constants or set fields. For example, for the first field in the previous OUTREC statement:

- The input field starts in position 110.
- The length of the input field and find constants is 5.
- The length of the output field and set constants is 16.
- The first find constant is 'HIST' (padded with a blank at the end to 5 characters).
- The first set constant is 'History' (padded with blanks at the end to 16 characters).

Whenever the course department contains 'HIST', the output field is set to 'History'. Whenever the course department contains 'BUSIN', the output field is set to 'Business', and so on.

You can use `NOMATCH=(set)` to set the output value when an input field value does not match any of the find constants. `NOMATCH=(C'string')` sets a character string constant. `NOMATCH=(X'string')` sets a hexadecimal string constant. `NOMATCH=(q,n)` sets an input field value. For example, in the previous OUTREC statement, an input field value of blanks results in an output field value of 'Unaffiliated' (padded with blanks at the end to 16 characters). If you do not specify `NOMATCH=(set)`, DFSORT terminates with an error message when an input field value does not match any of the find constants.

For complete details on lookup and change, see [z/OS DFSORT Application Programming Guide](#).

Left-justifying and right-justifying data

Suppose you had input records that looked like this:

```

      History
Psychology
      Business
    Biology
    Computer Science

```

Since this is rather messy looking, you can use the following statements to left-justify the data to make it more presentable:

```

OPTION COPY
OUTREC FIELDS=(1,30,JFY=(SHIFT=LEFT))

```

`SHIFT=LEFT` indicates that you want to left-justify. The results produced for this OUTREC statement are:

```

History
Psychology
Business
Biology
Computer Science

```

Alternatively, you can use the following statements to right-justify the data:


```
OPTION COPY
OUTREC FIELDS=(1,30,JFY=(SHIFT=RIGHT))
```

SHIFT=RIGHT indicates that you want to right-justify. The results produced for this OUTREC statement are:

```
      History
    Psychology
    Business
    Biology
Computer Science
```

DFSORT's justify feature also lets you add a leading string, a trailing string, or both, to the data. For example, you can use the following statements to surround the left-justified data with '<' and '>':

```
OPTION COPY
OUTREC FIELDS=(1,30,JFY=(SHIFT=LEFT,LEAD=C'<',TRAIL=C'>'))
```

The results produced for this OUTREC statement are:

```
<History>
<Psychology>
<Business>
<Biology>
<Computer Science>
```

LEAD=string specifies the leading string as a character or hexadecimal constant (1 to 50 bytes).

TRAIL=string specifies the trailing string as a character or hexadecimal constant (1 to 50 bytes).

Notice that when you left-justify, the trailing string is placed directly to the right of the last non-blank character in your data.

If you want to right-justify instead of left-justify, you can use the following statements:

```
OPTION COPY
OUTREC FIELDS=(1,30,JFY=(SHIFT=RIGHT,LEAD=C'<',TRAIL=C'>'))
```

The results produced for this OUTREC statement are:

```
      <History>
    <Psychology>
    <Business>
    <Biology>
Computer <Science>
```

Notice that when you right-justify, the leading string is placed directly to the left of the first non-blank character in your data.

If your leading or trailing string causes the output field to be longer than the input field, you will lose characters. In order to avoid that, you can increase the length of the output field with the LENGTH parameter. For example, suppose you had input records that looked like this:

```
  rats
bats
cats
```

and you added a leading string with these control statements:

```
OPTION COPY
OUTREC FIELDS=(1,7,JFY=(SHIFT=LEFT,LEAD=C'*I love ',TRAIL=C'*'))
```

Since your input field is 7 bytes, your output field is also 7 bytes by default, so your output fields are truncated to:

```
*I love
*I love
*I love
```

Reformatting Records

To avoid truncation, you can use LENGTH=16 to increase the output field length by the 9 characters you added for the leading and trailing strings:

```
OPTION COPY
OUTREC FIELDS=(1,7,JFY=(SHIFT=LEFT,LEAD=C'*I love ',TRAIL=C'*',
LENGTH=16))
```

The larger output field can accommodate your leading and trailing strings so you can show your appreciation for these wonderful creatures with the following output:

```
*I love rats*
*I love bats*
*I love cats*
```

The justify feature can also be used to remove leading and trailing characters other than blanks from your data. Suppose you had input records that looked like this:

```
      (History)
(Psychology)
      (Business)
      (Biology)
      (Computer Science)
```

You can use the following statements to remove the leading '(' character and the trailing ')' character before you left-justify the data:

```
OPTION COPY
OUTREC FIELDS=(1,30,JFY=(SHIFT=LEFT,PREBLANK=C'()''))
```

PREBLANK=list specifies a list of characters you want to replace with blanks before DFSORT starts to justify the data. You can specify the list as a character or hexadecimal constant (1 to 10 bytes). Remember that each character in the list is independent of the other characters. For example, PREBLANK=C'*/' replaces each leading or trailing '*' character and '/' character with a blank before justify processing begins (for example, leading and trailing character sequences of /*, /*, */ and * are all replaced with blanks).

The results produced for this OUTREC statement are:

```
History
Psychology
Business
Biology
Computer Science
```

You could use the following statements to right-justify the data and replace the leading '(' character and trailing ')' character with a leading '<' character and a trailing '>' character:

```
OPTION COPY
OUTREC FIELDS=(1,30,JFY=(SHIFT=RIGHT,PREBLANK=C'()' ',
LEAD=C'<',TRAIL=C'>'))
```

The results produced for this OUTREC statement are:

```
      <History>
    <Psychology>
    <Business>
    <Biology>
  <Computer Science>
```

For complete details on justify, see [z/OS DFSORT Application Programming Guide](#).

Left-squeezing and right-squeezing data

Suppose you had input records that looked like this

```
<tag> History </tag>
<tag> Psychology </tag>
```

```
<tag>      Business </tag>
<tag>Biology</tag>
<tag>      Science  </tag>
```

If you want to remove the white space (blanks), you can use the following statements to left-squeeze the data:

```
OPTION COPY
OUTREC  FIELDS=(1,40,SQZ=(SHIFT=LEFT))
```

SHIFT=LEFT indicates that you want to left-squeeze by removing all of the blanks, shifting the remaining characters to the left and padding on the right with blanks if needed. The results produced for this OUTREC statement are:

```
<tag>History</tag>
<tag>Psychology</tag>
<tag>Business</tag>
<tag>Biology</tag>
<tag>Science</tag>
```

Alternatively, you can use the following statements to right-squeeze the data:

```
OPTION COPY
OUTREC  FIELDS=(1,40,SQZ=(SHIFT=RIGHT))
```

SHIFT=RIGHT indicates that you want to right-squeeze by removing all of the blanks, shifting the remaining characters to the right and padding on the left with blanks if needed. The results produced for this OUTREC statement are:

```
<tag>History</tag>
<tag>Psychology</tag>
<tag>Business</tag>
<tag>Biology</tag>
<tag>Science</tag>
```

DFSORT's squeeze feature also lets you add a leading string, a trailing string, or both, to the data. For example, you can use the following statements to surround the left-squeezed data with '<tag1>' and '</tag1>':

```
OPTION COPY
OUTREC  FIELDS=(1,40,SQZ=(SHIFT=LEFT,LEAD=C'<tag1>',
      TRAIL=C'</tag1>'))
```

The results produced for this OUTREC statement are:

```
<tag1><tag>History</tag></tag1>
<tag1><tag>Psychology</tag></tag1>
<tag1><tag>Business</tag></tag1>
<tag1><tag>Biology</tag></tag1>
<tag1><tag>Science</tag></tag1>
```

LEAD=string specifies the leading string as a character or hexadecimal constant (1 to 50 bytes).

TRAIL=string specifies the trailing string as a character or hexadecimal constant (1 to 50 bytes).

Notice that when you left-squeeze, the trailing string is placed directly to the right of the last non-blank character in your data.

If you want to right-squeeze instead of left-squeeze, you can use the following statements:

```
OPTION COPY
OUTREC  FIELDS=(1,40,SQZ=(SHIFT=RIGHT,LEAD=C'<tag1>',
      TRAIL=C'</tag1>'))
```

The results produced for this OUTREC statement are:

```
<tag1><tag>History</tag></tag1>
<tag1><tag>Psychology</tag></tag1>
<tag1><tag>Business</tag></tag1>
```

```
<tag1><tag>Biology</tag></tag1>
<tag1><tag>Science</tag></tag1>
```

Notice that when you right-squeeze, the leading string is placed directly to the left of the first non-blank character in your data.

If your leading or trailing string causes the output field to be longer than the input field, you will lose characters. In order to avoid that, you can increase the length of the output field with the LENGTH parameter as discussed previously under [“Left-justifying and right-justifying data” on page 52](#).

The squeeze feature can also be used to remove characters other than blanks from your data. Suppose you had input records that looked like this:

```
<tag> (History) </tag>
  <tag> (Psychology) </tag>
  <tag> (Business) </tag>
<tag>(Biology)</tag>
  <tag> (Science) </tag>
```

You can use the following statements to remove the '(' and ')' characters before you left-squeeze the data:

```
OPTION COPY
OUTREC FIELDS=(1,40,SQZ=(SHIFT=LEFT,PREBLANK=C'()'))
```

PREBLANK=list specifies a list of characters you want to replace with blanks before DFSORT starts to squeeze the data. You can specify the list as a character or hexadecimal constant (1 to 10 bytes). Remember that each character in the list is independent of the other characters. For example, PREBLANK=C'*/' replaces each '*' character and '/' character with a blank before squeeze processing begins (for example, character sequences of /*, /*, /*, // and * are all replaced with blanks).

The results produced for this OUTREC statement are:

```
<tag>History</tag>
<tag>Psychology</tag>
<tag>Business</tag>
<tag>Biology</tag>
<tag>Science</tag>
```

You could use the following statements to right-squeeze the data and remove the '(' character and ')' character:

```
OPTION COPY
OUTREC FIELDS=(1,40,SQZ=(SHIFT=RIGHT,PREBLANK=C'()'))
```

The results produced for this OUTREC statement are:

```
<tag>History</tag>
<tag>Psychology</tag>
<tag>Business</tag>
<tag>Biology</tag>
<tag>Science</tag>
```

The squeeze feature can be used to replace groups of blanks between the first nonblank and last nonblank with other characters. Suppose you had input records that looked like this:

Manufacturing	California	+100000
Research	Arizona	+50000
Marketing	Texas	+75000

You could use the following statements to create comma separated variable records:

```
OPTION COPY
OUTREC FIELDS=(1,80,SQZ=(SHIFT=LEFT,MID=C','))
```

The results produced for this OUTREC statement are:

```
Manufacturing,California,+100000
Research,Arizona,+50000
Marketing,Texas,+75000
```

MID=string specifies the string to replace removed blanks or PREBLANK characters as a character or hexadecimal constant (1 to 10 bytes).

If you want DFSORT to "ignore" blanks and PREBLANK characters between pairs of quotes, you can use PAIR=QUOTE with SQZ.

Suppose you had input records that looked like this:

```
"Computer Science A+"      +123
"Ancient Civilization B-"  +521
"Sanskrit A-"              -263
```

You could use the following statements to "protect" the blanks, + and - signs inside the paired quotes while removing them outside the paired quotes, and leave only one blank between the two squeezed fields:

```
OPTION COPY
OUTREC FIELDS=(1,80,SQZ=(SHIFT=LEFT,PAIR=QUOTE,
  PREBLANK=C'+- ',MID=C' '))
```

The results produced for this OUTREC statement are:

```
"Computer Science A+" 123
"Ancient Civilization B-" 521
"Sanskrit A-" 263
```

If you want DFSORT to "ignore" blanks and PREBLANK characters between pairs of apostrophes, you can use PAIR=APOST with SQZ.

Suppose you had input records that looked like this:

```
'Computer Science A+'      +123
'Ancient Civilization B-'  +521
'Sanskrit A-'              -263
```

You could use the following statements to "protect" the blanks, + and - signs inside the paired apostrophes while removing them outside the paired apostrophes, and leave only one blank between the two squeezed fields:

```
OPTION COPY
OUTREC FIELDS=(1,80,SQZ=(SHIFT=LEFT,PAIR=APOST,
  PREBLANK=C'+- ',MID=C' '))
```

The results produced for this OUTREC statement are:

```
'Computer Science A+' 123
'Ancient Civilization B-' 521
'Sanskrit A-' 263
```

For complete details on squeeze, see [z/OS DFSORT Application Programming Guide](#).

So far

Now you know how to use the many features available with the BUILD or FIELDS parameter of the OUTREC statement to reformat your input records in many ways for output. Keep in mind that you can use all of these reformatting features with the BUILD or FIELDS parameter of the INREC statement and with the BUILD or OUTREC parameter of the OUTFIL statement, as well as with the OUTREC statement. In fact, you can use all of these statements together, when appropriate. Next, you will learn how to use the OVERLAY parameter of the OUTREC statement to change one or more fields without affecting the rest of your record.

Reformatting records with OVERLAY

With the BUILD or FIELDS parameter of the OUTREC statement, you build your reformatted output record one item at a time. You must specify each item (unedited, edited or converted input field, blanks, string, and so on) you want in the output record in the order in which you want it to appear.

But if you only want to change one or a few items and keep the rest of the record in its original form, there's an easier way: the OVERLAY parameter of the OUTREC statement. You can use the same items with the OVERLAY parameter we discussed earlier for the BUILD or FIELDS parameter.

Suppose you want to change the course department to lowercase and discount the price of each book by 10%. You can use the following statements to create an output data set with the new price instead of the old price in each output record:

```
OPTION COPY
OUTREC BUILD=(1,109,
              110:110,5,TRAN=UTOL,
              115:115,55,
              170:170,4,BI,SUB,(170,4,BI,DIV,+10),
              TO=BI,LENGTH=4)
```

The arithmetic expression in the OUTREC parameter gives you a new 4-byte BI price as follows:

```
new price = old price - (old price / 10)
```

With the BUILD parameter, you must build the entire record, so you must specify the following:

- 1,109 to keep the bytes before the course department
- 110,5,TRAN=UTOL to change the course department from uppercase to lowercase
- 115,55 to keep the bytes between the course department and the price
- 170:170,4,BI,SUB,(170,4,BI,DIV,+10),TO=BI,LENGTH=4 to replace price with price=(price-(price/10))

But since you are not changing the bytes before the course department or between the course department and the price, it's easier to use the OVERLAY parameter to just overlay the bytes you want to change as follows:

```
OPTION COPY
OUTREC OVERLAY=(110:110,5,TRAN=UTOL,
                170:170,4,BI,SUB,(170,4,BI,DIV,+10),
                TO=BI,LENGTH=4)
```

Table 34 on page 58 shows the copied data set.

Table 34. Books with Course Department and Price Changes		
Book Title	Department	Price
1 75	110 114	170 173

Table 34. Books with Course Department and Price Changes (continued)

Book Title	Department	Price
COMPUTER LANGUAGES	comp	2340
LIVING WELL ON A SMALL BUDGET		8910
SUPPLYING THE DEMAND	busin	1733
VIDEO GAME DESIGN	comp	1980
INKLINGS: AN ANTHOLOGY OF YOUNG POETS	engl	0536
COMPUTERS: AN INTRODUCTION	comp	1710
PICK'S POCKET DICTIONARY		0266
EDITING SOFTWARE MANUALS	engl	1305
NUMBERING SYSTEMS	comp	0324
STRATEGIC MARKETING	busin	2115
THE INDUSTRIAL REVOLUTION	hist	0716
MODERN ANTHOLOGY OF WOMEN POETS	engl	0405
INTRODUCTION TO PSYCHOLOGY	psych	1980
THE COMPLETE PROOFREADER	engl	0563
SYSTEM PROGRAMMING	comp	2876
SHORT STORIES AND TALL TALES	engl	1368
INTRODUCTION TO BIOLOGY	biol	2115
ADVANCED TOPICS IN PSYCHOANALYSIS	psych	2340
EIGHTEENTH CENTURY EUROPE	hist	1611
CRISES OF THE MIDDLE AGES	hist	1080

With the OVERLAY parameter, you don't have to figure out or specify 1,109 or 115,55. Instead you just specify each output starting column (110: and 170:) and its replacement item. Any column you don't overlay remains the same. In addition, in contrast to BUILD items, OVERLAY items can overlap each other and can be specified in any order. In our case, the following OUTREC statement would give the same results as the previous OUTREC statement:

```
OPTION COPY
OUTREC OVERLAY=(170:170,4,BI,SUB,(170,4,BI,DIV,+10),
                 TO=BI,LENGTH=4,
                 110:110,5,TRAN=UTOL)
```

Extending records with OVERLAY

You can also use the OVERLAY parameter to extend your output record without affecting the rest of the record.

Suppose you want to replace the old price with a new price discounted by 10%, but also extend the record with a new field containing the original price. You can use the following statements:

```
OPTION COPY
OUTREC OVERLAY=(174:170,4,
                 170:170,4,BI,SUB,(170,4,BI,DIV,+10),
                 TO=BI,LENGTH=4)
```

Table 35 on page 59 shows the copied data set.

Table 35. New and old prices	
New price	Old price
170 173	174 177

Table 35. New and old prices (continued)

New price	Old price
2340	2600
8910	9900
1733	1925
1980	2199
0536	0595
1710	1899
0266	0295
1305	1450
0324	0360
2115	2350
0716	0795
0405	0450
1980	2200
0563	0625
2876	3195
1368	1520
2115	2350
2340	2600
1611	1790
1080	1200

Since you added a new field at positions 174-177, DFSORT automatically increases the record length from 173 bytes to 177 bytes.

In this case, the order in which you specify the two overlay items is important because you are using the old price for both items. So you must make the copy of the old price before you modify it. Thus, in the previous OUTREC statement, the first overlay item copies the old price to the new field, and the second overlay item replaces the old price with the new price. If you used this OUTREC statement:

```
OUTREC OVERLAY=(170:170,4,BI,SUB,(170,4,BI,DIV,+10),
                 TO=BI,LENGTH=4,
                 174:170,4)
```

you would inadvertently replace the old price with the new price and then copy the new price to the new field. Thus, you would have the new price in both fields instead of having the new price in one field and the old price in the other field. Since OVERLAY allows overlapping items, you must be careful not to destroy a field with an earlier overlay item that you want to use in a later overlay item.

So far

Now you know how to use the OVERLAY parameter of the OUTREC statement to overlay specified columns with reformatting items, without affecting the rest of your reformatted record. Keep in mind that you can use all of these reformatting features with the OVERLAY parameter of the INREC statement and OUTFIL statement, as well as with the OUTREC statement. Next, you will learn how to use the FINDREP parameter of the OUTREC statement to replace or remove data anywhere in your records.

Reformatting records with FINDREP

With the BUILD, FIELDS or OVERLAY parameter of the OUTREC statement, you reformat output records by specifying items that start at a specific position. But if you want to replace or remove data anywhere in your records, you would use the FINDREP parameter of the OUTREC statement instead.

Suppose you had input records that looked like this:


```
*"Goodbye John"*"Goodbye William"*"Goodbye Goodboy"*
"Goodbye Mike""Good Dog""Goodbye Goodbye"
```

You could use the following statements to replace all instances of 'Goodbye' anywhere in the input records with 'Bye'.

```
OPTION COPY
OUTREC FINDREP=(IN=C'Goodbye',OUT=C'Bye')
```

FINDREP indicates that you want to do a find and replace operation. IN identifies the constant you are looking for (the "find" constant) and OUT identifies the constant you want instead (the "replace" constant).

The output records produced by this OUTREC statement are:

```
*"Bye John"*"Bye William"*"Bye Goodboy"*
"Bye Mike""Good Dog""Bye Bye"
```

You can use OUT=C'' (null constant) to remove identified constants. For example, you could use the following statements to remove 'bye' anywhere in the input records:

```
OPTION COPY
OUTREC FINDREP=(IN=C'bye',OUT=C'')
```

The results produced for this OUTREC statement using the input records shown previously are:

```
*"Good John"*"Good William"*"Good Goodboy"*
"Good Mike""Good Dog""Good Good"
```

You could use the following statements to change 'William' to 'Bill', 'Mike' to 'Michael', 'Dog' to 'Beagle' and '*' to '#' anywhere in the input records:

```
OPTION COPY
OUTREC FINDREP=(INOUT=(C'William',C'Bill',
C'Mike',C'Michael',C'Dog',C'Beagle',C'*,C'#'))
```

INOUT identifies pairs of find and replace constants.

The results produced for this OUTREC statement using the input records shown previously are:

```
#"Goodbye John"#"Goodbye Bill"#"Goodbye Goodboy"#
"Goodbye Michael""Good Beagle""Goodbye Goodbye"
```

For complete details on find and replace, see *z/OS DFSORT Application Programming Guide*.

So far

Now you know how to use the FINDREP parameter of the OUTREC statement to replace or remove data anywhere in your records. Keep in mind that you can use all of these reformatting features with the FINDREP parameter of the INREC statement and OUTFIL statement, as well as with the OUTREC statement. Next, you will learn how to use IFTHEN clauses with the OUTREC statement to reformat different records in different ways.

Reformatting records with IFTHEN

The OVERLAY, FINDREP, and BUILD or FIELDS parameters discussed in previous sections let you use the same reformatting items for every output record. IFTHEN clauses for the OUTREC statement let you select subsets of the output records and apply different BUILD, FINDREP or OVERLAY items to them. So, for example, you can apply a set of BUILD items to "type 1" records, a set of OVERLAY items to "type 2" records, do find and replace operations on "type 3" records, and apply a different set of OVERLAY items (or no items) to other types of records.

You can use five types of IFTHEN clauses as follows:

- **WHEN=INIT:** Use one or more WHEN=INIT clauses to apply BUILD, FINDREP or OVERLAY 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 groups of records. You define the records that belong to a group using an appropriate combination of BEGIN=(logexp), END=(logexp), KEYBEGIN=(field) and RECORDS=n parameters.

You can use any logical expression for BEGIN=(logexp) and END=(logexp) that you can use for the COND=(logexp) parameter of an INCLUDE statement as previously discussed in Chapter 3, “Including or omitting records,” on page 23. A BEGIN=(logexp) or END=(logexp) parameter is satisfied when the logical expression evaluates as true.

A KEYBEGIN=(field) parameter is satisfied when the field value changes.

You define how the records in a group are to be changed using the PUSH parameter. You use c: (output column), p,m (field), ID=n (zoned decimal identifier of length n) and SEQ=n (zoned decimal sequence number of length n) in the PUSH parameter to specify how the records in the group are to be changed.

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, FINDREP or OVERLAY items to the subset of your records that satisfy a specified logical expression. You can use any logical expression for WHEN=(logexp) that you can use for the COND=(logexp) parameter of an INCLUDE statement as previously discussed in Chapter 3, “Including or omitting records,” on page 23. 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, FINDREP or OVERLAY items to your 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, FINDREP or OVERLAY items to your 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 changes (if any) and WHEN=GROUP changes (if any) are applied to input records that do not meet the criteria for any of the WHEN=(logexp) clauses.

Suppose you want to produce a report showing the title, publisher, edited price and % discount if you:

- discount the price of books from publisher COR that cost more than \$20.00 by 20%
- discount the price of books from publisher COR that cost \$20.00 or less by 10%
- discount the price of all books from publisher VALD by 25%
- discount the price of all books from the other publishers by 15%

You can use the following statements:

```
OPTION COPY
OUTREC IFTHEN=(WHEN=INIT,
    BUILD=(1:1,40,50:106,4,60:170,4,BI,EDIT=(TT.TT),70:170,4)),
    IFTHEN=(WHEN=(50,4,CH,EQ,C'COR',AND,70,4,BI,GT,+2000),
        OVERLAY=(70:C'20%',X)),
    IFTHEN=(WHEN=(50,4,CH,EQ,C'COR',AND,70,4,BI,LE,+2000),
        OVERLAY=(70:C'10%',X)),
    IFTHEN=(WHEN=(50,4,CH,EQ,C'VALD'),OVERLAY=(70:C'25%',X)),
    IFTHEN=(WHEN=NONE,OVERLAY=(70:C'15%',X))
```

Table 36 on page 62 shows the copied data set.

Table 36. Proposed discounts for books			
Book Title	Publisher	Price	Discount
1 40	50 53	60 64	70 73

Table 36. Proposed discounts for books (continued)			
Book Title	Publisher	Price	Discount
COMPUTER LANGUAGES	FERN	26.00	15%
LIVING WELL ON A SMALL BUDGET	COR	99.00	20%
SUPPLYING THE DEMAND	COR	19.25	10%
VIDEO GAME DESIGN	VALD	21.99	25%
INKLINGS: AN ANTHOLOGY OF YOUNG POETS	COR	05.95	10%
COMPUTERS: AN INTRODUCTION	WETH	18.99	15%
PICK'S POCKET DICTIONARY	COR	02.95	10%
EDITING SOFTWARE MANUALS	VALD	14.50	25%
NUMBERING SYSTEMS	FERN	03.60	15%
STRATEGIC MARKETING	VALD	23.50	25%
THE INDUSTRIAL REVOLUTION	WETH	07.95	15%
MODERN ANTHOLOGY OF WOMEN POETS	COR	04.50	10%
INTRODUCTION TO PSYCHOLOGY	COR	22.00	20%
THE COMPLETE PROOFREADER	FERN	06.25	15%
SYSTEM PROGRAMMING	WETH	31.95	15%
SHORT STORIES AND TALL TALES	VALD	15.20	25%
INTRODUCTION TO BIOLOGY	VALD	23.50	25%
ADVANCED TOPICS IN PSYCHOANALYSIS	FERN	26.00	15%
EIGHTEENTH CENTURY EUROPE	WETH	17.90	15%
CRISES OF THE MIDDLE AGES	COR	12.00	10%

The first IFTHEN clause is a WHEN=INIT clause that uses BUILD to initialize every record with the title, publisher, edited price and a placeholder for the discount. Notice that the publisher field is copied to positions 50-53 and the 4-byte BI price field is copied to positions 70-73. Subsequent IFTHEN clauses are processed for all records and must refer to these fields at their new positions.

The second IFTHEN clause is a WHEN=(logexp) clause that uses OVERLAY to set the discount field to '20%' for the subset of records with 'COR ' as the publisher and more than 2000 as the price. Subsequent IFTHEN clauses are not processed for records that satisfy this logical expression. Subsequent IFTHEN clauses are processed for records that do not satisfy this logical expression.

The third IFTHEN clause is a WHEN=(logexp) clause that uses OVERLAY to set the discount field to '10%' for the subset of records with 'COR ' as the publisher and 2000 or less as the price. Subsequent IFTHEN clauses are not processed for records that satisfy this logical expression. Subsequent IFTHEN clauses are processed for records that do not satisfy this logical expression.

The fourth IFTHEN clause is a WHEN=(logexp) clause that uses OVERLAY to set the discount field to '25%' for the subset of records with 'VALD' as the publisher. Subsequent IFTHEN clauses are not processed for records that satisfy this logical expression. Subsequent IFTHEN clauses are processed for records that do not satisfy this logical expression.

The fifth IFTHEN clause is a WHEN=NONE clause that uses OVERLAY to set the discount field to '15%' for the subset of records that did not satisfy the logical expression for the second, third and fourth IFTHEN clauses.

DFSORT determines an appropriate reformatted output record length from the IFTHEN clauses you specify. However, you can use the IFOUTLEN=n parameter to tell DFSORT the length you want it to use for the reformatted records. For example, suppose you had 80 byte input records, and this DFSORT statement:

```
OUTREC IFTHEN=(WHEN=INIT,OVERLAY=(81:SEQNUM,8,ZD)),
          IFTHEN=(WHEN=(81,8,ZD,EQ,+5),OVERLAY=(21:C'J82')),
          IFTHEN=(WHEN=(81,8,ZD,EQ,+10),OVERLAY=(21:C'M72'))
```

By default, DFSORT would set the reformatted output record length to 88 to accommodate the 8-byte ZD sequence number you added. If you're only using the 8-byte sequence number for the WHEN comparisons, and you don't want the sequence number as part of the reformatted output record, you

could use this statement instead to tell DFSORT to use 80 for the reformatted output record length instead of 88:

```
OUTREC IFTHEN=(WHEN=INIT,OVERLAY=(81:SEQNUM,8,ZD)),
          IFTHEN=(WHEN=(81,8,ZD,EQ,+5),OVERLAY=(21:C'J82')),
          IFTHEN=(WHEN=(81,8,ZD,EQ,+10),OVERLAY=(21:C'M72')),
          IFOUTLEN=80
```

Suppose you have the following 30-byte FB input records:

```
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 write the following DFSORT control statements:

```
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)
```

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
```

For detailed information on using IFTHEN clauses and IFOUTLEN, see z/OS DFSORT Application Programming Guide.

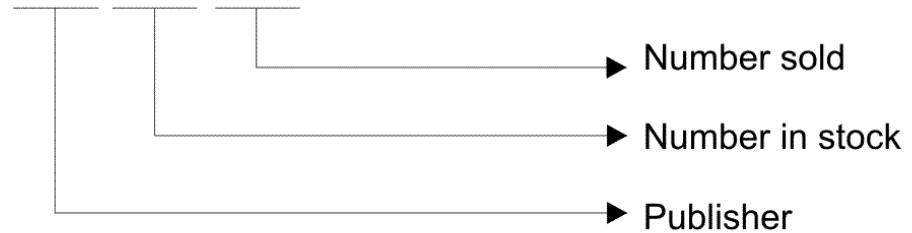
So far

Now you know how to use IFTHEN clauses with the OUTREC statement to reformat different records in different ways and to perform group operations. Keep in mind that you can use IFTHEN clauses with the INREC statement and OUTFIL statement, as well as with the OUTREC statement. Next, you will learn about some special considerations for using the INREC statement.

Reformatting records before sorting

The INREC statement has the same format as the OUTREC statement. Therefore, in the first example of “Reformatting records after sorting with BUILD or FIELDS” on page 38, where you used OUTREC to write only the publisher, number in stock, and number sold fields, you can use INREC instead.

INREC FIELDS=(106,4,162,4,166,4)



Using other statements with INREC

Because INREC reformats the records *before* they are sorted, the SORT and SUM statements must refer to the *reformatted* records as they will appear in the output data set.

Thus, after INREC, the input records for the control statement in the previous section are 12 bytes long (see Table 25 on page 38 for an example).

You write the SORT and SUM statements to process the byte positions in the reformatted records:

```

* Move input positions 106-109 to reformatted record positions 1-4.
* Move input positions 162-169 to reformatted record positions 5-12.
INREC FIELDS=(106,4,162,8)
* Sort reformatted record positions 1-4.
SORT FIELDS=(1,4,CH,A)
* Sum reformatted record positions 5-8 and 9-12.
SUM FIELDS=(5,4,BI,9,4,BI)
  
```

Table 37 on page 65 shows the result.

Table 37. Using INREC to Write Only Publisher, Number in Stock, and Number Sold

Publisher	Number In Stock	Number Sold
1 4	5 8	9 12
COR	103	161
FERN	19	87
VALD	42	97
WETH	62	79

As the flowchart in Appendix C, “Processing order of control statements,” on page 177 shows, DFSORT processes the INREC statement *before* SORT, SUM, and OUTREC, but *after* INCLUDE and OMIT. Therefore, when used with the INREC statement, SORT, SUM, and OUTREC must refer to the *reformatted* records, and INCLUDE and OMIT must refer to the *original* records.

DFSORT processes the OUTFIL statements after the INREC and OUTREC statements. Therefore, OUTFIL must refer to the reformatted records produced by OUTREC if specified, or to the reformatted records produced by INREC if it is specified without OUTREC. You will learn about OUTFIL statements in Chapter 7, “Creating multiple output data sets and reports,” on page 79.

The following control statements illustrate how the INREC and OUTREC statements affect the positions you specify for various other statements:

```
INCLUDE COND=(110,5,CH,EQ,C'ENGL',OR,110,5,CH,EQ,C'PSYCH')
INREC  FIELDS=(1:1,75,76:170,4,80:110,5)
SORT  FIELDS=(76,4,BI,D)
OUTREC FIELDS=(1:1,75,85:76,4,BI,EDIT=($IT.TT),95:80,5)
OUTFIL FNAMES=ENGL,INCLUDE=(95,5,CH,EQ,C'ENGL')
OUTFIL FNAMES=PSYCH,INCLUDE=(95,5,CH,EQ,C'PSYCH')
```

The INCLUDE and INREC statements refer to fields as they appear in the input records. The SORT and OUTREC statements refer to fields as they appear in the reformatted INREC records. The OUTFIL statements refer to fields as they appear in the reformatted OUTREC records.

Preventing overflow when summing values

In some cases, you can prevent overflow by using INREC to pad summary fields with zeros. However, this method cannot be used for negative fixed-point binary data, because padding with zeros rather than with ones would change the sign.

If the summary fields in Table 37 on page 65 were overflowing, you could pad each of them on the left with 4 bytes (binary fields must be 2, 4, or 8 bytes long), as shown in Table 38 on page 66.

Table 38. Padding summary fields (example 1)

Publisher	X'0...0'	Number In Stock	X'0...0'	Number Sold
1 4	5 8	9 12	13 16	17 20
COR		103		161
FERN		19		87
VALD		42		97
WETH		62		79

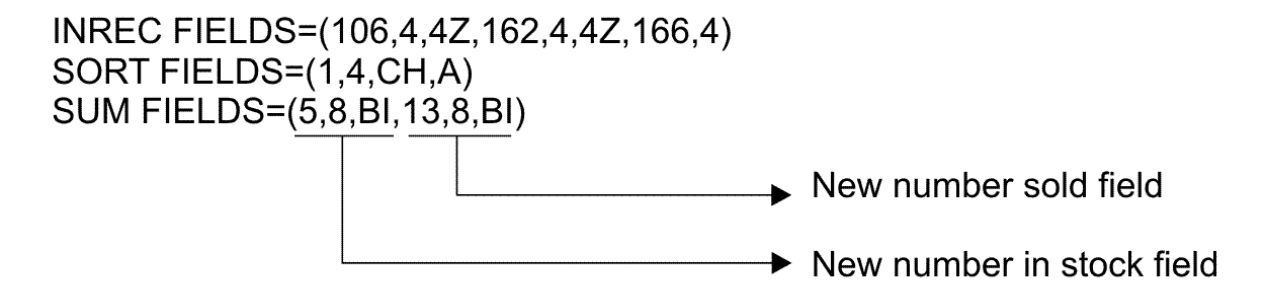


Table 39 on page 67 shows the output records, each 20 bytes long.

Table 39. Padding summary fields (example 2)

Publisher	Number In Stock	Number Sold
1 4	5 12	13 20
COR	103	161
FERN	19	87
VALD	42	97
WETH	62	79



Attention: You cannot use the OUTREC statement to prevent overflow, because it is processed after summarization.

Inserting sequence numbers

Suppose you want to show the total price for all of the books associated with each course department, but you need the list in its original course department order rather than sorted by course department. In order to get the totals, you need to SORT by the department field, and SUM the price field, using the following statements:

```
INREC FIELDS=(110,5,170,4)
SORT FIELDS=(1,5,CH,A)
SUM FIELDS=(6,4,BI)
```

Table 40 on page 67 shows the result of these statements.

Table 40. Total book prices by course

Department	Total Price
1 5	6 9
BIOL	10195
BUSIN	2350
COMP	4275
ENGL	10253
HIST	4640
PSYCH	3785
	4800

Because you had to SORT in order to SUM (remember that you cannot SUM with COPY), the list is in sorted order by course department instead of the original order of COMP, blank, BUSIN, ENGL, HIST, PSYCH, and BIOL you need. To get back the original order, you can add a sequence number to each record **before** the records are sorted, and use a **second step** to sort the records back into their original order by that sequence number. You can remove the sequence numbers at the end of the second step.

You can use the previous statements with a slightly modified INREC statement to add the sequence number for the first step:

```
INREC FIELDS=(110,5,170,4,SEQNUM,8,ZD)
SORT FIELDS=(1,5,CH,A)
SUM FIELDS=(6,4,BI)
```

The sequence numbers start at 00000001 for the first record and are incremented by 00000001 for each subsequent record. This allows you to sort the summed records back into their original course department order by the sequence number.

Table 41 on page 68 shows the result of this first step. The output records are stored in a temporary data set.

Table 41. Total Book Prices by Course with Sequence Numbers

Department	Total Price	Sequence Number
1 5	6 9	10 17
BIOL	10195	00000002
BUSIN	2350	00000017
COMP	4275	00000003
ENGL	10253	00000001
HIST	4640	00000005
PSYCH	3785	00000011
	4800	00000013

You can use the following statements for the second step to sort by the sequence number, display the data in a meaningful way, and remove the sequence numbers. The input for this second step is the temporary data set you created in the first step.

```

SORT  FIELDS=(10,8,CH,A)
OUTREC FIELDS=(1,5,CHANGE=(16,
    C'HIST',C'History',
    C'BUSIN',C'Business',
    C'COMP',C'Computer Science',
    C'ENGL',C'English',
    C'BIOL',C'Biology',
    C'PSYCH',C'Psychology'),
    NOMATCH=(C'Unaffiliated'),
    C'|',
    6,4,BI,EDIT=($III,IIT.TT))

```

The output records from this second step are:

Computer Science		\$102.53
Unaffiliated		\$101.95
Business		\$42.75
English		\$46.40
History		\$37.85
Psychology		\$48.00
Biology		\$23.50

You can use SEQNUM,m,f to create sequence numbers of various lengths in various formats. You can use START=j to start the sequence numbers at j instead of 1. You can use INCR=i to increment each sequence number by i instead of 1. You can use RESTART=(p,m) to start the sequence number at the starting value again each time the value in a particular field changes.

Suppose you have the following input records:

New York	Albany
California	Morgan Hill
New York	Buffalo
Arizona	Tuscon
California	San Jose
New York	Poughkeepsie
Arizona	Phoenix
California	Davis
New York	Armonk

If you wanted to sort the records by the State field in positions 1-15 and by the City field in positions 16-30, and add a third field with a sequence number starting from 1000 and incrementing by 10, you could use START=1000 and INCR=10 as shown in the following statements:

```

SORT  FIELDS=(1,30,CH,A)
OUTREC OVERLAY=(32:SEQNUM,5,ZD,START=1000,INCR=10)

```

The output records would look like this:

Arizona	Phoenix	01000
Arizona	Tuscon	01010

California	Davis	01020
California	Morgan Hill	01030
California	San Jose	01040
New York	Albany	01050
New York	Armonk	01060
New York	Buffalo	01070
New York	Poughkeepsie	01080

Note that each record has a different sequence number.

If you wanted to start the sequence number over at 1000 again each time the State changed, you could add RESTART=(1,15) as shown in the following statements:

```
SORT  FIELDS=(1,30,CH,A)
      OUTREC OVERLAY=(32:SEQNUM,5,ZD,START=1000,INCR=10,
                     RESTART=(1,15))
```

The output records would look like this:

Arizona	Phoenix	01000
Arizona	Tuscon	01010
California	Davis	01000
California	Morgan Hill	01010
California	San Jose	01020
New York	Albany	01000
New York	Armonk	01010
New York	Buffalo	01020
New York	Poughkeepsie	01030

Note that the sequence number starts over at 1000 again each time the State changes.

If you specify an IFTHEN clause with a sequence number, the sequence number is only incremented for the subset of records selected by that IFTHEN clause. For example, if you specified the following statements for the input records shown previously:

```
OPTION COPY
OUTREC IFTHEN=(WHEN=(1,15,CH,EQ,C'Arizona'),
               OVERLAY=(32:SEQNUM,2,ZD)),
        IFTHEN=(WHEN=(1,15,CH,EQ,C'California'),
               OVERLAY=(32:SEQNUM,4,ZD)),
        IFTHEN=(WHEN=(1,15,CH,EQ,C'New York'),
               OVERLAY=(32:SEQNUM,3,ZD))
```

The output records would look like this:

New York	Albany	001
California	Morgan Hill	0001
New York	Buffalo	002
Arizona	Tuscon	01
California	San Jose	0002
New York	Poughkeepsie	003
Arizona	Phoenix	02
California	Davis	0003
New York	Armonk	004

For complete details about creating sequence numbers with INREC, OUTREC and OUTFIL OUTREC, see [z/OS DFSORT Application Programming Guide](#).

VB data set considerations

Some of the same VB data set considerations you learned about previously for the SORT, MERGE, INCLUDE, OMIT and SUM statements also apply to the INREC and OUTREC statements, but some additional considerations apply as well

RDW

If you use the BUILD, FIELDS, or IFTHEN BUILD parameter, you must begin your INREC or OUTREC statement fields by specifying 1,4 (starting position is 1 and length is 4) to represent the RDW in your VB records. For example:

Reformatting Records

```
INREC FIELDS=(1,4,8,25)
```

If you want your first field to include input data bytes immediately after the RDW (that is, starting with position 5), you can use 1,n (starting position is 1 and length is n, with n greater than 4) instead of 1,4. For example:

```
OUTREC BUILD=(1,10,21,30)
```

If you want to display the record length (from positions 1-2 in the RDW), you can use 1,4 to represent the RDW followed by 1,2 for the record length. For example:

```
INREC FIELDS=(1,4,C'Record length is ',1,2,BI,M11)
```

If you use the OVERLAY, IFTHEN OVERLAY, or IFTHEN PUSH parameter of INREC or OUTREC, you must not overlay the RDW in positions 1-4. So all of your items must start at or after position 5. For example:

```
INREC OVERLAY=(5:5,3,TRAN=UTOL,21:18,2,11:C'***')
```

converts the input field at positions 5-7 to lowercase. If you forget to specify c: for the first item, c will default to 1 which will result in an error message because you would be overlaying the RDW. For example:

```
INREC OVERLAY=(5,3,TRAN=UTOL) Results in an error message
```

is interpreted as overlaying output positions 1-3 with the lowercase conversion of input positions 5-7. Since you cannot overlay positions 1-4, you will get an error message .

Be careful not to use 1:, 2:, 3: or 4: for any OVERLAY, IFTHEN OVERLAY, or IFTHEN PUSH item. For example, the following will also result in an error message, since the second item would overlay the RDW:

```
OUTREC OVERLAY=(8:5,3,TRAN=UTOL, ok  
2:18,2) Results in an error message
```

Starting positions and columns

When you code your input fields for the BUILD, FIELDS, OVERLAY, IFTHEN BUILD, IFTHEN OVERLAY, or IFTHEN PUSH parameters, remember to add 4 to the starting position to account for the 4-byte RDW. Likewise, when you code your output columns. For example, the following OUTREC statement puts the field in the first through sixth data byte of the VB input record into positions 21-26 of the VB output record.

```
OUTREC FIELDS=(1,4,21:5,6)
```

Variable data

If you want to include the variable data at the end of each VB input record in the VB output record with the BUILD, FIELDS, or IFTHEN BUILD parameter, specify the starting position of the variable input data without the length. For example:

```
OUTREC FIELDS=(1,4,C'Fixed data ',5,10,C' Variable data ',15)
```

You can also use the following for the variable data at the end of each VB record:

```
p,TRAN=keyword
```

where keyword can be UTOL, LTOU, ETOA, ATOE, ALTSEQ, HEX, UNHEX, BIT or UNBIT.

For example, the following OUTREC statement translates all of the data bytes in each VB record from EBCDIC to ASCII:

```
OUTREC BUILD=(1,4,5,TRAN=ETOA)
```

You must not specify a starting position without a length with the OVERLAY or IFTHEN parameters. DFSORT will issue an error message if you specify any of the following with OVERLAY or IFTHEN OVERLAY:

- p
- p,TRAN=keyword
- p,HEX

For example:

```
INREC OVERLAY=(5:C'***',  
21:21,TRAN=BIT) Results in an error message
```

Summary

This chapter covered using the BUILD, FIELDS, OVERLAY, FINDREP, and IFTHEN parameters of the OUTREC statement and INREC statement to reformat fixed position/length input records in a variety of ways for output. It explained how you can delete fields, reorder fields, overlay fields, insert separators and sequence numbers, convert character and numeric data to various forms, perform arithmetic operations, and change different records in different ways, with the OUTREC and INREC statements, as well as with the OUTFIL statement.

Chapter 6. Reformatting records with variable fields

The previous chapter described how you can reformat records with **fixed fields**, that is, fields that start in the same position and have the same length in every record. This chapter describes how you can reformat records in similar ways with **variable fields**, that is, fields that have different starting positions and lengths in different records, such as comma separated values (CSV).

Using %nnn, %nn and %n parsed fields with BUILD and OVERLAY

There are many types of variable position/length fields such as delimited fields, comma separated values (CSV), tab separated values, blank separated values, keyword separated fields, and so on. For example, you might have four records with comma separated values as follows:

```
Wayne,M,-53,-1732,Gotham
Summers,F,+7258,-273,Sunnydale
Kent,M,+213,-158,Metropolis
Prince,F,-164,+1289,Gateway
```

Note that each record has five variable fields separated by commas. The fields do not start and end in the same position in every record and have different lengths in different records, so you could not just specify the starting position and length (p,m) for any of these fields in a BUILD (or FIELDS or OUTREC) or OVERLAY operand of the INREC, OUTREC or OUTFIL statement. But you can use the PARSE operand of an INREC, OUTREC or OUTFIL statement to define rules that tell DFSORT how to extract the relevant data from each variable input field into a fixed parsed field, and then use the fixed parsed fields in a BUILD or OVERLAY operand as you would use fixed input fields.

You define a parsed field for converting a variable field to a fixed parsed field using a %nnn name where nnn can be 000 to 999, a %nn name where nn can be 00 to 99, or a %n name where n can be 0 to 9. You can define and use up to 1000 parsed fields (%0-%999) per run. Each %nnn, %nn or %n parsed field must be defined only once. %n, %0n and %00n (for example, %1, %01 and %001) are treated as the same parsed field. Likewise, %nn and %0nn (for example, %22 and %022) are treated as the same parsed field. A %nnn, %nn or %n parsed field must be defined in a PARSE operand **before** it is used in a BUILD or OVERLAY operand.

Suppose you wanted to reformat the CSV records to produce these output records:

Wayne	-178.5	Gotham
Summers	698.5	Sunnydale
Kent	5.5	Metropolis
Prince	112.5	Gateway

You can use the following OUTREC statement to parse and reformat the variable fields:

```
OUTREC PARSE=(%01=(ENDBEFR=C',' ,FIXLEN=8),
              %=(ENDBEFR=C',' ),
              %03=(ENDBEFR=C',' ,FIXLEN=5),
              %04=(ENDBEFR=C',' ,FIXLEN=5),
              %05=(FIXLEN=10)),
        BUILD=(%01,14:%03,SFF,ADD,%04,SFF,EDIT=(SIIT.T),SIGNS=(,-),
              25:%05)
```

The PARSE operand defines how each variable field is to be extracted to a fixed parsed field as follows:

- The %01 parsed field is used to extract the first variable field into an 8-byte fixed parsed field. ENDBEFR=C',' tells DFSORT to stop extracting data at the byte before the next comma (the comma after the first variable field). FIXLEN=8 tells DFSORT that the %01 parsed field is 8 bytes long. Thus, for the first record, DFSORT extracts Wayne into the 8-byte %01 parsed field. Since Wayne is only 5 characters, but the %01 parsed field is 8 bytes long, DFSORT pads the %01 parsed field on the right with 3 blanks. ENDBEFR=C',' also tells DFSORT to skip over the comma after the first variable field before it parses the second variable field.

- The % parsed field is used to skip the second variable field without extracting anything for it. Since we don't want this field in the output record, we can use % to ignore it. Thus, for the first record, we ignore M. ENDBEFR=C,' tells DFSORT to skip over the comma after the second variable field before it parses the third variable field.
- The %03 parsed field is used to extract the third variable field into a 5-byte fixed parsed field. ENDBEFR=C,' tells DFSORT to stop extracting data before the next comma (the comma after the third variable field). FIXLEN=5 tells DFSORT that the %03 parsed field is 5 bytes long. Thus, for the first record, DFSORT extracts -53 into the 5-byte %03 parsed field. Since -53 is only 3 characters, but the %03 parsed field is 5 bytes long, DFSORT pads the %03 parsed field on the right with 2 blanks. ENDBEFR=C,' also tells DFSORT to skip over the comma after the third variable field before it parses the fourth variable field.
- The %04 parsed field is used to extract the fourth variable field into a 5-byte fixed parsed field. ENDBEFR=C,' tells DFSORT to stop extracting data before the next comma (the comma after the fourth variable field). FIXLEN= 5 tells DFSORT that the %04 parsed field is 5 bytes long. Thus, for the first record, DFSORT extracts -1732 into the 5-byte %04 parsed field. Since -1732 is 5 characters, it fills up the 5-byte %05 parsed field and padding is not needed. ENDBEFR=C,' also tells DFSORT to skip over the comma after the fourth variable field before it parses the fifth variable field.
- The %05 parsed field is used to extract the fifth variable field into a 10-byte fixed parsed field. FIXLEN=10 tells DFSORT that the %05 parsed field is 10 bytes long. Thus, for the first record, DFSORT extracts Gotham and 4 blanks into the 10-byte %01 parsed field.

The BUILD operand uses the previously extracted fixed parsed fields to build the output record as follows:

- %01 copies the 8-byte fixed-length data extracted from the first variable field to positions 1-8 of the output record. For the first record, positions 1-8 contain 'Wayne '.
- 14:%03,SFF,ADD,%04,SFF,EDIT=(SIIT.T),SIGNS=(,-) adds the 5-byte fixed-length data extracted from the third variable field to the 5-byte fixed-length data extracted from the fourth variable field and places the 6-byte edited result in positions 14-19 of the output record. For the first record, positions 14-19 contain -178.5 (-53 + -1732 = -1785 edited to -178.5). Note that since the %03 and %04 parsed fields may be padded on the right with blanks, we must use the SFF format to handle the sign and digits correctly.
- 25:%05 copies the 10-byte fixed-length data extracted from the fifth variable field to positions 25-34 of the output record. For the first record, positions 25-34 contain 'Gotham '.

For more information, see “Where you can use %nnn, %nn and %n fields in BUILD and OVERLAY” on page 76 and [Example 21](#) in *z/OS DFSORT Application Programming Guide*.

Using %nnn, %nn and %n Parsed Fields with IFTHEN

If you have different variable position/length fields in different types of records, you can convert them to %nnn, %nn or %n fixed parsed fields and use them in IFTHEN clauses. If you define a %nnn, %nn or %n parsed field in a WHEN=INIT clause, you can use it in the IFTHEN BUILD or IFTHEN OVERLAY suboperand of that clause as well as in any IFTHEN clause that follows. If you define a %nnn, %nn or %n parsed field in a WHEN=(logexp), WHEN=ANY or WHEN=NONE clause, you can use it in the IFTHEN BUILD or IFTHEN OVERLAY suboperand of that clause.

Suppose you have the following input records:

```
1/Sam/Charlie;27
2/Bill/48
1/Frank/Vicky;02
1/William/Dale;86
2/Helen/15
```

Note that the records that start with '1' have one fixed field ('1') and three variable fields, whereas the records that start with '2' have one fixed field ('2') and two variable fields. The variable fields in each type of record do not start and end in the same position in every record and have different lengths in different records.

Suppose you wanted to reformat these input records to produce these output records:

1	Sam	Charlie	2.7
2	Bill		4.8
1	Frank	Vicky	0.2
1	William	Dale	8.6
2	Helen		1.5

You can use the following INREC statement to parse and reformat the fixed and variable fields:

```
INREC IFOUTLEN=50,
      IFTHEN=(WHEN=INIT,
        PARSE=(%00=(ABSPOS=3,ENDBEFR=C '/' ,FIXLEN=8))),
      IFTHEN=(WHEN=(1,1,CH,EQ,C'1'),
        PARSE=(%01=(ABSPOS=3,STARTAFT=C '/' ,ENDBEFR=C ';' ,
          FIXLEN=8),%02=(FIXLEN=2))),
        BUILD=(1,1,4:%00,18:%01,30:%02,ZD,EDIT=(T.T))),
      IFTHEN=(WHEN=(1,1,CH,EQ,C'2'),
        PARSE=(%03=(ABSPOS=3,STARTAFT=C '/' ,FIXLEN=2))),
        BUILD=(1,1,4:%00,30:%03,ZD,EDIT=(T.T)))
```

The PARSE suboperand of the WHEN=INIT clause uses the %00 parsed field to extract the first variable field into an 8-byte fixed parsed field for every record. ABSPOS=3 tells DFSORT to start at position 3. ENDBEFR=C '/' tells DFSORT to stop extracting data before the next slash (the slash after the first variable field). FIXLEN=8 tells DFSORT that the %00 parsed field is 8 bytes long. Thus, for the first record, DFSORT extracts Sam into the 8-byte %00 parsed field. Since Sam is only 3 characters, but the %00 parsed field is 8 bytes long, DFSORT pads the %00 parsed field on the right with 5 blanks. The WHEN=INIT clause defines %00 so it can be used for the clauses that follow. (%00 could be used in a BUILD or OVERLAY suboperand for this WHEN=INIT clause, but in this case, it just defines %00 for the other clauses.)

The PARSE suboperand of the first WHEN=(logexp) clause uses the %01 parsed field to extract the second variable field into an 8-byte fixed parsed field, and the %02 parsed field to extract the third variable field into a 2-byte fixed parsed field, for the '1' records.

For %01, ABSPOS=3 tells DFSORT to start at position 3. STARTAFT=C '/' tells DFSORT to start extracting data after the next slash (the slash after the first variable field). ENDBEFR=C ';' tells DFSORT to stop extracting data before the next semicolon (the semicolon after the second variable field). FIXLEN=8 tells DFSORT that the %01 parsed field is 8 bytes long. Thus, for the first record, DFSORT extracts Charlie into the 8-byte %01 parsed field. Since Charlie is only 7 characters, but the %01 parsed field is 8 bytes long, DFSORT pads the %01 parsed field on the right with one blank.

For %02, FIXLEN=2 tells DFSORT that the %02 parsed field is 2 bytes long. Thus, for the first record, DFSORT extracts 27 into the 2-byte %02 parsed field.

The BUILD suboperand of the first WHEN=(logexp) clause uses the %00 parsed field defined by the WHEN=INIT clause and the %01 and %02 parsed fields defined by the first WHEN=(logexp) clause to build each '1' output record.

The PARSE suboperand of the second WHEN=(logexp) clause uses the %03 parsed field to extract the second variable field into a 2-byte fixed parsed field for the '2' records.

For %03, ABSPOS=3 tells DFSORT to start at position 3. STARTAFT=C '/' tells DFSORT to start extracting data after the next slash (the slash after the first variable field). FIXLEN=2 tells DFSORT that the %03 parsed field is 2 bytes long. Thus, for the second record, DFSORT extracts 48 into the 2-byte %03 parsed field.

The BUILD suboperand of the second WHEN=(logexp) clause uses the %00 parsed field defined by the WHEN=INIT clause and the %03 parsed field defined by the second WHEN=(logexp) clause to build each '2' output record.

Where you can use %nnn, %nn and %n fields in BUILD and OVERLAY

You can use a %nnn, %nn or %n parsed field in BUILD or OVERLAY in the same way you can use p,m. For simplicity, we will use %nn to represent %nnn, %nn or %n. Here is a list of the items where you can use %nn (see [z/OS DFSORT Application Programming Guide](#) for details):

- %nn
- %nn,TRAN=keyword
- %nn,HEX
- %nn,f,edit
- %nn,f,to
- %nn,f in arexp,edit
- %nn,f in arexp,to
- %nn,Y2x
- %nn,Y4x
- %nn,Y2x,edit
- %nn,Y4x,edit
- %nn,Y2x,to
- %nn,Y4x,to
- %nn,Y2x,todate
- %nn,Y4x,todate
- %nn,Y2x on left side of date field arithmetic function
- %nn,Y4x on left side of date field arithmetic function
- %nn,Y2x(s)
- %nn,Y4x(s)
- %nn,Y2xP
- %nn,lookup
- %nn as set field in lookup
- %nn as set field in NOMATCH
- %nn,justify
- %nn,squeeze
- RESTART=(%nn) in SEQNUM

PARSE parameters

You can use the following parameters in PARSE to define the rules for extracting variable position/length data to %nnn, %nn and %n fixed parsed fields:

- **FIXLEN=m:** Specifies the length (m) of the fixed area to contain the extracted variable data for this %nnn, %nn or %n fixed parsed field.
- **ABSPOS=p:** Start extracting data at input position p.
- **ADDPOS=x:** Start extracting data at the current position + x.
- **SUBPOS=y:** Start extracting data at the current position - y.
- **STARTAFT=string:** Start extracting data at the byte after the end of the character or hexadecimal string.
- **STARTAFT=an:** Start extracting data at the byte after the first character found from a specified alphanumeric set.

- **STARTAFT=BLANKS:** Start extracting data after the end of the next group of blanks.
- **STARTAT=string:** Start extracting data at the first byte of the character or hexadecimal string.
- **STARTAT=an:** Start extracting data from the first character found from a specified alphanumeric set.
- **STARTAT=BLANKS:** Start extracting data at the start of the first group of blanks.
- **STARTAT=NONBLANK:** Start extracting data at the next nonblank.
- **ENDBEFR=string:** Stop extracting data at the byte before the start of the character or hexadecimal string.
- **ENDBEFR=an:** Stop extracting data at the byte before the first character found from a specified alphanumeric set.
- **ENDBEFR=BLANKS:** Stop extracting data at the byte before the next group of blanks.
- **ENDAT=string:** Stop extracting data at the last byte of the character or hexadecimal string.
- **ENDAT=an:** Stop extracting data at the first character found from a specified alphanumeric set.
- **ENDAT=BLANKS:** Stop extracting data at the end of the next group of blanks.
- **PAIR=APOST:** Do not search for strings or blanks between apostrophe (') pairs.
- **PAIR=QUOTE:** Do not search for strings or blanks between quote (") pairs.
- **REPEAT=v:** Repeat this parsed field v times.

See [*z/OS DFSORT Application Programming Guide*](#) for complete details about defining and using %nnn, %nn and %n parsed fields, as well as more examples.

Summary

You can use variable position/length fields as %nnn, %nn and %n fixed parsed fields in the BUILD, OVERLAY and IFTHEN parameters of the INREC, OUTREC and OUTFIL statements.

Chapter 7. Creating multiple output data sets and reports

You can create one or more output data sets and reports from a single pass over sorted, merged, or copied input using OUTFIL control statements. You can use different OUTFIL parameters on different OUTFIL statements, so the output data sets you create in a single DFSORT application can be identical or very different. You can perform a wide variety of tasks with each OUTFIL statement, including the following:

- Select a sequential subset of records with the STARTREC and ENDREC parameters.
- Select a sample of records with the SAMPLE parameter.
- Select a subset of records with the INCLUDE and OMIT parameters. These parameters have all of the capabilities of the INCLUDE and OMIT statements.
- Limit the number of records selected with the ACCEPT parameter.
- Select "discarded" records with the SAVE parameter. This parameter saves any records that are not selected as a result of STARTREC, ENDREC, SAMPLE, INCLUDE, OMIT or ACCEPT parameters.
- Reformat records with fixed position/length fields or variable position/length fields using the PARSE, BUILD, OUTREC, OVERLAY, or IFTHEN parameters. These parameters have all of the capabilities of the PARSE, BUILD, FIELDS, OVERLAY, or IFTHEN parameters on the OUTREC statement, and also allow you to create multiple output records and blank output records from each input record.
- Repeat records with the REPEAT parameter.
- Split records between output data sets using the SPLIT, SPLITBY, and SPLIT1R parameters.
- Create detailed reports with up to three levels (report, page, and section) using the BUILD, OUTREC, OVERLAY, IFTHEN, HEADERn, TRAILERn, SECTIONS, LINES, NODETAIL, BLKCCH1, BLKCCH2, BLKCCT1, and REMOVECC parameters. Your reports can contain elements such as headers, data, blank lines, trailers, statistics, sections, timestamps, page numbers, and counts.
- Update count and total values in an existing trailer (last) record, based on the current data records, with the IFTRAIL parameter.
- Convert FB records to VB records with the FTOV parameter, or convert VB records to FB records with the VTOF parameter.

You can use OUTFIL statements with any of the other DFSORT statements (for example, INCLUDE, OMIT, INREC, SORT, MERGE, OPTION, SUM and OUTREC). OUTFIL processing is performed last, after all of the processing for the other statements, as shown in Appendix C, "Processing order of control statements," on page 177. Thus, the output records created by the processing for the other statements are the input records for OUTFIL processing. For example, OUTFIL processes the sorted records if the SORT statement is specified, or the copied records if the OPTION COPY statement is specified.

Combining the various DFSORT statements in different ways gives you a lot of flexibility. You can actually use an INREC statement, an OUTREC statement, and an OUTFIL statement to reformat your input records three times. You can also use an INCLUDE or OMIT statement to keep a subset of the input records, and use OUTFIL statements with different INCLUDE or OMIT and OUTREC, OVERLAY, or IFTHEN parameters to write different subsets of the remaining records into different OUTFIL data sets reformatted in different ways

Creating multiple identical copies

Suppose you want to create backups for your data set; on disk for the local site and on tapes for the remote sites. You can do this by using OUTFIL and the FNAMES parameter with an OPTION COPY statement. The FNAMES parameter identifies the DD statements in your JCL for your output data sets by their DD names. Sample JCL and DFSORT control statements are as follows.

```
//COPY JOB A492,PROGRAMMER
//S1 EXEC PGM=SORT
//SYSOUT DD SYSOUT=A
//SORTIN DD DSN=A123456.SORT.SAMPIN,DISP=SHR
//BACKUP DD DSN=A123456.BOOKS.BACKUP,DISP=OLD
//NEWYORK DD DSN=B00KS,UNIT=3490,DISP=(,KEEP),VOL=SER=REMOT1,LABEL=(,SL)
//SANJOSE DD DSN=B00KS,UNIT=3490,DISP=(,KEEP),VOL=SER=REMOT2,LABEL=(,SL)
//SYSIN DD *
    OPTION COPY
    OUTFIL FNAMES=(BACKUP,NEWYORK,SANJOSE)
/*
```

Note: The previous JCL includes two tape data sets. Substitute your own tape data set information if you want to run this example.

Here are the steps for writing the OUTFIL statement:

Table 42. Creating the OUTFIL Statement for the Multiple Output Data Set Job

Step	Action
1	Leave at least one blank and type OUTFIL
2	Leave at least one blank and type FNAMES=
3	Type, in parentheses and separated by commas, each output data set DD name. In this example, they are BACKUP , NEWYORK , and SANJOSE .

This OUTFIL statement creates three backup copies of your data set: one on-site disk data set, and two tapes that can be sent to remote sites.

You can also complete this task using the FILES parameter of OUTFIL. With FILES, you assign SORTOFd or SORTOFdd DD statements instead of naming unique DD statements. Sample JCL and DFSORT control statements using FILES are as follows.

```
//COPY JOB A492,PROGRAMMER
//S1 EXEC PGM=SORT
//SYSOUT DD SYSOUT=A
//SORTIN DD DSN=A123456.SORT.SAMPIN,DISP=SHR
//SORTOF1 DD DSN=A123456.BOOKS.BACKUP,DISP=OLD
//SORTOF2 DD DSN=B00KS,UNIT=3490,DISP=(,KEEP),VOL=SER=REMOT1
//SORTOF3 DD DSN=B00KS,UNIT=3490,DISP=(,KEEP),VOL=SER=REMOT2
//SYSIN DD *
    OPTION COPY
    OUTFIL FILES=(1,2,3)
/*
```

Both FNAMES and FILES create the same output, but FNAMES lets you use descriptive names of up to 8 characters for your OUTFIL DD statements instead of one or two cryptic suffix characters. FNAMES is recommended and is used in all of the multiple output examples in the rest of this chapter.

Note: If you do not specify a FNAMES or FILES parameter for an OUTFIL statement, the OUTFIL ddname is SORTOUT by default.

Selecting and sampling by relative record number

When you copy, sort or merge input records, the resulting output records have implicit relative record numbers starting from 1 for the first record and ending at n (the total number of copied, sorted or merged records) for the last record.

Table 43 on page 81 shows the City field of the output records created by copying the SORT.BRANCH data set, along with the relative record number (abbreviated RRN) for each record (the relative record numbers **do not** actually appear in the output data set).

Table 43. Relative Records Numbers for Copy

City	RRN
1 15	
Los Angeles	1
San Francisco	2
Fort Collins	3
Sacramento	4
Sunnyvale	5
Denver	6
Boulder	7
Morgan Hill	8
Vail	9
San Jose	10
San Diego	11
Aspen	12

Table 44 on page 81 shows the City field of the output records created by sorting the SORT.BRANCH data set on the City field, along with the RRN for each record. Note that the RRNs for the copied output data set and the sorted output data set are different because the order of the output records is different (for example, the record containing 'Denver' has a relative record number of 6 for the copied data set and 3 for the sorted data set).

Table 44. Relative Records Numbers for Sort

City	RRN
1 15	
Aspen	1
Boulder	2
Denver	3
Fort Collins	4
Los Angeles	5
Morgan Hill	6
Sacramento	7
San Diego	8
San Francisco	9
San Jose	10
Sunnyvale	11
Vail	12

You can use these relative record numbers to select groups of sequential records to be written to different OUTFIL data sets. STARTREC=n tells DFSORT to start OUTFIL processing at relative record n. ENDREC=m tells DFSORT to end OUTFIL processing at relative record m. The following statements show how this works:

```
SORT FIELDS=(1,15,CH,A)
OUTFIL FNames=OUT1,ENDREC=4
OUTFIL FNames=OUT2,STARTREC=2,ENDREC=6
OUTFIL FNames=OUT3,STARTREC=7,ENDREC=9
OUTFIL FNames=OUT4,STARTREC=10
```

Because STARTREC is not specified for OUT1, DFSORT starts at RRN 1 and ends at RRN 4. For OUT2, DFSORT starts at RRN 2 and ends at RRN 6. (Note that RRNs 2, 3 and 4 are written to both OUT1 and OUT2.) For OUT3, DFSORT starts at RRN 7 and ends at RRN 9. Because ENDREC is not specified for OUT4, DFSORT starts at RRN 10 and ends at the last record (RRN 12). The RRNs in each OUTFIL data set are as follows:

```
OUT1: 1, 2, 3, 4
OUT2: 2, 3, 4, 5, 6
OUT3: 7, 8, 9
OUT4: 10, 11, 12
```

You can also use relative record numbers to take a "sample" of your records. SAMPLE=n tells DFSORT to process every nth record, starting at the record indicated by STARTREC=x and ending at or before the record indicated by ENDREC=y. SAMPLE=(n,m) tells DFSORT to process m records every nth record, starting at the record indicated by STARTREC=x and ending at or before the record indicated by ENDREC=y. The following statements show how this works:

```
OPTION COPY
OUTFIL FNames=OUT1,SAMPLE=3
OUTFIL FNames=OUT2,STARTREC=4,SAMPLE=2,ENDREC=10
OUTFIL FNames=OUT3,STARTREC=2,SAMPLE=(5,3)
```

For OUT1, DFSORT starts at RRN 1 and writes every third record up to RRN 10 (RRN 13 would be the next record, but there are only 12 records). For OUT2, DFSORT starts at RRN 4 and writes every other record up to RRN 10. For OUT3, DFSORT writes 3 records starting at RRN 2, and writes three records starting at every fifth record after record 2, up to RRN 12. The records in each OUTFIL data set are as follows:

```
OUT1: 1, 4, 7, 10
OUT2: 4, 6, 8, 10
OUT3: 2, 3, 4, 7, 8, 9, 12
```

Including, omitting, and saving discards

In Chapter 3, “Including or omitting records,” on page 23, you learned how to use the INCLUDE and OMIT statements to select only a subset of the input records for sorting, copying or merging. You can also select subsets of records for your individual OUTFIL output data sets in the same way using the INCLUDE and OMIT parameters. Different INCLUDE or OMIT parameters can be used for different OUTFIL statements.

All of the logical expressions that are valid for the COND parameter of the INCLUDE and OMIT statements are also valid for the INCLUDE and OMIT parameters of OUTFIL. However, you cannot specify the FORMAT parameter with the OUTFIL statement as you can with the INCLUDE and OMIT statements. VB data set considerations for the INCLUDE and OMIT statements also apply to the INCLUDE and OMIT parameters.

Suppose you wanted to sort the bookstore data set by title, and create separate output data sets for the English, History and Psychology departments. You could use the following OUTFIL statements to create these three output data sets:

```
SORT FIELDS=(1,75,CH,A)
OUTFIL FNames=ENGLOUT,INCLUDE=(110,5,CH,EQ,C'ENGL')
OUTFIL FNames=HISTOUT,INCLUDE=(110,5,CH,EQ,C'HIST')
OUTFIL FNames=PSYCHOUT,INCLUDE=(110,5,CH,EQ,C'PSYCH')
```

The SORT statement sorts the input records by the title field.

The first OUTFIL statement writes the records for the English department to the ENGLOUT data set. The result is shown in Table 45 on page 82.

Table 45. Output records in ENGLOUT data set

Book Title		Course Department
1	75	110 114
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

The second OUTFIL statement writes the records for the History department to the HISTOUT data set. The result is shown in [Table 46 on page 83](#).

Table 46. Output records in HISTOUT data set

Book Title		Course Department
1	75	110 114
CRISES OF THE MIDDLE AGES		HIST
EIGHTEENTH CENTURY EUROPE		HIST
THE INDUSTRIAL REVOLUTION		HIST

The third OUTFIL statement writes the records for the Psychology department to the PSYCHOUT data set. The result is shown in [Table 47 on page 83](#).

Table 47. Output records in PSYCHOUT data set

Book Title		Course Department
1	75	110 114
ADVANCED TOPICS IN PSYCHOANALYSIS		PSYCH
INTRODUCTION TO PSYCHOLOGY		PSYCH

The previous example creates output data sets for the English, History and Psychology department's books. If you wanted to create an additional output data set (RESTOUT) containing the rest of the books, you could use the following statements:

```
SORT FIELDS=(1,75,CH,A)
OUTFIL FNAMES=ENGLOUT,INCLUDE=(110,5,CH,EQ,C'ENGL')
OUTFIL FNAMES=HISTOUT,INCLUDE=(110,5,CH,EQ,C'HIST')
OUTFIL FNAMES=PSYCHOUT,INCLUDE=(110,5,CH,EQ,C'PSYCH')
OUTFIL FNAMES=RESTOUT,
  INCLUDE=(110,5,CH,NE,C'ENGL',AND,
           110,5,CH,NE,C'HIST',AND,
           110,5,CH,NE,C'PSYCH')
```

This is not too bad for only three departments, but the INCLUDE statement for RESTOUT would get more involved as the number of departments increased.

DFSORT's SAVE parameter gives you an easy way to keep the records discarded by a group of OUTFIL statements. The SAVE parameter selects records for its OUTFIL data set that are not selected for any other OUTFIL data set as a result of the INCLUDE, OMIT, ACCEPT, STARTREC, ENDREC or SAMPLE parameters. So instead of the preceding statements, you could use the following statements to accomplish the same thing

```
SORT FIELDS=(1,75,CH,A)
OUTFIL FNAMES=ENGLOUT,INCLUDE=(110,5,CH,EQ,C'ENGL')
OUTFIL FNAMES=HISTOUT,INCLUDE=(110,5,CH,EQ,C'HIST')
OUTFIL FNAMES=PSYCHOUT,INCLUDE=(110,5,CH,EQ,C'PSYCH')
OUTFIL FNAMES=RESTOUT,SAVE
```

The fourth OUTFIL statement writes the records not used for English, History or Psychology to the RESTOUT data set. The result is shown in [Table 48 on page 83](#).

Table 48. Output records in RESTOUT data set

Book Title		Course Department
1	75	110 114

Table 48. Output records in RESTOUT data set (continued)

Book Title	Course Department
COMPUTER LANGUAGES	COMP
COMPUTERS: AN INTRODUCTION	COMP
INTRODUCTION TO BIOLOGY	BIOL
LIVING WELL ON A SMALL BUDGET	
NUMBERING SYSTEMS	COMP
PICK'S POCKET DICTIONARY	
STRATEGIC MARKETING	BUSIN
SUPPLYING THE DEMAND	BUSIN
SYSTEM PROGRAMMING	COMP
VIDEO GAME DESIGN	COMP

So far

So far, you have learned how to create multiple identical copies of an input data set; select and sample input records by relative record number for output; and include, omit or save selected input records for output. Next, you will learn about reformatting, repeating, and splitting records with OUTFIL.

Reformatting

In Chapter 5, “Reformatting records with fixed fields,” on page 37 and Chapter 6, “Reformatting records with variable fields,” on page 73, you learned how to use the INREC and OUTREC statements to reformat your input records in various ways while sorting, copying, or merging. You can also reformat your OUTFIL output records in the same ways using the PARSE, BUILD, OUTREC, OVERLAY, FINDREP, or IFTHEN parameters. VB data set considerations for the INREC and OUTREC statements also apply to the PARSE, BUILD, OUTREC, OVERLAY and IFTHEN parameters of OUTFIL statements.

Different PARSE, BUILD, OUTREC, OVERLAY or IFTHEN parameters can be used for different OUTFIL statements. You can use these parameters to perform the same wide variety of tasks as for the INREC and OUTREC statements, as described in Chapter 5, “Reformatting records with fixed fields,” on page 37 and Chapter 6, “Reformatting records with variable fields,” on page 73.

In addition, you can use the BUILD, OUTREC, or IFTHEN BUILD parameters to create one, two or more output records from each input record, and insert blank lines before, between or after your records.

Suppose you wanted to create separate data sets for publishers FERN and WETH. For WETH, you want the number of copies sold and the instructor's name on one line. For FERN, you want the course number on one line and the price on another line. You also want two blank lines to separate the information for each book. You can use the following statements to create these two output data sets:

```
OPTION COPY
OUTFIL FNames=WETHBKS,
  INCLUDE=(106,4,CH,EQ,C'WETH'),
  OUTREC=(C'Sold ',166,4,BI,EDIT=(IIT),
    C' copies of Wethman, Inc. book for instructor ',
    160,2,X,145,15)
OUTFIL FNames=FERNBKS,
  INCLUDE=(106,4,CH,EQ,C'FERN'),
  OUTREC=(3:C'Fernal Brothers book #',SEQNUM,2,ZD,
    C' for course ',115,5,/,
    5:C'costs ',170,4,BI,EDIT=($IIT.TT),2/)
```

The first OUTFIL statement uses an INCLUDE parameter to select the books for publisher WETH, and an OUTREC parameter to create the output records for each book with the needed information. The second OUTFIL statement uses an INCLUDE parameter to select the books for publisher FERN, and a different OUTREC parameter to create the output records for each book, with the needed information. Fields, constants, sequence numbers, and numeric editing are used in the OUTREC parameters as previously explained for the INREC and OUTREC statements.

/, n/ and /.../ can be used in the BUILD, OUTREC, or IFTHEN BUILD parameters of the OUTFIL statement, but not in the OVERLAY or IFTHEN OVERLAY parameters of the OUTFIL statement, or in any parameters of the INREC and OUTREC statements. If n/ is used at the start or end of the BUILD, OUTREC or IFTHEN BUILD parameters, n blank lines are inserted. If n/ is used in the middle of the BUILD, OUTREC or IFTHEN parameters, n-1 blank lines are inserted.

Thus, in the OUTREC parameter for FERNBKS, the / (0 blank lines) in the middle after 115,5 is used to create two output records from each FERN input record, and the 2/ (two blank lines) at the end is used to insert two blank lines after the information for each FERN book.

The results produced for WETHBKS are:

```
Sold 26 copies of Wethman, Inc. book for instructor CL CHATTERJEE
Sold 9 copies of Wethman, Inc. book for instructor ST GOODGOLD
Sold 23 copies of Wethman, Inc. book for instructor DC SMITH
Sold 21 copies of Wethman, Inc. Book for instructor HR BISCARDI
```

The results produced for FERNBKS are:

```
Fernall Brothers book #01 for course 00032
costs $26.00

Fernall Brothers book #02 for course 00032
costs $3.60

Fernall Brothers book #03 for course 10347
costs $6.25

Fernall Brothers book #04 for course 30975
costs $26.00
```

DFSORT automatically sets the LRECL of each OUTFIL data set to the reformatted output record length. If n/ is used to create multiple output records from each input record, the LRECL is set to the longest output record.

For WETHBKS, the output record is 71 bytes long, so DFSORT sets the LRECL to 71. For FERNBKS, the first output record is 44 bytes long and the second output record is 17 bytes long, so DFSORT sets the LRECL to 44. If your reformatted length is shorter than you want for the LRECL, you can use c:X to increase the LRECL to c. For example, if you want the LRECL for FERNBKS to be 80 instead of 44, you can change the last line of the OUTREC parameter for FERNBKS to:

```
5:C'costs ',170,4,BI,EDIT=($IIT.TT),80:X,2/)
```

Repeating

OUTFIL's REPEAT=n parameter lets you repeat each output record n times. If you specify the BUILD, OUTREC, OVERLAY, or IFTHEN parameters with a sequence number, the sequence numbers are incremented for each repeated record, but otherwise the repeated records are identical. For example, you could use the following statements to create two output data sets with repeated records; one with just the course name and the other with the course name and a sequence number.

```
OPTION COPY
INCLUDE COND=(110,5,CH,EQ,C'PSYCH')
OUTFIL FNames=OUT1,
      REPEAT=3,
      BUILD=(120,25)
OUTFIL FNames=OUT2,
      REPEAT=3,
      BUILD=(120,25,X,SEQNUM,4,ZD)
```

The results produced for OUT1 are:

```
PSYCHOLOGY I
PSYCHOLOGY I
PSYCHOLOGY I
```

```
PSYCHOANALYSIS  
PSYCHOANALYSIS  
PSYCHOANALYSIS
```

The results produced for OUT2 are:

```
PSYCHOLOGY I      0001  
PSYCHOLOGY I      0002  
PSYCHOLOGY I      0003  
PSYCHOANALYSIS    0004  
PSYCHOANALYSIS    0005  
PSYCHOANALYSIS    0006
```

If you specify an IFTHEN clause with a sequence number, the sequence number is only incremented for the subset of records selected by the IFTHEN clause. For example, if you specified the following statements:

```
OPTION COPY  
INCLUDE COND=(110,5,CH,EQ,C'PSYCH')  
OUTFIL REPEAT=3,  
IFTHEN=(WHEN=(120,7,CH,EQ,C'PSYCHOL'),  
          BUILD=(120,25,X,SEQNUM,4,ZD)),  
IFTHEN=(WHEN=NONE,BUILD=(120,25,X,SEQNUM,2,ZD))
```

The results produced for SORTOUT are:

```
PSYCHOLOGY I      0001  
PSYCHOLOGY I      0002  
PSYCHOLOGY I      0003  
PSYCHOANALYSIS    01  
PSYCHOANALYSIS    02  
PSYCHOANALYSIS    03
```

Splitting

Suppose you don't know how many records are in a data set, but you want to divide the records as equally as possible between two output data sets. You can use OUTFIL's SPLIT parameter to put the first record into OUTPUT1, the second record into OUTPUT2, the third record into OUTPUT1, the fourth record into OUTPUT2, and so on until you run out of records. SPLIT splits the records one at a time among the data sets specified by FNames. The following statements split the records between two OUTFIL data sets:

```
OPTION COPY  
OUTFIL FNames=(OUTPUT1,OUTPUT2),SPLIT
```

With 17 input records, the results produced for OUTPUT1 are:

```
Record 01  
Record 03  
Record 05  
Record 07  
Record 09  
Record 11  
Record 13  
Record 15  
Record 17
```

The results produced for OUTPUT2 are:

```
Record 02  
Record 04  
Record 06  
Record 08  
Record 10  
Record 12  
Record 14  
Record 16
```

Similarly, OUTFIL's SPLITBY=n parameter splits the records n at a time among the data sets specified by FNames. The following statements split the records four at a time between three OUTFIL data sets:

```
OPTION COPY
OUTFIL FNames=(OUT1,OUT2,OUT3),SPLITBY=4
```

With 17 input records, the results produced for OUT1 are:

```
Record 01
Record 02
Record 03
Record 04
Record 13
Record 14
Record 15
Record 16
```

The results produced for OUT2 are:

```
Record 05
Record 06
Record 07
Record 08
Record 17
```

The results produced for OUT3 are:

```
Record 09
Record 10
Record 11
Record 12
```

SPLIT and SPLITBY=n both start over with the first ddname after writing records to the last ddname. This can give you non-contiguous records in one or more of the OUTFIL data sets. For example, in the previous example with SPLITBY=4, OUT1 has records 1-4 and 13-16. If instead, you only want contiguous records in each OUTFIL data set, you can use OUTFIL's SPLIT1R=n parameter which writes n records to each OUTFIL data set and then writes the remaining records to the last OUTFIL data set. Thus, whereas SPLIT and SPLITBY=n rotate many times among the OUTFIL data sets, SPLIT1R=n only rotates once among the OUTFIL data sets.

The following statements ensure that the records are split with only contiguous records in each OUTFIL data set:

```
OPTION COPY
OUTFIL FNames=(OUTA,OUTB,OUTC),SPLIT1R=5
```

With 17 input records, the results produced for OUTA are:

```
Record 01
Record 02
Record 03
Record 04
Record 05
```

The results produced for OUTB are:

```
Record 06
Record 07
Record 08
Record 09
Record 10
```

The results produced for OUTC are:

```
Record 11
Record 12
Record 13
Record 14
Record 15
Record 16
Record 17
```

So far

So far, you have learned how to create multiple identical copies of an input data set; select and sample input records by relative record number for output; include, omit or save selected input records for output; and reformat, repeat and split input records for output. Next, you will learn about creating various types of reports.

Creating reports: OUTFIL vs ICETOOL

Both the OUTFIL statement and ICETOOL's DISPLAY operator can be used to create reports. While each performs some reporting functions that the other does not, in general the difference between them is one of control and effort. With OUTFIL, you have more control over the appearance of reports, but considerable effort may be required on your part to specify in detail where every piece of information is to appear in the report and how it is to look. With ICETOOL, much of the work of determining where information is to appear in the report and how it is to look is done for you, but you have somewhat less control over the appearance of the reports.

ICETOOL should be your primary choice for creating reports, because it is easier to use than OUTFIL. But remember that OUTFIL is available if you need any of its specific reporting features, or more control of the appearance of reports than ICETOOL gives you. Next, you will learn about using OUTFIL to create reports. See [Chapter 12, "Using the ICETOOL utility," on page 123](#) for tutorials on using ICETOOL's DISPLAY operator to create reports.

Creating reports with OUTFIL

You can use OUTFIL to create one or more detailed reports using the OUTFIL parameters BUILD, OUTREC, OVERLAY, FINDREP, IFTHEN, HEADERn, TRAILERn, SECTIONS, LINES, NODETAIL, BLKCCH1, BLKCCH2, BLKCCT1, and REMOVECC. You can include any or all of these in your reports:

- A cover sheet (report header)
- A header at the top of each page (page header)
- A trailer at the bottom of each page (page trailer)
- A header at the start of each section (section header)
- A trailer at the end of each section (section trailer)
- A summary sheet (report trailer)

Your reports can contain a variety of elements you specify such as current date, current time, edited or converted page numbers, character strings, and blank lines. Your reports can also contain a variety of elements you derive from the input records such as 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.

Data

The data for your report consists of the OUTFIL output records. You can use the INCLUDE, OMIT, INREC, SUM and OUTREC statements, and the OUTFIL parameters STARTREC, SAMPLE, ENDREC, INCLUDE, OMIT, ACCEPT, SAVE, PARSE, BUILD, OUTREC, OVERLAY, FINDREP, and IFTHEN to determine which data records appear in your report and what they look like.

The following statements show the simplest possible type of OUTFIL report you can produce:

```
OPTION COPY
OUTFIL FNames=RPT1,LINES=10
```

LINES=10 indicates you want a report with 10 lines per page. These statements might produce the following two page result for RPT1:

```
1Data line 01
Data line 02
```

```
Data line 03
Data line 04
Data line 05
Data line 06
Data line 07
Data line 08
Data line 09
Data line 10
1Data line 11
Data line 12
Data line 13
Data line 14
Data line 15
```

Note: In some of the examples later in this section, small LINES values are used to illustrate a point. However, the default for LINES is 60 and you would generally want to use either that value or one close to it.

For reports, OUTFIL places an ANSI carriage control character in the first byte of each output line to tell a printer what action to take for that line. The '1' for Data line 01 and Data line 11 tells the printer that these lines each start on a new page. This is the way DFSORT translates LINES=10 into 10 lines per page for the printer. Other ANSI carriage control characters that DFSORT uses for OUTFIL reports are: blank for single space (no blank lines before the output line), '0' for double space (one blank line before the output line), and '-' for triple space (two blank lines before the output line).

When you view a report on your display, the ANSI carriage control is not meaningful, and is usually not displayed even though it's actually in the record. If you don't want ANSI carriage control characters in your output records, you can use OUTFIL's REMOVECC parameter to remove them (more on this later).

In general, you will want to use the PARSE, BUILD, OUTREC, OVERLAY, FINDREP, or IFTHEN parameters to reformat the data records for your report. Suppose you want to print a report from the SORT.BRANCH data set showing each branch's revenue and profit or loss, with 9 lines per page. You can use the following statements to show the branches in sorted order, along with their revenue and profit fields in readable form:

```
SORT FIELDS=(1,15,CH,A)
OUTFIL FNAMES=RPT2,LINES=9,
      OUTREC=(1,15,X,
              22,6,PD,EDIT=(SIII,IIT),SIGNS=(,-),X,
              28,6,PD,EDIT=(SIII,IIT),SIGNS=(,-))
```

The two page result produced for RPT2 is the following:

1Aspen	25,800	5,200
Boulder	33,866	7,351
Denver	31,876	6,288
Fort Collins	12,300	-2,863
Los Angeles	22,530	-4,278
Morgan Hill	18,200	3,271
Sacramento	42,726	8,276
San Diego	32,940	8,275
San Francisco	42,820	6,832
1San Jose	27,225	8,264
Sunnyvale	16,152	-978
Vail	23,202	5,027

Headers

The preceding output for RPT2 does not look much like a report, but you can fix that by adding page headings with OUTFIL's HEADER2 parameter. This parameter (as well as the HEADER1 and HEADER3 parameters discussed later), lets you specify multi-line headings with character strings, hexadecimal strings, input fields, the current date, the current time, page numbers and blank lines.

The following statements create a report with page headings:

```
SORT FIELDS=(1,15,CH,A)
OUTFIL FNAMES=RPT3,LINES=15,
      HEADER2=(/,3:'Branch Revenue Report',
              30:'Page',PAGE,45:DATE=(MD4-),2/,
              3:'Branch',25:'Revenue',50:'Profit',/,
```

```
3: '-----',25: '-----',50: '-----'),
OUTREC=(3:1,15,X,
25:22,6,PD,EDIT=(SIII,IIT),SIGNS=(,-),X,
50:28,6,PD,EDIT=(SIII,IIT),SIGNS=(,-))
```

All of the elements in the HEADER2 parameter should be familiar from the previous discussions of the OUTREC statement, except for the PAGE and DATE=(MD4-) parameters.

PAGE tells DFSORT to print a 6-character page number with leading blanks starting at 1 for the first page and incrementing by 1 for each subsequent page. For page 3, PAGE would give you ' 3'. If you wanted to print a 3-character page number with leading zeros, you could use PAGE=(EDIT=(TTT)) or PAGE=(M11,LENGTH=3) or PAGE=(TO=ZD,LENGTH=3). For page 10, these would all give you '010'. See [“Converting numeric fields to different formats”](#) on page 44 and [“Editing numeric fields”](#) on page 45, for more information on conversion and editing.

DATE=(MD4-) tells DFSORT to print the date in the form mm-dd-yyyy. See [z/OS DFSORT Application Programming Guide](#) for more information on the different forms of the date you can print in headers and trailers.

You do not need to start character strings in headers (HEADERn parameters) or trailers (TRAILERn parameters) with C as you do for the BUILD, OUTREC, OVERLAY, FINDREP, or IFTHEN parameters, although you can if you want to.

If n/ is used at the start or end of a header or trailer, n blank lines are printed. If n/ is used in the middle of a header or trailer, n-1 blank lines are printed.

The following is a two-page result produced for RPT3:

```
1
0 Branch Revenue Report      Page      1      04-16-2005
  Branch      Revenue      Profit
-----
Aspen          25,800          5,200
Boulder        33,866          7,351
Denver         31,876          6,288
Fort Collins   12,300         -2,863
Los Angeles    22,530         -4,278
Morgan Hill    18,200          3,271
Sacramento     42,726          8,276
San Diego      32,940          8,275
San Francisco  42,820          6,832
San Jose       27,225          8,264

1
0 Branch Revenue Report      Page      2      04-16-2005
  Branch      Revenue      Profit
-----
Sunnyvale      16,152         -978
Vail           23,202          5,027
```

If you print this report, the ANSI carriage control characters tell the printer to eject a page when it sees '1' in the first byte, and to insert a blank line before the lines with '0' in the first byte. So you get a blank line between the 'Branch Revenue Report' line and the 'Branch' line in your headers, corresponding to the 2/ in your OUTREC parameter. However, if you view this report on a display, the ANSI carriage control characters are ignored, so the '0' line does not have a blank line before it on the display. To force the blank line to appear whether you print the report or view it on a display, you can change the second line of HEADER2 to:

```
30: 'Page ',PAGE,45:DATE=(MD4-),/,X,/,
```

This forces DFSORT to write an additional blank output line instead of using the '0' ANSI carriage control character to insert a blank line for the printer. The blank line appears on the printer or display.

OUTFIL's HEADER1 parameter is very similar to the HEADER2 parameter, except that it produces a report heading on a separate page before the first page of data. You can add the following HEADER1 parameter to your OUTFIL statement:

```
HEADER1=(20:'Cover sheet for Branch Revenue Report',3/,
20:'Printed on ',DATE=(MD4/),', at ',TIME),
```

This separate page is printed at the beginning of the report for RPT3, before the other two pages shown previously. Thus, the first two pages of the report would start like this:

```

1          Cover sheet for Branch Revenue Report
-          Printed on 04/16/2005 at 16:42:20

1
  Branch Revenue Report      Page      1      04-16-2005
  ...

```

Note that the report heading only has a few lines and then we eject to a new page for the first page heading. If we wanted to start the page heading on the same page as the report heading, we could use OUTFIL's BLKCCH2 parameter. This parameter tells DFSORT to replace the '1' in the first line of the first page heading with a blank, thus avoiding the page eject. With BLKCCH2, the first page of the report would start like this:

```

1          Cover sheet for Branch Revenue Report
-          Printed on 04/16/2005 at 16:42:20

  Branch Revenue Report      Page      1      04-16-2005
  ...

```

You can also use OUTFIL's BLKCCH1 parameter to tell DFSORT to replace the '1' in the first line of the report heading with a blank, thus avoiding the page eject for the report heading.

Here are sample JCL statements and control statements for the report using HEADER1, HEADER2, and BLKCCH2:

```

//HDRPT JOB A492,PROGRAMMER
//S1 EXEC PGM=SORT
//SYSOUT DD SYSOUT=A
//SORTIN DD DSN=A123456.SORT.BRANCH,DISP=SHR
//RPT3 DD SYSOUT=A
//SYSIN DD *
  SORT FIELDS=(1,15,CH,A)
  OUTFIL FNames=RPT3,LINES=15,BLKCH2,
    HEADER1=(20:'Cover sheet for Branch Revenue Report',3/,
      20:'Printed on ',DATE=(MD4/),' at ',TIME),
    HEADER2=(/,3:'Branch Revenue Report',
      30:'Page',PAGE,45:DATE=(MD4-),2/,
      3:'Branch',25:'Revenue',50:'Profit',/,
      3:'-----',25:'-----',50:'-----'),
    OUTREC=(3:1,15,X,
      25:22,6,PD,EDIT=(SIII,IIT),SIGNS=(,-),X,
      50:28,6,PD,EDIT=(SIII,IIT),SIGNS=(,-))
/*

```

Of course, you could use a temporary or permanent data set for RPT3 instead of SYSOUT=A. For example:

```

//RPT3 DD DSN= &&MYRPT,DISP=(,PASS),SPACE=(CYL,(2,2)),UNIT=SYSDA

```

Trailers and statistics

You can add trailers to your report with OUTFIL's TRAILER1 (report trailer) and TRAILER2 (page trailer) parameters. You can use any combination of headers and trailers you need.

Like the HEADERn parameters, the TRAILER1 and TRAILER2 parameters (as well as the TRAILER3 parameter discussed later), let you specify multi-line headings with character strings, hexadecimal strings, input fields, the current date, the current time, page numbers and blank lines. In addition, you can include edited or converted record counts, and edited or converted totals, maximums, minimums, and averages for numeric fields, in your trailers.

The following statements create a report with a separate report trailer page containing the overall record count and various overall statistics for the revenue brought in by the branches.

```

SORT FIELDS=(1,15,CH,A)
OUTFIL FNames=RPT4,
  HEADER2=(1:'BRANCH',18:'REVENUE',28:'PROFIT'),
  OUTREC=(1,15,X,

```

```

22,6,PD,EDIT=(SIII,IIT),SIGNS=(,-),X,
28,6,PD,EDIT=(SIII,IIT),SIGNS=(,-),
50:X),
TRAILER1=(2/,
3:'Overall results for branches on ',DATE=(MD4/),' ':',2/,
3:COUNT=(EDIT=(IIT)), ' branches are included',2/,
5:'Total revenue      = ',
TOT=(22,6,PD,M12,LENGTH=10),/,
5:'Average revenue   = ',
AVG=(22,6,PD,M12,LENGTH=10),/,
5:'Lowest revenue    = ',
MIN=(22,6,PD,M12,LENGTH=10),/,
5:'Highest revenue   = ',
MAX=(22,6,PD,M12,LENGTH=10))

```

For OUTFIL reports, DFSORT terminates if any header or trailer record is longer than the data records. In this case, the TRAILER1 line with DATE=(MD4/) is 46 bytes long, so the data records must be at least 46 bytes long. To increase the data length set by the OUTREC parameter from 42 bytes to 50 bytes, you can put 50:X at the end of the OUTREC parameter as shown.

COUNT=(EDIT=(IIT)) shows the count of data records as three digits with leading zeros suppressed. TOT=(22,6,PD,M12,LENGTH=10) shows the total revenue as ten bytes of the M12 pattern (that is, SI,III,IIT). Similarly, AVG, MIN and MAX show the average, minimum and maximum, respectively, as ten bytes of the M12 pattern. See [“Converting numeric fields to different formats” on page 44](#) and [“Editing numeric fields” on page 45](#), for more information on conversion and editing.

Because LINES=n is not specified, the default of 60 lines per page is used.

The two page result produced for RPT4 is:

1BRANCH	REVENUE	PROFIT
Aspen	25,800	5,200
Boulder	33,866	7,351
Denver	31,876	6,288
Fort Collins	12,300	-2,863
Los Angeles	22,530	-4,278
Morgan Hill	18,200	3,271
Sacramento	42,726	8,276
San Diego	32,940	8,275
San Francisco	42,820	6,832
San Jose	27,225	8,264
Sunnyvale	16,152	-978
Vail	23,202	5,027

1
0 Overall results for branches on 04/16/2005:
0 12 branches are included
0 Total revenue = 329,637
Average revenue = 27,469
Lowest revenue = 12,300
Highest revenue = 42,820

The second page is the report trailer page produced for TRAILER1, which normally starts on a new page. However, you can use OUTFIL's BLKCCT1 parameter to avoid forcing a new page for the report trailer. For example, if you change the first line of the OUTFIL statement to:

```
OUTFIL FNames=RPT4,BLKcct1,
```

The one page result produced for RPT4 is:

1BRANCH	REVENUE	PROFIT
Aspen	25,800	5,200
Boulder	33,866	7,351
Denver	31,876	6,288
Fort Collins	12,300	-2,863
Los Angeles	22,530	-4,278
Morgan Hill	18,200	3,271
Sacramento	42,726	8,276
San Diego	32,940	8,275
San Francisco	42,820	6,832
San Jose	27,225	8,264
Sunnyvale	16,152	-978
Vail	23,202	5,027

0 Overall results for branches on 04/16/2005:


```

0  12 branches are included
0  Total revenue   =    329,637
   Average revenue =    27,469
   Lowest revenue  =    12,300
   Highest revenue =    42,820

```

No data or carriage control characters

If you want your report to contain the overall statistics page, but not the actual data lines, you can add UTFIL's NODETAIL parameter to your UTFIL statement. NODETAIL tells DFSORT to process the data records in the usual way, but not to write them to the report data set.

If you do not want the ANSI carriage control characters in your report, you can add UTFIL's REMOVECC parameter to your UTFIL statement. REMOVECC tells DFSORT to remove the carriage control character from the first byte of each record. As a result, the data starts in column 1 of the report rather than in column 2.

Suppose you want to produce two output data sets. The first data set should have a single output record with the total count of records in your input data set. The second data set should have a single output record with today's date in the form ddmmyyyy. The following statements would produce these two output data sets:

```

OPTION COPY
UTFIL FNames=BRANCHCT,NODETAIL,REMOVECC,
      TRAILER1=(COUNT=(TO=ZD,LENGTH=10))
UTFIL FNames=CURDT,NODETAIL,REMOVECC,
      HEADER1=(DATENS=(DM4))

```

If the input data set has 12 records, the single record produced for BRANCHCT with the count in columns 1-10 is:

```
0000000012
```

If the job is run on July 15th, 2004, the single record produced for CURDT with the ddmmyyyy date in columns 1-8 is:

```
15072004
```

VB data set considerations for headers and trailers

When you specify an output column in a HEADERn or TRAILERn parameter, you must not add 4 for the RDW to the column, as you must for the BUILD, OUTREC, OVERLAY, IFTHEN OUTREC, IFTHEN BUILD, IFTHEN OVERLAY or IFTHEN PUSH parameters. Consider this UTFIL statement:

```

UTFIL REMOVECC,
      HEADER2=(1:C'PAGE HEADER'),
      OUTREC=(1,4,5:C'OUTREC FOR DATA')

```

Both the 'PAGE HEADER' string and the 'OUTREC FOR DATA' string start in position 5 of their respective output records. For HEADER2, you use 1: because 4 must not be added to the output column for the RDW, even though the RDW appears in the header record. For OUTREC, you use 5: because 4 must be added to the output column for the RDW. Just be aware of this difference and specify your output columns accordingly.

When you specify an input field in a HEADERn or TRAILERn parameter, you must add 4 for the RDW to the starting position as you do for a BUILD, OUTREC, OVERLAY, IFTHEN BUILD, IFTHEN OUTREC, IFTHEN OVERLAY or IFTHEN PUSH parameter. However, you cannot specify the variable part of the input record (that is, position without length) in a HEADERn or TRAILERn parameter, as you can for a BUILD, OUTREC, IFTHEN BUILD or IFTHEN OUTREC parameter.

Sections

Suppose you want to print one page with information about the Computer department's books, one page with information about the English department's books, and one page about the History department's books. For each department's books, you want to show their course numbers, number in stock, number sold year to date, and price, as well as totals by department for each of the three numeric (BI) fields. The following statements create a report with all of this information:

```
INCLUDE COND=(110,4,SS,EQ,C'COMP,ENGL,HIST')
INREC FIELDS=(1:110,4, Course Department
  CHANGE=(16,
    C'COMP',C'Computer Science',
    C'ENGL',C'English',
    C'HIST',C'History'),
  17:115,5, Course Number
  22:162,4, Number in Stock
  26:166,4, Number Sold Y-to-D
  30:170,4, Price
  50:X) Ensure data length greater than header/trailer length
SORT FIELDS=(1,16,CH,A,17,5,CH,A)
OUTFIL FNAMES=RPT5,
  SECTIONS=(1,16,SKIP=P,
  HEADER3=(3:X,/,
    3:'Department: ',1,16,/,X,/,
    3:'Number',12:'In Stock',23:'Sold YTD',34:' Price',/,
    3:'-----',12:'-----',23:'-----',34:'-----'),
  TRAILER3=(3:'=====',12:'=====',23:'=====',34:'=====',/,
    3:'Totals',
    15:TOT=(22,4,BI,EDIT=(IIIIIT)),
    26:TOT=(26,4,BI,EDIT=(IIIIIT)),
    34:TOT=(30,4,BI,EDIT=(IIIT.TT))),
  OUTREC=(3:17,5,
    15:22,4,BI,EDIT=(IIIIIT),
    26:26,4,BI,EDIT=(IIIIIT),
    34:30,4,BI,EDIT=(IIIT.TT)))
```

The INCLUDE statement and INREC statement remove the unneeded records and fields before sorting. The INREC statement also changes department identifiers (COMP, ENGL, HIST) to more readable strings. As you learned previously, the INREC statement changes the starting positions of various fields, so you must use those new positions for the statements that are processed after INREC (SORT and OUTFIL in this case).

OUTFIL's SECTION parameter is used to divide the report up into sections by the course department. 1,16 tells DFSORT to start a new section every time the department value in columns 1-16 changes. The SORT statement sorts on the department field to bring all of the records for each department value together, so only one section is produced for each department value. If your records are not already sorted by the section field, use the SORT statement to sort them that way. You can sort by other fields as well, but the first sort field must be the section field.

SKIP=P tells DFSORT to start each section on a new page. Alternatively, you can use SKIP=nL if you want your sections to appear on the same page, when possible, with n lines between them.

HEADER3 creates section headers in the same way that HEADER1 and HEADER2 create report and page headers, respectively. TRAILER3 creates section trailers in the same way that TRAILER 1 and TRAILER2 create report and page trailers, respectively.

OUTFIL's OUTREC parameter is used to reformat the data records for the report.

The three page result produced for RPT5 is:

```
1
  Department:  Computer Science

  Number      In Stock      Sold YTD      Price
  -----      -
00032          5           29       26.00
00032          6           27        3.60
00032         20           26       18.99
00103          4           23       31.95
00205         10           10       21.99
=====
Totals         45          115      102.53
```

```

1
Department:  English

Number      In Stock      Sold YTD      Price
-----
10054             10             9      15.20
10347             7             19       6.25
10347            13             32      14.50
10856             1             26       4.50
10856             2             32       5.95
=====
Totals           33            118      46.40

```

```

1
Department:  History

Number      In Stock      Sold YTD      Price
-----
50420             15             9       7.95
50521             14             17      12.00
50632             23             21      17.90
=====
Totals           52             47      37.85

```

Suppose you want to create a second report showing the totals for each department, but not the data lines. To accomplish this, you can use a second OUTFIL statement with the NODETAIL (no data), HEADER2 (page headers), SECTIONS (sections) and TRAILER3 (section trailers) parameters. In this case, because the data lines are not printed, you do not need to reformat them with the OUTREC parameter. The JCL and control statements for the complete job to produce your two reports is:

```

//SCTNRPT JOB A492,PROGRAMMER
//S1 EXEC PGM=SORT
//SYSOUT DD SYSOUT=A
//SORTIN DD DSN=A123456.SORT.SAMPIN,DISP=SHR
//RPT5 DD SYSOUT=A
//RPT6 DD SYSOUT=A
//SYSIN DD *
INCLUDE COND=(110,4,SS,EQ,C'COMP,ENGL,HIST')
INREC FIELDS=(1:110,4, Course Department
              CHANGE=(16,
                      C'COMP',C'Computer Science',
                      C'ENGL',C'English',
                      C'HIST',C'History'),
              17:115,5, Course Number
              22:162,4, Number in Stock
              26:166,4, Number Sold Y-to-D
              30:170,4, Price
              50:X) Ensure data length greater than header/trailer length
SORT FIELDS=(1,16,CH,A,17,5,CH,A)
OUTFIL FNames=RPT5,
        SECTIONS=(1,16,SKIP=P,
                  HEADER3=(3:X,/,
                          3:'Department: ',1,16,/,X,/,
                          3:'Number',12:'In Stock',23:'Sold YTD',34:' Price',/,
                          3:'-----',12:'-----',23:'-----',34:'-----'),
                  TRAILER3=(3:'=====',12:'=====',23:'=====',34:'=====',/,
                          3:'Totals',
                          15:TOT=(22,4,BI,EDIT=(IIIIIT)),
                          26:TOT=(26,4,BI,EDIT=(IIIIIT)),
                          34:TOT=(30,4,BI,EDIT=(IIII.TT)))),
        OUTREC=(3:17,5,
                15:22,4,BI,EDIT=(IIIIIT),
                26:26,4,BI,EDIT=(IIIIIT),
                34:30,4,BI,EDIT=(IIII.TT))
OUTFIL FNames=RPT6,
        NODETAIL,
        HEADER2=(3:'Department ',22:'In Stock',
                31:'Sold YTD',42:' Price',/,
                3:'-----',22:'-----',
                31:'-----',42:'-----'),
        SECTIONS=(1,16,SKIP=0L,
                  TRAILER3=(3:1,16,
                          25:TOT=(22,4,BI,EDIT=(IIIIIT)),
                          34:TOT=(26,4,BI,EDIT=(IIIIIT)),
                          42:TOT=(30,4,BI,EDIT=(IIII.TT))))
/*

```

The three page report produced for RPT5 is the same as shown previously. The one page result produced for RPT6 is:

1	Department	In Stock	Sold YTD	Price
	Computer Science	45	115	102.53
	English	33	118	46.40
	History	52	47	37.85

So far

So far, you have learned how to create multiple identical copies of an input data set; select and sample input records by relative record number for output; include, omit or save selected input records for output; reformat, repeat and split input records for output; and create various types of reports. Next, you will learn how to update counts and totals in trailer records.

Updating counts and totals in trailer with OUTFIL

You can use OUTFIL's IFTRAIL parameter to update count and total values in an existing trailer (last) record to reflect the actual data records in the OUTFIL data set.

Suppose another process deleted 3 records from a data set that originally contained 7 records, but did not update the counts and totals in the trailer record. The modified data set looks like this:

```

1 SAN JOSE      -1.23      +0.35
1 PALO ALTO     +8.34      +1.23
1 DALLAS        -0.03      -3.41
1 MORGAN HILL   -15.25     +20.21
9 COUNT=0007,TOTAL1= -12.32,TOTAL2= +26.15

```

The data records are identified by a '1' and the trailer record is identified by a '9'. Note that the count of data records is 7 even though there are now only 4 data records, and the totals are also incorrect.

You can use the following OUTFIL statement to update the trailer record with the correct count and totals:

```

OUTFIL IFTRAIL=(TRLID=(1,1,CH,EQ,C'9'),
  TRLUPD=(9:COUNT=(EDIT=(TTTT)),
    21:TOT=(16,6,SFF,EDIT=(SIIT.TT),SIGNS=(+,-)),
    36:TOT=(26,6,SFF,EDIT=(SIIT.TT),SIGNS=(+,-))))

```

IFTRAIL tells DFSORT to update the trailer record.

TRLID=(logexp) identifies the trailer record using the same type of logical expression you can use for INCLUDE. In this case, we identify the trailer record as having a '9' in position 1. The identified trailer record is treated as the last record; it is not treated as a data record.

TRLUPD indicates the fields in the data records to be used to update counts and totals. c., COUNT=, COUNT+n=, COUNT-n=, TOT= and TOTAL= can be used in TRLUPD in the same way they are used in TRAILER1.

The output data set will contain these data records and a trailer record updated with the correct count and totals:

```

1 SAN JOSE      -1.23      +0.35
1 PALO ALTO     +8.34      +1.23
1 DALLAS        -0.03      -3.41
1 MORGAN HILL   -15.25     +20.21
9 COUNT=0004,TOTAL1= -8.17,TOTAL2= +18.38

```

You can also use IFTRAIL to update counts and totals as you modify a data set, such as when you use two OUTFIL statements to split a data set.

Suppose your input data set has a header, data and trailer records like this:

```

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 header record is identified by an 'H', the data records are identified by a 'D' and the trailer record is identified by a 'T'. In the trailer record, 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.

You can use the following OUTFIL statements to split the data records by key (key1 and key2) into two output data sets each of which has the header and trailer with accurate count and total for the included data records.

```
OUTFIL FNames=OUT1,
INCLUDE=(3,4,CH,EQ,C'key1'),
IFTRAIL=(HD=YES,TRLID=(1,1,CH,EQ,C'T'),
  TRLUPT=(6:COUNT=(M11,LENGTH=4),
    14:TOT=(10,4,ZD,T0=ZD,LENGTH=6)))
OUTFIL FNames=OUT2,
INCLUDE=(3,4,CH,EQ,C'key2'),
IFTRAIL=(HD=YES,TRLID=(1,1,CH,EQ,C'T'),
  TRLUPT=(6:COUNT=(M11,LENGTH=4),
    14:TOT=(10,4,ZD,T0=ZD,LENGTH=6)))
```

IFTRAIL tells DFSORT to update the trailer record.

INCLUDE includes the records with key1 (for OUT1) or key2 (for OUT2).

HD=YES indicates the first record is a header record. The header record will be output without change and its fields will not be used for the count or total. If the input data set has a header, you must use HD=YES to avoid treating the header record as a data record.

TRLID=(logexp) identifies the trailer record as having a 'T' in position 1. The identified trailer record is treated as the last record; it is not treated as a data record.

TRLUPT indicates the counts and totals to be updated.

The results produced for OUT1 are:

```
H 2010/07/06
D key1    0100
D key1    0150
D key1    0025
T X 0003 QR 000275 D52-007-321-7526
```

The trailer record has a count (0003) and total (000275) for the three included data records.

The results produced for OUT2 are:

```
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
```

The trailer record has a count (0004) and total (001628) for the four included data records.

For complete details on using IFTRAIL, see [z/OS DFSORT Application Programming Guide](#).

So far

So far, you have learned how to create multiple identical copies of an input data set; select and sample input records by relative record number for output; include, omit or save selected input records for output; reformat, repeat and split input records for output; create various types of reports and update counts and totals in trailer records. Next, you will learn about converting between fixed and variable records.

Converting FB to VB

You can convert an FB data set to a VB data set with OUTFIL's FTOV parameter. Each VB output record has a 4-byte RDW followed by the corresponding data from the FB input record, and the length in the RDW is the length of the FB record plus 4.

The following JCL and DFSORT control statements convert the bookstore data set records from FB to VB.

```
//FBVB JOB A92,PROGRAMMER
//S1 EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=A123456.SORT.SAMPIN,DISP=SHR
//VBOUT DD DSN=A123456.SORT.VSAMP,DISP=(NEW,CATLG,DELETE),
// UNIT=3390,SPACE=(CYL,(5,5))
//SYSIN DD *
OPTION COPY
OUTFIL FNAMES=VBOUT,FTOV
/*
```

Because the LRECL of SORT.SAMPIN is 173 bytes, each VB record in SORT.VSAMP is 177 bytes (the FB record length of 173 plus 4 for the RDW) and SORT.VSAMP is given an LRECL of 177.

You can also use PARSE, BUILD, OUTREC, OVERLAY, FINDREP, or IFTHEN parameters with FTOV on the OUTFIL statement. All of the reformatting features are available (input fields, strings, editing, and so on). With FTOV, you specify the input positions, and output columns as you would for an FB record (without the RDW). DFSORT adds the RDW after the FB record is reformatted.

Here is an example of using FTOV with BUILD:

```
OUTFIL FTOV,BUILD=(1:120,25,32:C'in ',110,5)
```

The VB output records look like this:

Positions 1-2:	Length (in RDW) = hex 002B = 43
Positions 3-4:	Zeros (in RDW) = hex 0000 = 0
Positions 5-29:	Input positions 120-144
Positions 30-35:	Blanks
Positions 36-38:	'in '
Positions 39-43:	Input positions 110-114

Each VB output record is 43 bytes long and the output data set is given an LRECL of 43.

For the previous examples, the VB output records are all the same length. You can use OUTFIL'S VLTRIM parameter to create VB output records of different lengths from FB input records. Suppose you have an FB input data set with LRECL=20 and 20-byte records like this:

```
ABC
ABCDEF
AC
ABCDEFGHI
```

These statements change each 20-byte FB input record to a 24-byte VB output record and set LRECL=24 for the VB output data set:

```
OPTION COPY
OUTFIL FTOV
```

The VB output records look like this:

Length		X'0000'		Data	
1	2	3	4	5	24

	24		0	ABC	
	24		0	ABCDEF	
	24		0	AC	
	24		0	ABCDEFGHI	

To remove the trailing blanks from the end of each VB output record, use VLTRIM=C' ' (or VLTRIM=X'40') like this:

```
OUTFIL FTOV,VLTRIM=C' ' '
```

LRECL=24 is still set for the resulting VB output data set, indicating that the maximum record length is 24, but the VB output records are not padded on the right with blanks; they now have different record lengths like this.

Length	X'0000'	Data
1	2	3
7	0	ABC
10	0	ABCDEF
6	0	AC
13	0	ABCDEFGHI

You can use any character or hexadecimal byte value for VLTRIM. For example, VLTRIM=C'*' removes trailing asterisks, and VLTRIM=X'00' removes trailing binary zeros.

Converting VB to FB

You can convert a VB data set to an FB data set with OUTFIL's VTOF and BUILD or OUTREC parameters. All of the PARSE, BUILD, or OUTREC features are available (input fields, strings, editing, and so on). For VTOF, you specify the input positions of the VB input record (with the RDW), and the output columns of the FB output record (without the RDW). DFSORT does not include the RDW in the FB output records.



Attention: You cannot specify OVERLAY, FINDREP, or IFTHEN with VTOF.

The following JCL and DFSORT control statements convert a VB data set with LRECL=104 to an FB data set with LRECL=100:

```
//VBFB JOB A92,PROGRAMMER
//S1 EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=A123456.VBIN,DISP=SHR
//FBOUT DD DSN=A123456.FBOUT,DISP=(NEW,CATLG,DELETE),
// UNIT=3390,SPACE=(CYL,(5,5))
//SYSIN DD *
OPTION COPY
OUTFIL FNAMES=FBOUT,VTOF,OUTREC=(5,100)
/*
```

Up to 100 bytes of data starting at position 5 (after the RDW) of the VB input record appears in the FB output record starting at position 1. The FB output records are all 100 bytes long. By default, if a VB input record has less than 100 data bytes, DFSORT pads the output data with blanks on the right to 100 bytes. However, you can change the padding character to anything you like with OUTFIL's VLFILL parameter. For example, VLFILL=C'*' pads with asterisks and VLFILL=X'00' pads with binary zeros:

```
SORT FIELDS=(25,10,CH,A)
* Pad short fields on the right with blanks (default is VLFILL=C' ')
OUTFIL FNAMES=PADB,VTOF,
OUTREC=(41,40,C'Blank padding',11,20)
* Pad short fields on the right with asterisks
OUTFIL FNAMES=PADA,VTOF,
OUTREC=(5,60,C'Asterisk padding',61,40),VLFILL=C'*'.
* Pad short fields on the right with binary zeros
OUTFIL FNAMES=PADZ,VTOF,
OUTREC=(21,60,C'Binary zero padding'),VLFILL=X'00'
```

LRECL=73 is set for the PADB output data set, which has 73 byte FB records. Short input fields are padded with blanks on the right as needed.

LRECL=116 is set for the PADA output data set, which has 116 byte FB records. Short input fields are padded with asterisks on the right as needed.

LRECL=79 is set for the PADZ output data set, which has 79 byte FB records.
Short input fields are padded with binary zeros on the right as needed.

Summary
<p>This chapter covered the many ways you can use the OUTFIL statement and its various parameters to create multiple output data sets and reports.</p> <p>The next chapter will cover joining records from two data sets.</p>

Chapter 8. Joining records

Often, you have two data sets with common key fields and different data fields, and you want to join their records, that is, for records with matching keys, you want to create output records with some fields from one data set and some fields from the other data set. You might want a one record to one record join, a one record to many records join, a many records to one record join, or even a many records to many records join.

A JOINKEYS application helps you to perform various "join" applications on two data sets by one or more keys. You can do an inner join, full outer join, left outer join, right outer join and unpaired combinations. The two data sets can be of different types (fixed, variable, VSAM, and so on) and lengths, and have keys in different locations.

The records from the input data sets can be processed in a variety of ways before and after they are joined using most of the DFSORT control statements you learned about previously including SORT or COPY, INCLUDE or OMIT, INREC, OUTREC and OUTFIL.

Suppose you have two input data sets, REGION.IN1 and REGION.IN2 as shown in [Table 49 on page 101](#) and [Table 50 on page 101](#). REGION.IN1 has RECFM=FB and LRECL=35 and REGION.IN2 has RECFM=FB and LRECL=27, so their LRECLs are different.

Table 49. REGION.IN1 data set for JOINKEYS application

Region	Headquarters	Regional Director
1 5	6 20	21 35
East	Philadelphia	C. Kent
West	San Jose	B. Wayne
North	Boston	P. Parker
South	Charlotte	D. Prince

Table 50. REGION.IN2 data set for JOINKEYS application

Office	Region	Employees	Evaluation	Established
1 4	5 9	10 13	14 23	24 27
0001	East	0050	Fair	1983
0001	South	0023	Good	1976
0002	South	0068	Fair	1978
0002	East	0125	Excellent	1986
0001	West	0052	Good	1995
0003	East	0028	Good	1994
0002	West	0105	Excellent	2001
0003	South	0054	Fair	1992
0001	North	0200	Fair	1991
0004	South	0070	Good	2002

From these two input data sets, you want to create an output data set, REGION.OUT. For each record in REGION.IN2, you want to look up the corresponding Region in REGION.IN1, and combine fields from the two records into one output record in REGION.OUT, as shown in [Table 51 on page 101](#).

Table 51. REGION.OUT data set for JOINKEYS application

Office	Region	Regional Director	Employees	Evaluation	Headquarters
1 4	5 9	10 24	25 28	29 38	39 53

Table 51. REGION.OUT data set for JOINKEYS application (continued)

Office	Region	Regional Director	Employees	Evaluation	Headquarters
0001	East	C. Kent	0050	Fair	Philadelphia
0002	East	C. Kent	0125	Excellent	Philadelphia
0003	East	C. Kent	0028	Good	Philadelphia
0001	North	P. Parker	0200	Fair	Boston
0001	South	D. Prince	0023	Good	Charlotte
0002	South	D. Prince	0068	Fair	Charlotte
0003	South	D. Prince	0054	Fair	Charlotte
0004	South	D. Prince	0070	Good	Charlotte
0001	West	B. Wayne	0052	Good	San Jose
0002	West	B. Wayne	0105	Excellent	San Jose

Write the following DFSORT JCL and control statements to use a JOINKEYS application to create REGION.OUT from REGION.IN1 and REGION.IN2.

```
//JN1 EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//IN1 DD DSN=DSN=A123456.REGION.IN1,DISP=SHR
//IN2 DD DSN=DSN=A123456.REGION.IN2,DISP=SHR
//SORTOUT DD DSN=A123456.REGION.OUT,DISP=(NEW,CATLG,DELETE),UNIT=3390,
// SPACE=(CYL,(5,5))
//SYSIN DD *
JOINKEYS F1=IN1,FIELDS=(1,5,A) F1 has ddname IN1 and key in 1-5
JOINKEYS F2=IN2,FIELDS=(5,5,A) F2 has ddname IN2 and key in 5-9
REFORMAT FIELDS=(F2:1,4, Office from F2
F1:1,5,21,15, Region and Regional Director from F1
F2:10,4,14,10, Employees and Evaluation from F2
F1:6,15) Headquarters from F1
OPTION COPY Copy joined records
/*
```

Two JOINKEYS statements are required: one for the F1 data set and another for the F2 data set. In this case, the first JOINKEYS statement identifies IN1 as the ddname for the F1 data set and indicates an ascending key (Region) in positions 1-5 of that data set. The second JOINKEYS statement identifies IN2 as the ddname for the F2 data set and indicates an ascending key (Region) is at positions 5-9 of that data set. Each key in the F1 data set must be of the same length and order (ascending or descending) as the corresponding key in the F2 data set, but does not have to be in the same location. The keys are always treated as unsigned binary (INREC can be used to "normalize" the keys in each data set before the records are joined, if necessary).

The F1 data set will be sorted by the key in positions 1-5. The F2 data set will be sorted by the key in positions 5-9. If the records in a data set are already sorted by the key, you can specify SORTED on the JOINKEYS statement to tell DFSORT to copy the records of that data set rather than sorting them.

Records with the same key in both data sets are joined and constructed as directed by the REFORMAT statement using F1: for fields from the F1 record and F2: for fields from the F2 record. This REFORMAT statement creates joined records from the following fields:

- Output positions 1-4: Office from F2
- Output positions 5-9: Region from F1
- Output positions 10-24: Regional Director from F1
- Output positions 25-28: Employees from F2
- Output positions 29-38: Evaluation from F2
- Output positions 39-53: Headquarters from F1

The resulting joined records are 53 bytes long and are copied to the SORTOUT data set (REGION.OUT).

If we wanted to sort the resulting joined records on the Headquarters and Office fields, we could replace the OPTION COPY statement with the following SORT statement:

```
SORT FIELDS=(39,15,CH,A,1,4,CH,A)
```

The resulting sorted output records in REGION.OUT are shown in [Table 52 on page 103](#).

Table 52. REGION.OUT data set Sorted by Headquarters and Office

Office	Region	Regional Director	Employees	Evaluation	Headquarters
1 4	5 9	10 24	25 28	29 38	39 53
0001	North	P. Parker	0200	Fair	Boston
0001	South	D. Prince	0023	Good	Charlotte
0002	South	D. Prince	0068	Fair	Charlotte
0003	South	D. Prince	0054	Fair	Charlotte
0004	South	D. Prince	0070	Good	Charlotte
0001	East	C. Kent	0050	Fair	Philadelphia
0002	East	C. Kent	0125	Excellent	Philadelphia
0003	East	C. Kent	0028	Good	Philadelphia
0001	West	B. Wayne	0052	Good	San Jose
0002	West	B. Wayne	0105	Excellent	San Jose

You can also use a JOINKEYS application to match records from two different input data sets in various ways. Suppose you have two input data sets, CITIES.IN1 and CITIES.IN2 as shown in [Table 53 on page 103](#) and [Table 54 on page 104](#).

Table 53. CITIES.IN1 data set

City	State	District
1 20	21 35	36 37
GILROY	CALIFORNIA	05
GILROY	CALIFORNIA	10
MORGAN HILL	CALIFORNIA	03
PALO ALTO	CALIFORNIA	15
PALO ALTO	CALIFORNIA	08
PALO ALTO	CALIFORNIA	21
SACRAMENTO	CALIFORNIA	05
SAN JOSE	CALIFORNIA	02
SAN JOSE	CALIFORNIA	10
SAN MARTIN	CALIFORNIA	12
AUSTIN	TEXAS	21
DALLAS	TEXAS	15
DALLAS	TEXAS	25
BARRE	VERMONT	07
BARRE	VERMONT	12
STOWE	VERMONT	09

Table 54. CITIES.IN2 data set

State		City	
1	15	16	35
TEXAS		AUSTIN	
CALIFORNIA		SACRAMENTO	
CALIFORNIA		GILROY	
VERMONT		BURLINGTON	
CALIFORNIA		MODESTO	
TEXAS		LAREDO	
VERMONT		BARRE	
CALIFORNIA		LOS ANGELES	
CALIFORNIA		SAN JOSE	

From these two data sets, you want to create a CITIES.OUT data set with the records for Cities that appear in CITIES.IN1, but not in CITIES.IN2, as shown in [Table 55 on page 104](#).

Table 55. CITIES.OUT data set

City		State		District	
1	20	21	35	36	37
MORGAN HILL		CALIFORNIA		03	
PALO ALTO		CALIFORNIA		15	
PALO ALTO		CALIFORNIA		08	
PALO ALTO		CALIFORNIA		21	
SAN MARTIN		CALIFORNIA		12	
DALLAS		TEXAS		15	
DALLAS		TEXAS		25	
STOWE		VERMONT		09	

Write the following DFSORT JCL and control statements to use a JOINKEYS application to create CITIES.OUT from CITIES.IN1 and CITIES.IN2.

```
//JN2 EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//SORTJNF1 DD DSN=DSN=A123456.CITIES.IN1,DISP=SHR
//SORTJNF2 DD DSN=DSN=A123456.CITIES.IN2,DISP=SHR
//SORTOUT DD DSN=A123456.CITIES.OUT,DISP=OLD
//SYSIN DD *
* F1 keys are State and City - F1 is already sorted
* by those keys.
  JOINKEYS FILE=F1,FIELDS=(21,15,A,1,20,A),SORTED
* F2 keys are State and City
  JOINKEYS FILE=F2,FIELDS=(1,15,A,16,20,A)
* Keep the records in F1 that do not have a match in F2
* for the State and City.
  JOIN UNPAIRED,F1,ONLY
  OPTION COPY
/*
```

The first JOINKEYS statement identifies SORTJNF1 as the ddname for the F1 data set and indicates ascending keys (State and City) in positions 21-35 and 1-20 of that data set. Since the F1 records are already in order by the State and City fields, SORTED is used to do a Copy of the F1 records rather than a Sort.

The second JOINKEYS statement identifies SORTJNF2 as the ddname for the F2 data set and indicates ascending keys (State and City) in positions 1-15 and 16-35 of that data set. Since the F2 records are not already in order by the State and City fields, SORTED is not used and a Sort is performed for the F2 records on the indicated keys.

JOIN UNPAIRED,F1,ONLY is used to restrict the output (SORTOUT) to the records in F1 that do not have matching keys in F2. Since we want the entire F1 record, we do not need a REFORMAT statement.

Now suppose we want to use the CITIES.IN1 and CITIES.IN2 data sets again, but this time we want to produce the following output from these two data sets:

- BOTH.OUT: CALIFORNIA and TEXAS cities that appear in both CITIES.IN1 and CITIES.IN2.
- F1ONLY.OUT: CALIFORNIA and TEXAS cities that appear in CITIES.IN1, but not in CITIES.IN2.
- F2ONLY.OUT: CALIFORNIA and TEXAS cities that appear in CITIES.IN2, but not in CITIES.IN1.

Write the following DFSORT JCL and control statements to use a JOINKEYS application to create BOTH.OUT, F1ONLY.OUT and F2ONLY.OUT from CITIES.IN1 and CITIES.IN2.

```
//JN3 EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//CITIES1 DD DSN=A123456.CITIES.IN1,DISP=SHR
//CITIES2 DD DSN=A123456.CITIES.IN2,DISP=SHR
//JNF1CNTL DD *
  OMIT COND=(21,15,CH,EQ,C'VERMONT')
//JNF2CNTL DD *
  OMIT COND=(1,15,CH,EQ,C'VERMONT')
//BOTH DD DSN=A123456.BOTH.OUT,DISP=OLD
//F1ONLY DD DSN=A123456.F1ONLY.OUT,DISP=OLD
//F2ONLY DD DSN=A123456.F2ONLY.OUT,DISP=OLD
//SYSIN DD *
  JOINKEYS F1=CITIES1,FIELDS=(21,15,A,1,20,A),SORTED
  JOINKEYS F2=CITIES2,FIELDS=(1,15,A,16,20,A)
  JOIN UNPAIRED,F1,F2
  REFORMAT FIELDS=(?,F1:1,37,F2:1,35)
  OPTION COPY
  OUTFIL FNAMES=BOTH,INCLUDE=(1,1,CH,EQ,C'B'),
    BUILD=(2,37)
  OUTFIL FNAMES=F1ONLY,INCLUDE=(1,1,CH,EQ,C'1'),
    BUILD=(2,37)
  OUTFIL FNAMES=F2ONLY,INCLUDE=(1,1,CH,EQ,C'2'),
    BUILD=(54,20,39,15)
/*
```

The first JOINKEYS statement identifies CITIES1 as the ddname for the F1 data set and indicates ascending keys (State and City) in positions 21-35 and 1-20 of that data set. The JNF1CNTL data set contains an OMIT statement to remove the VERMONT records from the F1 data set so they will not be joined. Since the F1 records are already in order by the State and City fields, SORTED is used to do a Copy rather than a Sort for the F1 data set.

The second JOINKEYS statement identifies CITIES2 as the ddname for the F2 data set and indicates ascending keys (State and City) in positions 1-15 and 16-35 of that data set. The JNF2CNTL data set contains an OMIT statement to remove the VERMONT records from the F2 data set so they will not be joined. Since the F2 records are not already in order by the State and City fields, SORTED is not used and a Sort is performed for the F2 data set.

Since we want to separate out the BOTH, F1ONLY and F2ONLY joined records, we use a JOIN statement with UNPAIRED,F1,F2 to keep the unpaired joined records as well as the paired join records. In the REFORMAT statement, we use ? as the first field to give us an indicator of whether each key was found in both records ('B' indicator), only in F1 ('1' indicator) or only in F2 ('2' indicator). After the indicator, we put positions 1-37 (City, State, District) from F1 and 1-35 (State, City) from F2. After the records are joined, they will look as shown in [Table 56 on page 105](#).

Table 56. Joined City records

Indicator	F1 City	F1 State	F1 District	F2 State	F2 City
1 1	2 21	22 36	37 38	39 53	54 73

Table 56. Joined City records (continued)

Indicator	F1 City	F1 State	F1 District	F2 State	F2 City
B	GILROY	CALIFORNIA	05	CALIFORNIA	GILROY
B	GILROY	CALIFORNIA	10	CALIFORNIA	GILROY
2				CALIFORNIA	LOS ANGELES
2				CALIFORNIA	MODESTO
1	MORGAN HILL	CALIFORNIA	03		
1	PALO ALTO	CALIFORNIA	15		
1	PALO ALTO	CALIFORNIA	08		
1	PALO ALTO	CALIFORNIA	21		
B	SACRAMENTO	CALIFORNIA	05	CALIFORNIA	SACRAMENTO
B	SAN JOSE	CALIFORNIA	02	CALIFORNIA	SAN JOSE
B	SAN JOSE	CALIFORNIA	10	CALIFORNIA	SAN JOSE
1	SAN MARTIN	CALIFORNIA	12		
B	AUSTIN	TEXAS	21	TEXAS	AUSTIN
1	DALLAS	TEXAS	15		
1	DALLAS	TEXAS	25		
2				TEXAS	LAREDO

Now we can use `OUTFIL` statements to create our three different output data sets. We write positions 2-38 (F1 City, F1 State, F1 District) of the joined records with an indicator of 'B' to positions 1-37 of the `BOTH.OUT` data set. We write positions 2-38 (F1 City, F1 State, F1 District) of the joined records with an indicator of '1' to positions 1-37 of the `F1ONLY.OUT` data set. We write positions 54-73 (F2 City) and 39-53 (F2 State) of the joined records with an indicator of '2' to positions 1-35 of the `F2ONLY.OUT` data set. The resulting records are shown in [Table 57 on page 106](#), [Table 58 on page 107](#) and [Table 59 on page 107](#).

Table 57. BOTH.OUT data set

City	State	District
1 20	21 35	36 37
GILROY	CALIFORNIA	05
GILROY	CALIFORNIA	10
SACRAMENTO	CALIFORNIA	05
SAN JOSE	CALIFORNIA	02
SAN JOSE	CALIFORNIA	10
AUSTIN	TEXAS	21

Table 58. F1ONLY.OUT data set

City	State	District
1 20	21 35	36 37
MORGAN HILL	CALIFORNIA	03
PALO ALTO	CALIFORNIA	15
PALO ALTO	CALIFORNIA	08
PALO ALTO	CALIFORNIA	21
SAN MARTIN	CALIFORNIA	12
DALLAS	TEXAS	15
DALLAS	TEXAS	25

Table 59. F2ONLY.OUT data set

City	State
1 20	21 35
LOS ANGELES	CALIFORNIA
MODESTO	CALIFORNIA
LAREDO	TEXAS

These are just a few of the many types of join operations you can do with JOINKEYS. See [z/OS DFSORT Application Programming Guide](#) for complete details of using the JOINKEYS, JOIN and REFORMAT statements along with the other DFSORT statements to perform different JOINKEYS applications.

Summary

This topic covered the use of JOINKEYS, JOIN and REFORMAT to join records from two data sets.

The next topic will cover methods of calling DFSORT from a program.

Chapter 9. Calling DFSORT from a program

This chapter contains Programming Interface information.

In addition to processing your DFSORT program control statements directly with PGM=SORT or PGM=ICEMAN, you can call DFSORT from COBOL, PL/I, Assembler, or other programs. In this chapter, you will concentrate on sorting and merging using COBOL and sorting using PL/I. The examples in this chapter assume that the COBOL environment is available.

For information on restrictions when using these languages and on calling DFSORT from an assembler program, see [z/OS DFSORT Application Programming Guide](#).

Passing control statements

When you use COBOL or PL/I to call DFSORT, these program products create and pass a SORT or MERGE control statement and a RECORD control statement for you. You can use other DFSORT control statements, such as INCLUDE, OMIT, SUM, INREC, OUTREC and OUTFIL with your COBOL or PL/I programs. Use a SORTCNTL or DFSPARM data set for these DFSORT statements in the same way you would use a SYSIN data set. For example, you can use the SORTCNTL DD statement to pass the INCLUDE control statement that selects only the English department books:

```
//EXAMP    JOB  A492,PROGRAMMER
:
:
:
//SORTCNTL DD  *
           INCLUDE COND=(110,5,CH,EQ,C'ENGL')
/*
```

Either a SYSIN data set or a DFSPARM data set, or both, can be used for DFSORT control statements when DFSORT is invoked directly, that is, by PGM=SORT or PGM=ICEMAN. Either a SORTCNTL data set or a DFSPARM data set, or both, can be used for DFSORT control statements when DFSORT is called from a program.

If you use DFSORT with COBOL, you should be familiar with the SORT-CONTROL and SORT-RETURN special registers. For complete details, see the COBOL Programmer's Guide that describes the compiler version available at your site.

Calling DFSORT from a COBOL program

To call DFSORT from a COBOL program, use the COBOL statements SORT and MERGE. This section shows sample programs that use the COBOL SORT and MERGE statements. For complete information, see the COBOL Programmer's Guide describing the compiler version available at your site.

Sorting records

The sample COBOL program shown in this section calls DFSORT to sort the bookstore master file (MASTER-FILE) by title in ascending order. The sorted master file is written to SORTED-MASTER-FILE.

Following is the JCL that calls the sample COBOL program:

```
//EXAMP    JOB  A492,PROGRAMMER
//BOOKS    EXEC PGM=COBOLPGM
//STEPLIB  DD   DSN=USER.PGMLIB,DISP=SHR
//SYSOUT   DD   SYSOUT=A
//MASTIN   DD   DSN=A123456.MASTER,DISP=OLD
//MASTOUT  DD   DSN=A123456.OUTB,DISP=(NEW,CATLG,DELETE),
//          SPACE=(CYL,(1,1)),UNIT=SYSDA
//PRINTFL  DD   SYSOUT=A
```

In contrast to the JCL for executing DFSORT by using a JCL EXEC statement with PGM=SORT or PGM=ICEMAN (see [“JCL for sorting data sets directly”](#) on page 16) the previous JCL has these differences:

- The program name on the EXEC statement is that of the COBOL program.
- The STEPLIB DD statement defines the library containing the COBOL program.
- The name of the DD statement for the input file need not be SORTIN.
- The name of the DD statement for the output file need not be SORTOUT.

Notice that the control field and order for the sort are specified in the COBOL program itself rather than with a SORT control statement. The following shows the sample COBOL program.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    COBOLPGM.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT SD-FILE ASSIGN TO
        DUMMYNM.
    SELECT MASTER-FILE ASSIGN TO
        MASTIN.
    SELECT SORTED-MASTER-FILE ASSIGN TO
        MASTOUT.
    SELECT PRINT-FILE ASSIGN TO
        PRINTFL.
DATA DIVISION.
FILE SECTION.
SD SD-FILE
    DATA RECORD IS SD-RECORD.
01 SD-RECORD.
    05 TITLE-IN    PICTURE X(75).
    05 AUTH-LN-IN  PICTURE X(15).
    05 AUTH-FN-IN  PICTURE X(15).
    05 PUB-IN      PICTURE X(4).
    05 COUR-DEPT-IN PICTURE X(5).
    05 COUR-NO-IN  PICTURE X(5).
    05 COUR-NAM-IN PICTURE X(25).
    05 INST-LN-IN  PICTURE X(15).
    05 INST-INIT-IN PICTURE X(2).
    05 NO-STOCK-IN  PICTURE 9(8) BINARY.
    05 NO-SOLD-IN  PICTURE 9(8) BINARY.
    05 PRICE-IN    PICTURE 9(8) BINARY.
FD MASTER-FILE
    DATA RECORD IS MASTER-RECORD.
01 MASTER-RECORD.
    05 FILLER      PICTURE X(173).

FD SORTED-MASTER-FILE
    DATA RECORD IS SORTED-MASTER-RECORD.
01 SORTED-MASTER-RECORD.
    05 FILLER      PICTURE X(173).

FD PRINT-FILE
    DATA RECORD IS OUTPUT-REPORT-RECORD.
01 OUTPUT-REPORT-RECORD.
    05 REPORT-OUT  PICTURE X(120).
.
.
.
PROCEDURE DIVISION.
.
.
.
SORT-ROUTINE SECTION.
    SORT SD-FILE
    ASCENDING KEY TITLE-IN
    USING MASTER-FILE
    GIVING SORTED-MASTER-FILE.
    IF SORT-RETURN > 0
    DISPLAY "SORT FAILED".
.
.
.
SORT-REPORT SECTION.
    print a report on PRINT-FILE using SORTED-MASTER-FILE.
```

```

.
.
.
STOP RUN.

```

Merging records

The sample COBOL program in [Figure 2 on page 112](#) calls DFSORT to merge the presorted bookstore master file (MASTER-FILE) with another presorted file (NEW-BOOKS-FILE) to create a new master file (MERGED-FILE).

The JCL for the program is as follows:

```

//EXAMP    JOB    A492,PROGRAMMER
//BOOKS    EXEC   PGM=COBOLP
//STEPLIB  DD     DSN=USER.PGMLIB,DISP=SHR
//SYSOUT   DD     SYSOUT=A
//MASTERFL DD     DSN=A123456.MASTER,DISP=OLD
//NEWBOOKS DD    DSN=A123456.NEW,DISP=OLD
//MERGEDFL DD     DSN=A123456.OUTC,DISP=(NEW,CATLG,DELETE),
//          SPACE=(CYL,(1,1)),UNIT=SYSDA

```

[Figure 2 on page 112](#) shows the sample COBOL program.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.  
    COBOLP.  
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT SD-FILE ASSIGN TO  
        DUMMYNM.  
    SELECT MASTER-FILE ASSIGN TO  
        MASTERFL.  
    SELECT NEW-BOOKS-FILE ASSIGN TO  
        NEWBOOKS.  
    SELECT MERGED-FILE ASSIGN TO  
        MERGEDFL.  
DATA DIVISION.  
FILE SECTION.  
SD SD-FILE  
    DATA RECORD IS SD-RECORD.  
01 SD-RECORD.  
    05 TITLE-KEY    PICTURE X(75).  
    05 FILLER       PICTURE X(98).  
  
FD MASTER-FILE  
    DATA RECORD IS MASTER-RECORD.  
01 MASTER-RECORD.  
    05 FILLER       PICTURE X(173).  
  
FD NEW-BOOKS-FILE  
    DATA RECORD IS NEW-BOOKS-RECORD.  
01 NEW-BOOKS-RECORD.  
    05 FILLER       PICTURE X(173).  
  
FD MERGED-FILE  
    DATA RECORD IS MERGED-RECORD.  
01 MERGED-RECORD.  
    05 FILLER       PICTURE X(173).  
.  
.  
.  
  
PROCEDURE DIVISION.  
.  
.  
.  
  
MERGE-ROUTINE SECTION.  
    MERGE SD-FILE  
        ASCENDING KEY TITLE-KEY  
        USING MASTER-FILE NEW-BOOKS-FILE  
        GIVING MERGED-FILE.  
    IF SORT-RETURN > 0  
        DISPLAY "MERGE FAILED".  
    STOP RUN.
```

Figure 2. Sample COBOL Program with MERGE Commands

Sorting with COBOL FASTSRT

With the COBOL program for sorting records in “Sorting records” on page 109, the input (from MASTER-FILE) and the output (to SORTED-MASTER-FILE) qualify for the COBOL FASTSRT option. With this compile-time FASTSRT option, your sort runs considerably faster, because DFSORT does the input and output processing, rather than COBOL. For complete details on FASTSRT, refer to [z/OS DFSORT Application Programming Guide](#) and the COBOL Programmer's Guide that describes the compiler version available at your site.

Note: COBOL evaluates sort input and output independently to see if it qualifies for FASTSRT. If either the input or the output of your sort does not qualify because of the presence of an input or output procedure, you might be able to replace such a procedure and use DFSORT control statements to accomplish the

same thing. For example, you can use a control statement (OUTREC) to indicate how records will be reformatted before being written to the output data set.

Calling DFSORT from a PL/I program

When calling DFSORT, a PL/I program must pass a SORT control statement, a RECORD control statement, and the amount of main storage to be available to DFSORT. On the RECORD control statement, you specify the record type and length for the input data set. Following the RECORD control statement, you specify the main storage value for DFSORT either as MAX, or as a value in bytes. (DFSORT performs best when MAX is specified as the main storage value.)

You can also specify DFSORT control statements in a SORTCNTL or DFSPARM data set.

This JCL is for the program shown in [Figure 3 on page 114](#). A SORTCNTL data set is used to specify an INCLUDE control statement that selects only the English department books.

```
//EXAMP    JOB    A492,PROGRAMMER
//BOOKS    EXEC   PGM=PLIPGM
//STEPLIB  DD     DSN=USER.PGMLIB,DISP=SHR
//SYSOUT   DD     SYSOUT=A
//SORTIN   DD     DSN=A123456.SORT.SAMPIN,DISP=SHR
//SORTOUT  DD     DSN=A123456.SORT.SAMPOUT,DISP=OLD
//SORTCNTL DD     *
           INCLUDE COND=(110,5,CH,EQ,C'ENGL')
/*
//SYSPRINT DD     SYSOUT=A
```

The sample PL/I program shown in [Figure 3 on page 114](#) calls DFSORT to sort the bookstore file by title. It specifies appropriate SORT and RECORD control statements, and a DFSORT main storage value of MAX.

```
PLIPGM:  PROC OPTIONS(MAIN);

        DCL 1 MASTER_RECORD,
          5 TITLE_IN   CHAR(75),
          5 AUTH_LN_IN  CHAR(20),
          .
          .
          .
          5 PRICE_IN    BIN FIXED(31);

        DCL RETURN_CODE FIXED BIN(31,0);
        DCL MAXSTOR FIXED BIN(31,0);
        UNSPEC(MAXSTOR)='00000000'B||UNSPEC('MAX');
        .
        .
        .
        CALL PLISRTA (' SORT FIELDS=(1,75,CH,A) ',
          ' RECORD TYPE=F,LENGTH=(173) ',
          MAXSTOR,
          RETURN_CODE);
        IF RETURN_CODE ^= 0 THEN DO;
          PUT SKIP EDIT ('SORT FAILED')(A);
          CALL PLIRETC(RETURN_CODE);
        END;
        .
        .
        .
        CALL OUTPUT;
        .
        .
        .
        OUTPUT: PROCEDURE;
        .
        .
        .
        . Print a report from the sorted master file (SORTOUT)
        .
        .
        .
        END;
END PLIPGM;
```

Figure 3. Sample PL/I Program with SORT Commands

Summary

Methods of calling DFSORT from COBOL and PL/I were discussed.

Chapter 10. Overriding installation defaults

IBM ships DFSORT with pre-set defaults. During installation, your system programmer can change these defaults. For example, one IBM-supplied default is to list DFSORT control statements in the DFSORT message data set. However, at your site, the default might be to not list DFSORT control statements.

Furthermore, your system programmer can establish separate defaults for jobs with DFSORT executed directly and for jobs with DFSORT called from a program. So, when you execute DFSORT directly, the default might be to list DFSORT control statements, and when you call DFSORT from a program, the default might be to not list DFSORT control statements.

In many cases, if a particular installation default is not appropriate for your job, you can temporarily override it in one of several ways:

- as a PARM parameter in a DFSPARM data set
- as an OPTION, ALTSEQ or DEBUG parameter in a DFSPARM data set
- as a PARM parameter on the JCL EXEC statement
- as an OPTION, ALTSEQ or DEBUG parameter in a SYSIN or SORTCNTL data set
- as an OPTION, ALTSEQ or DEBUG parameter in a parameter list passed from a calling program

OPTION, ALTSEQ and DEBUG parameters in a DFSPARM data set are a good way to override installation defaults regardless of whether you execute DFSORT directly or call it from a program.

In this chapter, you will learn how to override some of the many available defaults, concentrating on PARM parameters in the JCL EXEC statement, and the OPTION control statement in a DFSPARM data set.

You can list all of the installation defaults selected at your site using an ICETOOL DEFAULTS job like the following:

```
//S1 EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=A
//DFSMSG DD SYSOUT=A
//DFLTS DD SYSOUT=A
//TOOLIN DD *
      DEFAULTS LIST(DFLT)
/*
```

You'll learn more about ICETOOL in [Chapter 12, "Using the ICETOOL utility,"](#) on page 123.

For detailed information on all of the parameters you can override and the various ways in which you can override them, see *z/OS DFSORT Application Programming Guide*. In particular, see Appendix C, "Specification/Override of DFSORT Options".

Specifying PARM parameters on a JCL EXEC statement

When you execute DFSORT directly, you can use the PARM parameter in the JCL EXEC statement to override certain defaults for your job. For example, if the default at your site is to list DFSORT statements (installation parameter LIST=YES), but you do not want them listed for your job, you can specify NOLIST in the PARM field

```
//S1 EXEC PGM=DFSORT,PARM='NOLIST'
```

On the other hand, if the default is not to list the DFSORT statements (installation parameter LIST=NO), but you do want them listed, you can specify LIST in the PARM field:

```
//S1 EXEC PGM=DFSORT,PARM='LIST'
```

You can also specify more than one PARM parameter. For example, if you want to list the DFSORT statements, have DFSORT set a return code of 4 if SUM fields overflow, use six dynamically allocated SYSDA work data sets, and have DFSORT ABEND if it detects an error, you can specify:

```
//S1 EXEC PGM=SORT,PARM='LIST,OVFLO=RC4,DYNALLOC=(SYSDA,6),ABEND'
```

Specifying an OPTION control statement in DFSPARM

Whether you execute DFSORT directly or call it from a program, you can use an OPTION statement in a DFSPARM data set to override certain defaults for your job. To do so, you can place the OPTION statement after a //DFSPARM DD * statement, either by itself or with other DFSORT control statements.

A particular default that you may want to override with the OPTION statement is the EQUALS=NO installation option that specifies whether equally collating records are to be written in their original order.

If your site has EQUALS=NO as the default (recommended) and you want to temporarily override it (so that equally collating records are written in their original order for **your** job), you can specify EQUALS on the OPTION statement:

```
//DFSPARM DD *  
  OPTION EQUALS  
/*
```

Or, if your site has established EQUALS=YES as the installation default and you want to temporarily override it (so that equally collating records can be written in random order for your job), you can specify NOEQUALS on the OPTION statement:

```
//DFSPARM DD *  
  OPTION NOEQUALS  
/*
```

You can also specify more than one parameter on the OPTION statement, as well as additional control statements, in a DFSPARM data set. For example, if you want to list the DFSORT statements, have DFSORT set a return code of 4 if SUM fields overflow, use six dynamically allocated SYSDA work data sets, and have DFSORT ABEND if it detects an error, you can specify:

```
//DFSPARM DD *  
  OPTION LIST,OVFLO=RC4,DYNALLOC=(SYSDA,6)  
  DEBUG ABEND  
/*
```

Summary

This chapter covered methods of overriding selected DFSORT installation defaults using PARM parameters on the JCL EXEC statement, or an OPTION control statement.

Chapter 11. Using DFSORT efficiently

You will get the best performance from DFSORT if you follow these guidelines:

- Be generous with main storage.
- Allow memory object sorting, hipersorting and dataspace sorting.
- Use high-speed disks for work space.
- Eliminate unnecessary fields with INREC.
- Eliminate unnecessary records with INCLUDE or OMIT.
- Eliminate unnecessary records with STOPAFT and SKIPREC.
- Consolidate records with SUM.
- Create multiple output data sets with OUTFIL
- Replace program logic with DFSORT control statements.
- Use FASTSORT with COBOL.
- Avoid options that might degrade performance.

Additional suggestions can be found in [z/OS DFSORT Application Programming Guide](#) and [z/OS DFSORT Tuning Guide](#).

Be generous with main storage

By default, DFSORT can use 6 megabytes of main memory for sort, merge and copy applications, or even more when appropriate. If your site does not have an installation default of SIZE=MAX, we recommend setting the MAINSIZE=MAX parameter for your job. As an example, you can set MAINSIZE=MAX like this:

```
//DFSPARM DD *
        OPTION MAINSIZE=MAX
/*
```

Alternatively, you can specify the MAINSIZE=*n*M parameter (*n* megabytes) to give DFSORT more or less storage for your job, but we do not recommend this unless you have a specific reason for using MAINSIZE=*n*M rather than MAINSIZE=MAX. As an example, you can set MAINSIZE=2M like this:

```
//DFSPARM DD *
        OPTION MAINSIZE=2M
/*
```

Allow memory object sorting, Hipersorting and dataspace sorting

By default, DFSORT can use memory objects, hiperspaces or data spaces, when appropriate, instead of or along with disk work data sets, to improve the performance of sort applications. If your site has an installation default of MOSIZE=0, HIPRMAX=0 or DSPSIZE=0, you might want to specify the MOSIZE=MAX, HIPRMAX=OPTIMAL and DSPSIZE=MAX parameters when you sort large data sets. As an example, you can set these parameters like this:

```
//DFSPARM DD *
        OPTION MOSIZE=MAX, HIPRMAX=OPTIMAL, DSPSIZE=MAX
/*
```

Use high-speed disks for work data sets

Using high-speed disks, such as IBM's Enterprise Storage Server subsystems, offers the best performance for work data sets. Throughput can be further improved by leveraging high speed channels such as IBM's

FICON connectivity solutions. DFSORT is optimized to leverage the high speed data transfer capabilities offered by these technologies. You should avoid using tapes for work data sets whenever possible.

Eliminate unnecessary fields with INREC

If you need to reformat your records, using INREC to significantly shorten them can result in faster processing.

Remember that INREC reformats records before they are processed, and OUTREC reformats them after they are processed. Therefore, you should use INREC to shorten records and OUTREC to lengthen records. For a summary of the control statements and the corresponding record positions to refer to when using INREC, see [Table 60 on page 118](#).

Table 60. Control statement and corresponding records with INREC

Control Statement	Original Records	Reformatted Records
SORT		✓
MERGE		✓
SUM		✓
OUTREC		✓
OUTFIL		✓
INCLUDE	✓	
OMIT	✓	

For details on INREC, see [Chapter 5, “Reformatting records with fixed fields,” on page 37](#).

Eliminate unnecessary records with INCLUDE or OMIT

Naturally, the number of input records affects the amount of time processing will take. The fewer the records, the faster the DFSORT application. You can improve performance by using INCLUDE or OMIT whenever possible to select only the records pertaining to your application.

For details on INCLUDE and OMIT, see [Chapter 3, “Including or omitting records,” on page 23](#).

Eliminate unnecessary records with STOPAFT and SKIPREC

You can also use the STOPAFT and SKIPREC options to reduce the number of input records you process.

- Use STOPAFT to specify the maximum number of records to be accepted for sorting or copying.
- Use SKIPREC to specify the number of records to be skipped before sorting or copying begins.

For information on how to use these options, see [z/OS DFSORT Application Programming Guide](#).

Consolidate records with SUM

You can improve performance for sorting or merging by using the SUM statement, if appropriate for your job, to:

- add the contents of fields whenever two records with equal control fields are found. DFSORT places the result in one record and deletes the other, reducing the number of output records, or
- delete records with duplicate control fields by specifying FIELDS=NONE.

For details on SUM, see [Chapter 4, “Summing records,” on page 33](#).

Create multiple output data sets with 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.

For details on OUTFIL, see [Chapter 7, “Creating multiple output data sets and reports,” on page 79](#).

Replace program logic with DFSORT control statements

As a rule, using appropriate DFSORT control statements is more efficient than program logic. Whenever possible, use DFSORT control statements INCLUDE, OMIT, INREC, OUTREC, SUM and OUTFIL instead of doing the same functions with program code.

Use FASTSRT with COBOL

With COBOL, you can enhance DFSORT performance by using the FASTSRT compile-time option. With FASTSRT, DFSORT does the input and output processing, rather than COBOL. For more information on the FASTSRT option, see the COBOL Programmer's Guide that describes the compiler version available at your site.

Avoid options that might degrade performance

This chapter covered methods for improving DFSORT performance, including the effective use of various control statements and parameters.

The following parameters and options might adversely affect DFSORT performance. Use them only when necessary. For a complete description of what these parameters and options are and how they affect performance, see [z/OS DFSORT Application Programming Guide](#) and [z/OS DFSORT Tuning Guide](#).

- VERIFY parameter
- EQUALS parameter
- NOBLKSET parameter
- CKPT parameter
- EQUCOUNT parameter
- NOCINV parameter
- LOCALE parameter
- BSAM parameter
- NOASSIST parameter
- NOCFW parameter
- Tape work data sets
- User exit routines
- EFS program
- Dynamic link-edit of user exit routines
- Small values for the MOSIZE, HIPRMAX, DSPSIZE, or MAINSIZE parameters
- Processing Unicode data formats

Part 3. Learning to use ICETOOL

Chapter 12. Using the ICETOOL utility

ICETOOL is a multipurpose DFSORT utility that uses the capabilities of DFSORT to perform multiple operations on one or more data sets in a single step.

This chapter introduces you to ICETOOL's 17 "operators", which allow you to do a wide variety of tasks. You will learn about ICETOOL's JCL and control statements while writing a large "main" ICETOOL job that uses many of the ICETOOL operators, as well as several additional smaller ICETOOL jobs that illustrate specific points. To fully use the capabilities of ICETOOL, you should become familiar with all of its operators, operands, and methods of invocation, as described in *z/OS DFSORT Application Programming Guide*.

Note: Most likely, you would do unrelated tasks in different ICETOOL jobs. But you can do unrelated tasks in the same ICETOOL job. The ICETOOL jobs constructed in this chapter take both approaches as a convenient way to illustrate the use of various ICETOOL operators.

ICETOOL operators

The following 17 ICETOOL operators can be used to perform a variety of functions. By using various combinations of the 17 ICETOOL operators, you can easily create applications that perform many complex tasks.

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 (for example, empty, not empty, less than, equal to, or greater than 5000 records, and so on).

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, which 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).

RANGE

Prints a message containing the count of values in a specified range for a specified numeric field in a data set.

RESIZE

Creates a larger record from multiple shorter records, or creates multiple shorter records from a larger record, that is, resizes fixed length records.

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, relative records or trailer 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.

Sample input data sets

Each ICETOOL operator (except DEFAULTS and MODE) requires an input data set. The input data set used by one operator can be the same or different from the input data set used by another operator. Thus, ICETOOL can process many data sets in a single step.

This chapter uses the branch data set SORT.BRANCH, the bookstore data set SORT.SAMPIN, and the additional bookstore data set SORT.SAMPADD as input data sets. See [Appendix A, "Creating the sample data sets," on page 171](#) and [Appendix B, "Descriptions of the sample data sets," on page 173](#) for additional information about these sample data sets. Two temporary data sets created by ICETOOL from SORT.BRANCH are also used as input.

Note: This chapter also uses data sets other than SORT.BRANCH, SORT.SAMPIN and SORT.SAMPADD. 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.

Writing required JCL statements

An ICETOOL job consists of:

1. The JCL statements that are required for every ICETOOL job.
2. The operator statements indicating the operations to be performed by the ICETOOL job.
3. The JCL statements that are required as a result of the specified operator statements.

The first step in creating any ICETOOL job is to write the JCL that is always required. Here is a skeleton of the JCL for an ICETOOL job:


```
//EXAMP    JOB    A492,PROGRAMMER
//TOOL     EXEC   PGM=ICETOOL
//TOOLMSG  DD     SYSOUT=A
//DFSMSG   DD     SYSOUT=A
//TOOLIN   DD     *
<ICETOOL statements go here>
/*
<Additional JCL statements go here>
```

- The JOB statement signals the beginning of the job.
- The EXEC statement signals the beginning of the job step and tells the operating system to run the ICETOOL program.
- The TOOLMSG statement defines the output data set for ICETOOL messages.
- The DFSMSG statement defines the output data set for DFSORT messages.
- The TOOLIN statement precedes the ICETOOL statements (comment, blank, and operator statements). The ICETOOL statements you write must appear after TOOLIN. The additional JCL statements you write can appear before the TOOLIN statement or after the ICETOOL statements. In the main ICETOOL job, the additional JCL statements will be placed after the ICETOOL statements.

ICETOOL comment and blank statements

Comment statements and blank statements can be placed anywhere among the ICETOOL operator statements.

- Comment statements start with an asterisk (*) in column 1 and are printed along with the ICETOOL operator statements.
- Blank statements contain blanks in columns 1-72 and are ignored because ICETOOL prints blank lines where appropriate.

A blank statement and a comment statement for our example looks like this:

```
* Statistics from all branches
```

Here are the steps for writing these statements:

Table 61. Steps to Create a Blank Statement and a Comment Statement

Step	Action
1	After the TOOLIN DD statement, skip one line.
2	Type an asterisk (*) in column 1 followed by the comment.

When complete, the TOOLIN statements look like this:

```
//TOOLIN DD *
* Statistics from all branches
/*
```

For the main ICETOOL job, a comment statement will be placed before each operator to describe its function. Although not required, this is a good practice to follow.

So far

So far, you have been introduced to the basics of the ICETOOL utility. Now, using the following tutorials, you can learn about many of the ICETOOL operators. The following sections contain parts of the main ICETOOL job, so that by the end of the chapter, you will have created the complete main ICETOOL job. At the end of the chapter, there is a section that contains the complete main ICETOOL job and its resulting messages.

Printing statistics for numeric fields

When working with data sets containing numeric fields, you may want statistical information about one or more of those fields. You can use the STATS operator to find the minimum, maximum, average, and total values of up to 10 specific numeric fields.

A STATS operator that prints statistics for the employees, profit, and revenue fields of the branch office data set looks like this:

STATS FROM(ALL) ON(18,4,ZD) ON(28,6,PD) ON(22,6,PD)

Revenue
Profit
Employees
ddname of input data set

Here are the steps for writing this STATS operator.

Table 62. Steps to Create the STATS Operator

Step	Action
1	Type STATS after the comment statement (you can leave one or more blanks before STATS if you like).
2	Leave at least one blank and type FROM(ALL) FROM specifies the ddname (that is, the name of the DD statement) for the input data set from which you want to print statistics. In this case ALL is the ddname chosen, but you can use any valid 1-8 character ddname you like.
3	Leave at least one blank and type ON ON defines a field for which you want to print statistics.
4	Type in parentheses, and separated by commas: <ol style="list-style-type: none">Where the employees field begins relative to the beginning of the input record (the first position is byte 1). The employees field begins at byte 18.The length of the employees field in bytes. The employees field is 4 bytes long.A code for the data format. The employees field contains zoned decimal data, which you specify as ZD.
5	Leave at least one blank and type ON ON defines another field for which you want to print statistics. You can print statistics for up to 10 fields with one STATS statement. Specify the ON fields in the same order in which you want their statistics to be printed.
6	Type in parentheses, and separated by commas the location (28), length (6), and format (PD for packed decimal) of the profit field.
7	Leave at least one blank and type ON . Type in parentheses and separated by commas, the location (22), length (6), and format (PD) of the revenue field. Make sure that the statement is coded between columns 1 and 72.

You must also write a DD statement for the A123456.SORT.BRANCH data set using the ddname ALL and place it at the end of the job:

```
//ALL DD DSN=A123456.SORT.BRANCH,DISP=SHR
```

When complete the TOOLIN statements and ALL statement look like this:

```
//TOOLIN DD *  
  
* Statistics from all branches  
STATS FROM(ALL) ON(18,4,ZD) ON(28,6,PD) ON(22,6,PD)  
/*  
//ALL DD DSN=A123456.SORT.BRANCH,DISP=SHR
```

When this STATS operation is run, the results are placed in the TOOLMSG data set. If you ran the ICETOOL job you created so far, the TOOLMSG output would look like this:

```
ICE600I 0 DFSORT ICETOOL UTILITY RUN STARTED  
  
ICE650I 0 VISIT http://www.ibm.com/storage/dfsor FOR ICETOOL PAPERS, EXAMPLES AND MORE  
  
ICE632I 0 SOURCE FOR ICETOOL STATEMENTS:  TOOLIN  
  
ICE630I 0 MODE IN EFFECT:  STOP  
  
      * Statistics from all branches  
      STATS FROM(ALL) ON(18,4,ZD) ON(28,6,PD) ON(22,6,PD)  
ICE627I 0 DFSORT CALL 0001 FOR COPY FROM ALL      TO E35 EXIT COMPLETED  
ICE628I 0 RECORD COUNT:  000000000000012  
ICE607I 0 STATISTICS FOR (18,4,ZD)      :  
ICE608I 0   MINIMUM: +000000000000015, MAXIMUM:  +000000000000035  
ICE609I 0   AVERAGE: +000000000000024, TOTAL  :  +000000000000298  
ICE607I 0 STATISTICS FOR (28,6,PD)      :  
ICE608I 0   MINIMUM: -000000000004278, MAXIMUM:  +000000000008276  
ICE609I 0   AVERAGE: +000000000004222, TOTAL  :  +0000000000050665  
ICE607I 0 STATISTICS FOR (22,6,PD)      :  
ICE608I 0   MINIMUM: +0000000000012300, MAXIMUM:  +0000000000042820  
ICE609I 0   AVERAGE: +0000000000027469, TOTAL  :  +00000000000329637  
ICE602I 0 OPERATION RETURN CODE:  00  
  
ICE601I 0 DFSORT ICETOOL UTILITY RUN ENDED - RETURN CODE:  00
```

Looking at the output, you will notice that:

- Message ICE628I gives the count of records processed.
- Messages ICE607I, ICE608I, and ICE609I give the numerical statistics for each ON field specified in the order in which they were specified.
- A return code for each operator is given in message ICE602I and the highest operator return code is given in message ICE601I.

Continuing an operator statement

If you cannot fit your STATS statement (or any other ICETOOL operator statement) between columns 1 and 72 of a single line, you can continue it across multiple lines. If you end a line with a hyphen (-) after the operator or any operand, the next line is treated as a continuation. Any characters specified after the hyphen are ignored.

Note that the operator and each operand must be completely specified on one line (between columns 1 and 72).

For example:

```
STATS      - this is the operator  
FROM(ALL) - ALL is the ddname for SORT.BRANCH  
ON(18,4,ZD) -  
ON(28,6,PD) -  
ON(22,6,PD)
```

Statistics for VB data set record lengths

When working with variable length record data sets, you can use the STATS operator to easily obtain the following information:

- The shortest record in the data set (minimum)
- The longest record in the data set (maximum)

- The average length of records in the data set (average)
- The total number of bytes in the data set (total)

ICETOOL provides the special ON(VLEN) field for printing these statistics. Specify ON(VLEN) as you would any other ON field.

When you code ON(p,m,f) fields for VB records, remember to add 4 to the starting position to account for the 4-byte RDW, in the same way discussed earlier for DFSORT control statement fields.

The following STATS operator reports the minimum, maximum, total and average values for the record length and for a PD field in the first two data bytes:

```
STATS FROM(VBIN) ON(VLEN) ON(5,2,PD)
```

So far

Now you know how to use ICETOOL's STATS operator to print statistics for numeric fields and VB record lengths. In the next section, you will learn how to use ICETOOL's SORT operator.

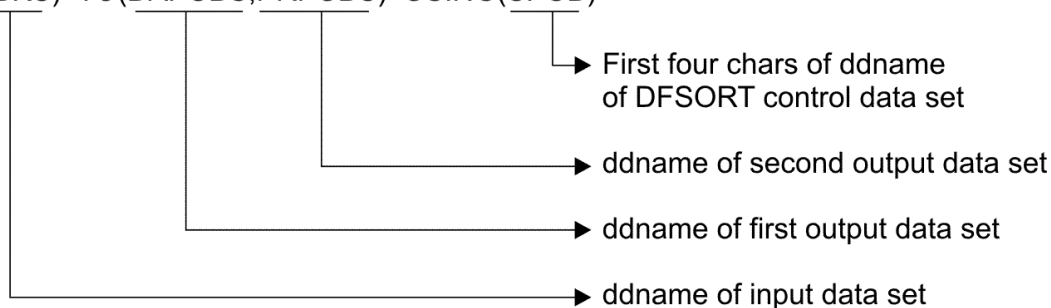
Creating identical sorted data sets

You can use ICETOOL's SORT operator to create sorted output data sets. A single SORT operator can be used to create one output data set or up to 10 identical output data sets. Using INCLUDE or OMIT statements, you can select a subset of the input records. Using INREC or OUTREC statements, you can rearrange the fields of the input records. Using OUTFIL statements, you can create any number of output data sets with different subsets of records or arrangements of fields.

For this example, we will use both the sample bookstore data set (SORT.SAMPIN) and the additional bookstore data set (SORT.SAMPADD) as input.

A SORT operator that selects the books from publishers VALD and WETH, sorts them by publisher and title, and writes them to disk and print data sets, looks like this:

```
SORT FROM(BKS) TO(DAPUBS,PRPUBS) USING(SPUB)
```



Here are the steps for writing this SORT operator:

Table 63. Steps to Create the SORT Operator

Step	Action
1	Write a comment statement (optional): <pre>* Books from VALD and WETH</pre>
2	Type SORT after the comment statement
3	Leave at least one blank and type FROM(BKS) BKS specifies the ddname for the input data sets you want to sort.

Table 63. Steps to Create the SORT Operator (continued)

Step	Action
4	<p>Leave at least one blank and type TO(DAPUBS,PRPUBS)</p> <p>TO specifies the ddnames for the output data sets to contain the sorted subset of records. You can create up to 10 identical output data sets of any type that DFSORT allows (permanent, temporary, disk, tape, print, etc).</p> <p>In this case, DAPUBS is the ddname chosen for the temporary disk data set and PRPUBS is the ddname chosen for the print data set. You can use any valid 1-8 character ddnames you like.</p> <p>ICETOOL will automatically use OUTFIL to create both output data sets from a single pass over the input data set.</p>
5	<p>Leave at least one blank and type USING(SPUB)</p> <p>USING specifies the first four characters of the ddname for the data set containing the DFSORT control statements. In this case, the four characters chosen are SPUB, but you can use any four characters you like as long as they are valid for a ddname. The last four characters of the ddname are always CNTL, so in this case the full ddname is SPUBCNTL.</p> <p>For the SORT operator, you must specify a SORT control statement in the DFSORT control statement data set (SPUBCNTL) in order to tell DFSORT how to sort the input data set. You can also specify additional DFSORT control statements, like INCLUDE, OMIT, INREC, OUTREC and OUTFIL, as appropriate.</p>

To write the JCL statements that go with the SORT operator:

Table 64. Steps to Create JCL Statements for the SORT Operator

Step	Action
1	<p>Write DD statements for the input data sets and place them at the end of the job. In order to "concatenate" the two input data sets together, you must leave the ddname field blank in the second DD statement:</p> <pre>//BKS DD DSN=A123456.SORT.SAMPIN,DISP=SHR // DD DSN=A123456.SORT.SAMPADD,DISP=SHR</pre>
2	<p>Write DD statements for the disk and print output data sets and place them at the end of the job:</p> <pre>//DAPUBS DD DSN=&&DSRT,DISP=(,PASS),SPACE=(CYL,(2,2)),UNIT=SYSDA //PRPUBS DD SYSOUT=A</pre>
3	<p>Write a DD statement for the DFSORT control statement data set and place it at the end of the job:</p> <pre>//SPUBCNTL DD *</pre>
4	<p>Write the SORT control statement to sort the input data sets by publisher and title, and the INCLUDE statement to select only the books by publishers VALD and WETH, and place them after the SPUBCNTL statement:</p> <pre>SORT FIELDS=(106,4,A,1,75,A),FORMAT=CH INCLUDE COND=(106,4,EQ,C'VALD',OR,106,4,EQ,C'WETH'), FORMAT=CH</pre>

When complete, the TOOLIN statements and DD statements for this SORT operator look like this:

```

* Books from VALD and WETH
SORT FROM(BKS) TO(DAPUBS,PRPUBS) USING(SPUB)
/*
//BKS      DD DSN=A123456.SORT.SAMPIN,DISP=SHR
//          DD DSN=A123456.SORT.SAMPADD,DISP=SHR
//DAPUBS   DD DSN=&&DSRT,DISP=(,PASS),SPACE=(CYL,(2,2)),UNIT=SYSDA
//PRPUBS   DD SYSOUT=A
//SPUBCNTL DD *
SORT FIELDS=(106,4,A,1,75,A),FORMAT=CH
INCLUDE COND=(106,4,EQ,C'VALD',OR,106,4,EQ,C'WETH'),
          FORMAT=CH
/*

```

Tip: Instead of writing the previous INCLUDE statement using two conditions joined by OR, you could write it with one condition using INCLUDE's substring search feature, as you learned in “[Substring search for INCLUDE and OMIT](#)” on page 30, like this:

```
INCLUDE COND=(106,4,SS,EQ,C'VALD,WETH')
```

Table 65 on page 130 shows the Book Title and Publisher fields for the records as they would appear in the resulting output data sets. The actual records contain all of the fields.

Table 65. Books from publishers VALD and WETH

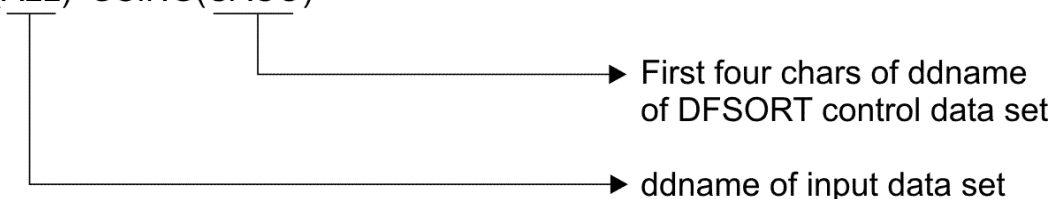
Book Title		Publisher
1	75	106 109
CELLS AND HOW THEY WORK		VALD
COMPLETE SPANISH DICTIONARY		VALD
EDITING SOFTWARE MANUALS		VALD
FREUD'S THEORIES		VALD
INTRODUCTION TO BIOLOGY		VALD
NOVEL IDEAS		VALD
SHORT STORIES AND TALL TALES		VALD
STRATEGIC MARKETING		VALD
VIDEO GAME DESIGN		VALD
ZEN BUSINESS		VALD
ANTICIPATING THE MARKET		WETH
CIVILIZATION SINCE ROME FELL		WETH
COMPUTERS: AN INTRODUCTION		WETH
EIGHTEENTH CENTURY EUROPE		WETH
GUIDE TO COLLEGE LIFE		WETH
GUNTHER'S GERMAN DICTIONARY		WETH
REBIRTH FROM ITALY		WETH
SYSTEM PROGRAMMING		WETH
THE INDUSTRIAL REVOLUTION		WETH

Creating different subsets of a sorted data set

If you want to create subsets of records from the same input data set, you can use OUTFIL statements with a SORT or COPY operator. The OUTFIL statements specify the ddnames of the output data sets, so the TO operand is not needed. All of the features of OUTFIL are available through the SORT and COPY operators.

A SORT operator that creates separate disk and tape data sets for the branch offices in California, and those in Colorado, sorted by city looks like this:

SORT FROM(ALL) USING(CACO)



Here are the steps for writing this SORT operator:

Table 66. Steps to Create the SORT Operator

Step	Action
1	Write a comment statement (optional): * Separate output for California and Colorado branches
2	Type SORT after the comment statement
3	Leave at least one blank and type FROM(ALL) ALL specifies the ddname for the input data set you want to sort. You can use the same ddname that you used for A123456.SORT.BRANCH in the STATS operator.
4	Leave at least one blank and type USING(CACO) The CACOCNTL data set contains the SORT and OUTFIL statements.

To write the JCL statements that go with the SORT operator follow these steps:

Table 67. Steps to Create JCL Statements for the SORT Operator

Step	Action
1	Write a DD statement for the DFSORT control statement data set and place it at the end of the job: //CACOCNTL DD *
2	Write the SORT control statement to sort the input data set by city, the OUTFIL statement to select only the California branches, and the OUTFIL statement to select only the Colorado branches, and place them after the CACOCNTL statement: SORT FIELDS=(1,15,CH,A) OUTFIL FNames=(CADASD,CATAPE),INCLUDE=(16,2,CH,EQ,C'CA') OUTFIL FNames=(CODASD,COTAPE),INCLUDE=(16,2,CH,EQ,C'CO')
3	Write DD statements for the disk and tape output data sets and place them at the end of the job: //CADASD DD DSN=&&CA,DISP=(,PASS),SPACE=(CYL,(2,2)),UNIT=3390 //CATAPE DD DSN=CA.BRANCH,UNIT=3480,VOL=SER=111111, // DISP=(NEW,KEEP),LABEL=(,SL) //CODASD DD DSN=&&CO,DISP=(,PASS),SPACE=(CYL,(2,2)),UNIT=3390 //COTAPE DD DSN=CO.BRANCH,UNIT=3480,VOL=SER=222222, // DISP=(NEW,KEEP),LABEL=(,SL)

When complete, the TOOLIN statements and DD statements for this SORT operator look like this:

```

* Separate output for California and Colorado branches
SORT FROM(ALL) USING(CACO)
/*
//CACOCNTL DD *
  SORT FIELDS=(1,15,CH,A)
  OUTFIL FNames=(CADASD,CATAPE),INCLUDE=(16,2,CH,EQ,C'CA')
  OUTFIL FNames=(CODASD,COTAPE),INCLUDE=(16,2,CH,EQ,C'CO')
/*
//CADASD DD DSN=&&CA,DISP=(,PASS),SPACE=(CYL,(2,2)),UNIT=3390
//CATAPE DD DSN=CA.BRANCH,UNIT=3480,VOL=SER=111111,
// DISP=(NEW,KEEP),LABEL=(,SL)
//CODASD DD DSN=&&CO,DISP=(,PASS),SPACE=(CYL,(2,2)),UNIT=3390
//COTAPE DD DSN=CO.BRANCH,UNIT=3480,VOL=SER=222222,
// DISP=(NEW,KEEP),LABEL=(,SL)

```

Table 68 on page 132 shows the records as they would appear in the CADASD data set (&&CA) and the CATAPE data set (CA.BRANCH) as a result of using the first OUTFIL statement.

Table 68. Records for California Sorted by City

City	State	Employees	Revenue	Profit
1 15	16 17	18 21	22 27	28 33
Los Angeles	CA	32	22530	-4278
Morgan Hill	CA	15	18200	3271
Sacramento	CA	29	42726	8276
San Diego	CA	22	32940	8275
San Francisco	CA	35	42820	6832
San Jose	CA	21	27225	8264
Sunnyvale	CA	18	16152	-978

Table 69 on page 132 shows the records as they would appear in the CODASD data set (&&CO) and the COTAPE data set (CO.BRANCH) as a result of using the second OUTFIL statement.

Table 69. Records for Colorado Sorted by City

City	State	Employees	Revenue	Profit
1 15	16 17	18 21	22 27	28 33
Aspen	CO	20	25800	5200
Boulder	CO	32	33866	7351
Denver	CO	33	31876	6288
Fort Collins	CO	22	12300	-2863
Vail	CO	19	23202	5027

Figure 4 on page 160 shows the complete TOOLMSG output.

So far

So far in this tutorial, you have learned how to print statistics for numeric fields using ICETOOL's STATS operator, and how to sort an input data set and create multiple output data sets using ICETOOL's SORT operator. Next, you will learn about ICETOOL's COPY operator.

Creating multiple unsorted data sets

If you want to create unsorted copies of an input data set, you can use ICETOOL's COPY operator. The COPY operator does not require any DFSORT statements. However, you can supply DFSORT statements (for example, INCLUDE, OMIT, INREC, OUTREC, or OUTFIL) if appropriate.

Here are a couple of examples of COPY operator statements with their accompanying JCL statements:

```
//TOOLIN DD *  
  COPY FROM(ALL) TO(D1,D2,D3)  
  COPY FROM(ALL) TO(P1) USING(COPY)  
/*  
//ALL DD DSN=A123456.SORT.BRANCH,DISP=SHR  
//D1 DD DSN=A123456.SORT.COPY1,DISP=OLD  
//D2 DD DSN=A123456.SORT.COPY2,DISP=OLD  
//D3 DD DSN=A123456.SORT.COPY3,DISP=OLD  
//P1 DD SYSOUT=*  
//COPYCNTL DD *  
  INCLUDE COND=(16,2,CH,EQ,C'CA')  
/*
```

The first COPY operator creates identical copies of A123456.SORT.BRANCH in A123456.SORT.COPY1, A123456.SORT.COPY2, and A123456.SORT.COPY3.

The second COPY operator prints the A123456.SORT.BRANCH records for the branches in California. Note that only the character fields in the resulting printed output will be readable. You will learn how to display numeric fields in readable format later in this chapter.

Because copying is more efficient than sorting, you should use the COPY operator rather than the SORT operator when possible.

So far

So far in this chapter you have learned about STATS, SORT, and COPY, three important ICETOOL operators. The next tutorial shows you how to use ICETOOL's RANGE operator.

Counting values in a range

You can use ICETOOL's RANGE operator to count the number of values for a particular numeric field that fall within a range you define. The range can be defined with:

Operand

Comparison

EQUAL (u)

Equal to u

NOTEQUAL (v)

Not equal to v

LOWER (w)

Less than w

HIGHER (x)

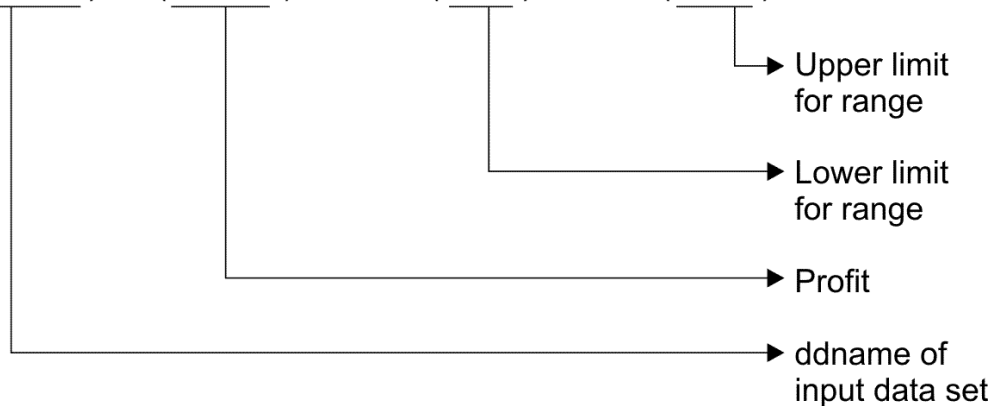
Greater than x

HIGHER (x) and LOWER (w)

Greater than x, but less than w

To print a count of the number of California branches with profit greater than -1500, but less than +8000, write the following RANGE statement:

RANGE FROM(CADASD) ON(28,6,PD) HIGHER(-1500) LOWER(+8000)



The input data set defined by CADASD is the same data set you created earlier (with the SORT operator) for the California branches. HIGHER(-1500) indicates that you want to count values in the profit field that are greater than -1500, while LOWER(+8000) indicates that you want to count values in the profit field that are less than +8000. For a negative limit, you must specify a minus (-) sign before the number. For a positive limit, you either specify a plus (+) sign before the number or leave it out, therefore, HIGHER(8000) is the same as HIGHER(+8000).

To print a count of the number of branches (in California and Colorado) with less than 32 employees, write the following RANGE statement:

RANGE FROM(ALL) ON(18,4,ZD) LOWER(32)



Because CADASD and ALL were previously defined, you don't need to add any new JCL statements to the ICETOOL job.

When these RANGE operators are run, the results are placed in the TOOLMSG data set. The TOOLMSG output produced for these RANGE operators would look like this:

```

      * California branches profit analysis
      RANGE FROM(CADASD) ON(28,6,PD) HIGHER(-1500) LOWER(+8000)
ICE627I 0 DFSORT CALL 0004 FOR COPY FROM CADASD TO E35 EXIT COMPLETED
ICE628I 0 RECORD COUNT: 0000000000000007
ICE631I 0 NUMBER OF VALUES IN RANGE FOR (28,6,PD) : 0000000000000003
ICE602I 0 OPERATION RETURN CODE: 00

      * Branches with less than 32 employees
      RANGE FROM(ALL) ON(18,4,ZD) LOWER(32)
ICE627I 0 DFSORT CALL 0005 FOR COPY FROM ALL TO E35 EXIT COMPLETED
ICE628I 0 RECORD COUNT: 0000000000000012
ICE631I 0 NUMBER OF VALUES IN RANGE FOR (18,4,ZD) : 0000000000000008
ICE602I 0 OPERATION RETURN CODE: 00

```

Looking at the output, you will notice that:

- Message ICE628I gives the count of records processed.
- Message ICE631I gives the count of values in the specified range. There were 3 California branches with profit greater than -1500, but less than +8000, and 8 branches in all with less than 32 employees.
- A return code for each operator is given in message ICE602I.

So far

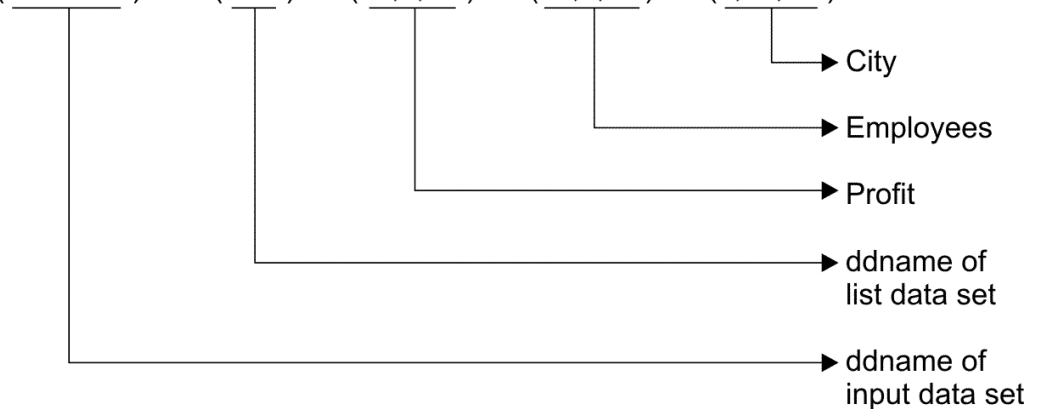
You have now learned how to count the number of values in a range for a particular field using ICETOOL's RANGE operator. Next, you will learn about ICETOOL's DISPLAY operator.

Printing simple reports

Numeric fields are often in a format (binary, fixed-point, or decimal) that is not readable when printed. You can use ICETOOL's DISPLAY operator to print simple reports showing up to 50 numeric and character fields from an input data set in readable form. The specified fields are printed in a list data set that you define. For numeric fields, appropriate plus (+) and minus (-) signs are printed along with the decimal value of each number.

To print a list data set showing the profit, employees, and city fields for the Colorado branches, write the following DISPLAY operator:

```
DISPLAY FROM(CODASD) LIST(OUT) ON(28,6,PD) ON(18,4,ZD) ON(1,15,CH)
```



The input data set defined by CODASD is the same data set you created earlier (with the SORT operator) for the Colorado branches. LIST specifies the ddname for the list data set you want the fields to be printed in. In this case, OUT is the ddname chosen for the list data set, but you can use any valid 1-8 character ddname you like. Specify the ON fields in the same order in which you want their values to be printed in the list data set.

Because OUT has not been defined previously, you must add a JCL statement for it to the end of the job:

```
//OUT DD SYSOUT=A
```

When this DISPLAY operator is run, the OUT data set looks like this:

(28,6,PD)	(18,4,ZD)	(1,15,CH)
+0000000000005200	+0000000000000020	Aspen
+0000000000007351	+0000000000000032	Boulder
+0000000000006288	+0000000000000033	Denver
-0000000000002863	+0000000000000022	Fort Collins
+0000000000005027	+0000000000000019	Vail

Note: The output actually contains ANSI carriage control characters in the first byte of the report, as explained previously in “Creating reports with OUTFIL” on page 88. However, ICETOOL always inserts actual blank lines in your report rather than using the '0' and '-' ANSI carriage control characters to print blank lines. So you do not have to do anything to force blank lines to be displayed when you view an ICETOOL report, as you do for an OUTFIL report. The ANSI carriage control characters are not shown in this section, although they are present in the reports.

If you don't want ANSI carriage control characters in your output records, you can use DISPLAY's NOCC operand to suppress them.

The values for profit, employees, and city are printed in separate columns across the page with a header for each column at the top. If more than one page is printed, DISPLAY puts the header at the top of each page. If you do not want the header printed, you can use DISPLAY's NOHEADER operand to suppress it.

DISPLAY also has two special ON fields you can use:

- **ON(VLEN)** can be used for variable length record data sets to print the length of each record.
- **ON(NUM)** can be used to print a relative record number for each record (starting with 1).

Use the ON(VLEN) or ON(NUM) field just as you would any other ON field.

Printing tailored reports

The previous tutorial showed you how to print a simple listing of numeric and character fields using the DISPLAY operator. By using additional operands of DISPLAY, you can create list data sets showing character and numeric fields in a variety of tailored report formats. You can specify:

- title strings in up to three title lines (TITLE operands), other title elements for the first title line (DATE, PAGE, and TIME operands) spacing between title elements (TBETWEEN operand), left alignment of title strings instead of center alignment (TLEFT operand) and title lines only on the first page instead of on every page (TFIRST operand)
- one, two or three line field headings (HEADER operand)
- spacing for columns (INDENT, BETWEEN, and STATLEFT operands)
- field formatting (BLANK and PLUS operands), and formatting items for individual ON fields
- statistics (TOTAL, AVERAGE, MAXIMUM, MINIMUM, and COUNT operands)
- lines per page (LINES operand)
- suppression of ANSI carriage control characters (NOCC operand)

Refer to *z/OS DFSORT Application Programming Guide* for complete details on these and other operands you can use with DISPLAY.

To print a report for the Colorado branches showing the city, profit and employee fields with a title line, field headings, totals, averages, minimums, and counts, write the following DISPLAY operator:

Total	21003	126
Average	4200	25
Lowest	-2863	19
Number of cities	5	

The title line and heading line are printed at the top of each page. The character data is left-justified and the numeric data is right-justified with zeros suppressed. The statistics are printed after the columns of data.

Using formatting items

The previous tutorial used the BLANK operand to change the way all numeric values in the report are displayed. You can use formatting items to change the appearance of individual numeric fields and their related statistics in the report, with respect to separators, number of digits, decimal point, decimal places, signs, leading zeros, division, leading strings, floating strings, and trailing strings. Formatting items can also be used to insert leading strings or trailing strings for character fields.

Formatting items are written as part of the ON operand, separated by commas, as follows: ON(p,m,f,formatting), ON(VLEN,formatting) and ON(NUM,formatting). EDCOUNT(formatting) can be used to specify formatting items for the count.

Edit masks

You can select from thirty-three pre-defined edit masks. The following table 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 70. Edit Mask Patterns			
Mask	Pattern	12345678	-1234567
A0	wdddddddddddddddddddddddddddddddd	12345678	-1234567
A1	wd,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	12.345.678	-1.234.567
A3	wd 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	12'345'678	-1'234'567
A5	d 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,d	1,234,567.8	-123,456.7
B2	wddd.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,d	1 234 567,8	-123 456,7
B4	wddd'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,d	1'234'567,8	-123'456,7
B6	ddd ddd ddd ddd ddd ddd ddd ddd ddd,dx	1 234 567,8	123 456,7-

Table 70. Edit Mask Patterns (continued)			
Mask	Pattern	12345678	-1234567
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,dd	123 456,78	-12 345,67
C4	wdd'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,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	12 345,678	-1 234,567
D4	wd'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	12'345,678	-1'234,567
D6	d 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,dddz	12,345,678	(1,234,567)
E2	yd.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,dddz	12 345 678	(1 234 567)
E4	yd'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,ddz	123,456.78	(12,345.67)
F2	ydd.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,ddz	123 456,78	(12 345,67)
F4	ydd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd,ddz	123'456.78	(12'345.67)
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,ddd	1,234.5678	-123.4567
G2	wddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd.ddd,ddd	1.234,5678	-123,4567
G3	wddd ddd ddd ddd ddd ddd ddd ddd ddd,ddd	1 234,5678	-123,4567
G4	wddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd,ddd	1'234.5678	-123.4567
G5	wddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd'ddd,ddd	1'234,5678	-123,4567
G6	ddd ddd ddd ddd ddd ddd ddd ddd ddd,dddx	1 234,5678	123,4567-

To use the E1 edit mask for the Profit field in the previous report, just change ON(28,6,PD) to ON(28,6,PD,E1). The RPT data set looks like this:

07/22/08	Colorado Branches Report	- 1 -
City	Profit	Employees
-----	-----	-----
Aspen	5,200	20
Boulder	7,351	32
Denver	6,288	33
Fort Collins	(2,863)	22
Vail	5,027	19
Total	21,003	126

Average	4,200	25
Lowest	(2,863)	19
Number of cities	5	

Number of digits

The default number of digits for a numeric field varies according to the type of field or statistic. You can change the number of digits for a numeric field by adding Udd to your formatting items where dd is the number of digits you want. To use 5 digits for the count value in the report, instead of the default of 15 digits, just add EDCOUNT(U05). The RPT data set looks like this:

07/22/08	Colorado Branches Report	- 1 -
City	Profit	Employees
-----	-----	-----
Aspen	5,200	20
Boulder	7,351	32
Denver	6,288	33
Fort Collins	(2,863)	22
Vail	5,027	19
Total	21,003	126
Average	4,200	25
Lowest	(2,863)	19
Number of cities	5	

Leading zeros

By default, leading zeros are not displayed when you use an edit mask, but you can change that by adding LZ to your formatting items as in the second ON field, as follows:

```
HEADER('No leading zeros','(without LZ)') ON(28,6,PD,E1) -
HEADER('Leading zeros','(with LZ)') ON(28,6,PD,E1,LZ)
```

The resulting report shows the difference in the way the Profit values from SORT.BRANCH are displayed with the E1 edit mask and without and with LZ. Note the use of HEADER('string1','string2') to produce a two line heading.

No leading zeros (without LZ)	Leading zeros (with LZ)
-----	-----
(4,278)	(00,000,004,278)
6,832	00,000,006,832
(2,863)	(00,000,002,863)
8,276	00,000,008,276
(978)	(00,000,000,978)
6,288	00,000,006,288
7,351	00,000,007,351
3,271	00,000,003,271
5,027	00,000,005,027
8,264	00,000,008,264
8,275	00,000,008,275
5,200	00,000,005,200

Edit patterns

The edit masks are not particularly useful for unsigned numeric data such as telephone numbers, dates, time-of-day, identification numbers, and so on. Instead, you can use edit patterns to change the way these types of numeric values are displayed in your report. E'pattern' is used to specify an edit pattern. You use a 9 in the pattern where you want a digit (0-9) from the numeric value to appear, and any other characters anywhere you want them to appear.

If you have an 8-byte ZD date in the form *mmddyyyy* in positions 21-28, you can display it as *mm/dd/yyyy* using ON(21,8,ZD,E'99/99/9999'). An 8-byte value of 03122004 is displayed as 03/12/2004.

If you have a 10-byte ZD telephone number in the form *aaapppnnnn* in positions 31-40, you can display it as (aaa)-ppp-nnnn using ON(31,10,ZD, E' (999) -999-9999'). A 10-byte value of 0123456789 is displayed as (012)-345-6789.

No statistics

By default, any statistics you request using TOTAL, MAXIMUM, MINIMUM, and AVERAGE (as well as BTOTAL, BMINIMUM, BMAXIMUM and BAVERAGE, which you will learn about later) are displayed for every numeric ON field. You can use the NOST formatting item to suppress statistics for numeric fields, such as telephone numbers, for which they are meaningless. The following DISPLAY operator prints a report from the SORT.BRANCH data set with totals for the Revenue and Profit fields, but not for the Employees field.

```
DISPLAY FROM(IN) LIST(RPT3) -
  HEADER('City') ON(1,15,CH) -
  HEADER('Employees') ON(18,4,ZD,NOST) -
  HEADER('Revenue') ON(22,6,PD) -
  HEADER('Profit') ON(28,6,PD) -
  TOTAL('Totals')
```

The RPT3 data set looks like this:

City	Employees	Revenue	Profit
Los Angeles	32	22530	-4278
San Francisco	35	42820	6832
Fort Collins	22	12300	-2863
Sacramento	29	42726	8276
Sunnyvale	18	16152	-978
Denver	33	31876	6288
Boulder	32	33866	7351
Morgan Hill	15	18200	3271
Vail	19	23202	5027
San Jose	21	27225	8264
San Diego	22	32940	8275
Aspen	20	25800	5200
Totals		329637	50665

Division

You can select from ten division items as follows:

- **/D** - divide by 10
- **/C** - divide by 100
- **/K** - divide by 1000
- **/DK** - divide by 10000 (10*1000)
- **/CK** - divide by 100000 (100*1000)
- **/M** - divide by 1000000 (1000*1000)
- **/G** - divide by 1000000000 (1000*1000*1000)
- **/KB** - divide by 1024
- **/MB** - divide by 1048576 (1024*1024)
- **/GB** - divide by 1073741824 (1024*1024*1024)

The Profit values from SORT.BRANCH look as follows with HEADER('Profit/(Loss) in K\$') and ON(28,6,PD,E1,/K):

```
Profit/(Loss) in K$
-----
              (4)
              6
```

```
(2)
8
0
6
7
3
5
8
8
5
```

Leading, floating and trailing characters

You can add floating characters to your numeric fields and add leading and trailing characters to your numeric and character fields as follows:

- **F'string'** - a floating string to appear to the left of the first non-blank character of the formatted numeric data.
- **L'string'** - a leading string to appear at the beginning of the character or numeric data column.
- **T'string'** - a trailing string to appear at the end of the character or numeric data column.

The Profit values from SORT.BRANCH look as follows with HEADER('Profit') and ON(28,6,PD,A1,F'\$,T'**):

```
----- Profit
$-4,278**
$6,832**
$-2,863**
$8,276**
$-978**
$6,288**
$7,351**
$3,271**
$5,027**
$8,264**
$8,275**
$5,200**
```

Printing sectioned reports

The previous tutorial showed you how to print tailored reports using the DISPLAY operator. By using the BREAK operand of DISPLAY, you can create reports divided into sections by a character or numeric break field on which you have previously sorted. You can use formatting items with BREAK(p,m,f,formatting) in the same way you can use them with ON(p,m,f,formatting). You can also specify a string for the break title (BTITLE operand) and statistics for the individual sections (BTOTAL, BAVERAGE, BMAXIMUM, BMINIMUM and BCOUNT operands). EDBCOUNT(formatting) can be used to specify formatting items for BCOUNT.

For this example, we will use the data set with books from publishers VALD and WETH, sorted by publisher and title, that we created previously. To print a report with sections by publisher showing the title and price fields with a title line, field headings, break title, break

averages and totals, and overall averages and totals, write the following DISPLAY operator:

*** Print a report of books for individual publishers**

DISPLAY FROM(DAPUBS) LIST(SECTIONS) -

→ ddnames of data sets

TITLE('BOOKS FOR INDIVIDUAL PUBLISHERS') PAGE -

→ Title line elements

HEADER('TITLE OF BOOK') ON(1,35,CH) -

→ Heading and field

HEADER('PRICE OF BOOK') ON(170,4,BI,C1,F'\$') -

→ Heading and field

BTITLE('PUBLISHER:') BREAK(106,4,CH) -

→ Break field

→ Break title

BAVERAGE('AVERAGE FOR THIS PUBLISHER') -

→ Section average

BTOTAL('TOTAL FOR THIS PUBLISHER') -

→ Section total

AVERAGE('AVERAGE FOR ALL PUBLISHERS') -

→ Overall average

TOTAL('TOTAL FOR ALL PUBLISHERS') -

→ Overall total

DAPUBS is the ddname for the previously created VALD and WETH data set. SECTIONS is the ddname for the list data set in which you want the report to be printed.

TITLE and PAGE indicate the elements to be included in the title line and their placement.

Each HEADER and ON pair indicate a field to be included in the report and the heading to be used for it.

BTITLE indicates a string to be used for the break title and its placement (before or after the break field).

BREAK indicates the break field to be used to create sections. BVERAGE and BTOTAL indicate section statistics to be produced at the end of each section.

AVERAGE and TOTAL indicate overall statistics to be produced at the end of the report.

Because SECTIONS has not been defined previously, you must add a JCL statement for it at the end of the job:

```
//SECTIONS DD SYSOUT=A
```

When this DISPLAY operator is run, it produces a three-page report for the SECTIONS data set that looks like this:

```
BOOKS FOR INDIVIDUAL PUBLISHERS      - 1 -
PUBLISHER:  VALD

TITLE OF BOOK                        PRICE OF BOOK
-----
CELLS AND HOW THEY WORK              $24.95
COMPLETE SPANISH DICTIONARY          $6.50
EDITING SOFTWARE MANUALS            $14.50
FREUD'S THEORIES                    $12.50
INTRODUCTION TO BIOLOGY              $23.50
NOVEL IDEAS                          $24.50
SHORT STORIES AND TALL TALES         $15.20
STRATEGIC MARKETING                 $23.50
VIDEO GAME DESIGN                   $21.99
ZEN BUSINESS                         $12.00

AVERAGE FOR THIS PUBLISHER           $17.91
TOTAL FOR THIS PUBLISHER              $179.14
```

```
BOOKS FOR INDIVIDUAL PUBLISHERS      - 2 -
PUBLISHER:  WETH

TITLE OF BOOK                        PRICE OF BOOK
-----
ANTICIPATING THE MARKET             $20.00
CIVILIZATION SINCE ROME FELL         $13.50
COMPUTERS: AN INTRODUCTION           $18.99
EIGHTEENTH CENTURY EUROPE            $17.90
GUIDE TO COLLEGE LIFE                $20.00
GUNTHER'S GERMAN DICTIONARY          $10.88
REBIRTH FROM ITALY                   $25.60
SYSTEM PROGRAMMING                  $31.95
THE INDUSTRIAL REVOLUTION             $7.95

AVERAGE FOR THIS PUBLISHER           $18.53
TOTAL FOR THIS PUBLISHER              $166.77
```

```
BOOKS FOR INDIVIDUAL PUBLISHERS      - 3 -

TITLE OF BOOK                        PRICE OF BOOK
-----

AVERAGE FOR ALL PUBLISHERS           $18.20
TOTAL FOR ALL PUBLISHERS              $345.91
```

So far

You have now learned how to print simple, tailored and sectioned reports using ICETOOL's DISPLAY operator. Next, you will learn about ICETOOL's OCCUR operator.

Printing how many times fields occur

You can use ICETOOL's OCCUR operator to print a simple or tailored report showing how many times different ON field values occur, sorted by those ON field values. Values that occur only once are called "non-duplicate" values. Values that occur more than once are called "duplicate" values.

You can list all of the values in the data set, or list different combinations of duplicate and non-duplicate values, using the following operands:

- **ALLDUPS** - only list duplicate values
- **NODUPS** - only list non-duplicate values
- **EQUAL(n)** - only list values that occur n times
- **HIGHER(n)** - only list values that occur more than n times
- **LOWER(n)** - only list values that occur less than n times

Setting up an OCCUR report is similar to setting up a DISPLAY report. Specify ON fields for OCCUR in the same order you want their values to be printed in the list data set. OCCUR also has two special ON fields you can use:

- **ON(VALCNT)** can be used to print the number of times each field value occurs.
- **ON(VLEN)** can be used for the record length of variable length records.

You can use up to 10 ON fields; all of the ON field values, except ON(VALCNT) and ON(VLEN), are used for sorting and counting occurrences. For example, if you use ON(1,4,CH), 'ABCD+01' and 'ABCD-01' are sorted in that order (by 1,4,CH) and counted as two occurrences of 'ABCD'. However, if you use ON(1,4,CH) and ON(5,3,FS), 'ABCD-01' is sorted before 'ABCD+01' (by 1,4,CH,A and 5,3,FS,A) and counted as one occurrence of 'ABCD-01' and one occurrence of 'ABCD+01'.

You can use the following for an OCCUR report in the same way you use them for a DISPLAY report:

- title strings in up to three title lines (TITLE operands), other title elements for the first title line (DATE, PAGE, and TIME operands), spacing between title elements (TBETWEEN operand), left alignment of title strings instead of center alignment (TLEFT operand) and title lines only on the first page instead of on every page (TFIRST operand).
- field headings (HEADER operand)
- spacing for columns (INDENT and BETWEEN operands)
- field formatting (BLANK and PLUS operands), and formatting items for individual ON fields as follows: ON(p,m,f,formatting), ON(VALCNT,formatting) and ON(VLEN,formatting)
- lines per page (LINES operand)
- Suppression of ANSI carriage control characters (NOCC operand)

Refer to [*z/OS DFSORT Application Programming Guide*](#) for complete details about these and other operands you can use with OCCUR.

For this example, we will use the sample bookstore data set as input. To print a report showing the number of different books in use from each publisher, write the following OCCUR statement:

* Print the count of books in use from each publisher

OCCUR FROM(BKIN) LIST(PUBCT) BLANK -

→ Alternate print format

→ ddnames of data sets

TITLE('Books from Publishers') DATE(DMY.) -

→ Title line elements

HEADER('Publisher') HEADER('Books Used') -

→ Field headings

ON(106,4,CH) ON(VALCNT,N05)

→ Publisher and Count

BKIN is the ddname for the sample bookstore data set. PUBCT is the ddname for the list data set in which you want the report to be printed. BLANK specifies the field format to be used.

TITLE indicates the title string to appear in the title line. DATE indicates the format in which the date is to appear in the title line (dd.mm.yy where dd is the two-digit day, mm is the two-digit month and yy is the last two digits of the year).

The HEADER strings correspond to the ON fields. ON(VALCNT,N05) is a special ON field used with OCCUR to print the count of occurrences. N05 is a formatting item used to set the number of digits for the count to 5, overriding the default of 15 digits for VALCNT.

Write DD statements for the A123456.SORT.SAMPIN and PUBCT data sets and place them at the end of the job:

```
//BKIN DD DSN=A123456.SORT.SAMPIN,DISP=SHR
//PUBCT DD SYSOUT=A
```

When this OCCUR operator is run, the PUBCT data set looks like this:

```
Books from Publishers      21.10.03

Publisher   Books Used
-----
COR         7
FERN        4
VALD        5
WETH        4
```

The name of each publisher is printed along with the number of times that publisher appeared in the sample bookstore data set, which is equivalent to the number of different books from that publisher.

So far

You have now learned how use ICETOOL's OCCUR operator to print a simple or tailored report showing how many times different field values occur. Next, you will learn how to use ICETOOL's SELECT operator.

Selecting records by field occurrences

You can use ICETOOL's SELECT operator to create an output data set with records selected according to how many times different ON field values occur, sorted by those ON field values. As with the OCCUR

operator, values that occur only once are called non-duplicate values, and values that occur more than once are called duplicate values.

You can use up to 10 ON fields; all of the ON field values are used for sorting and counting occurrences. For example, if you use ON(1,4,CH), 'ABCD+01' and 'ABCD-01' are sorted in that order (by 1,4,CH,A) and counted two occurrences of ('ABCD'). However, if you use ON(1,4,CH) and ON(5,3,FS), 'ABCD-01' is sorted before 'ABCD+01' (by 1,4,CH,A and 5,3,FS,A) and counted as one occurrence of 'ABCD-01' and one occurrence of 'ABCD+01'.

You can select different combinations of records with duplicate and non-duplicate values using the following operands:

- **FIRST** - keep only the first record for each value (that is, records with non-duplicate values, and the first record for duplicate values)
- **FIRST(n)** - keep only the first n records for each value (that is, records with non-duplicate values, and the first n records for duplicate values)
- **LAST** - keep only the last record for each value (that is, records with non-duplicate values, and the last record for duplicate values)
- **FIRSTDUP** - only keep the first record for duplicate values
- **FIRSTDUP(n)** - only keep the first n records for duplicate values
- **LASTDUP** - only keep the last record for duplicate values
- **ALLDUPS** - only keep records with duplicate values
- **NODUPS** - only keep records with non-duplicate values
- **EQUAL(n)** - only keep records with values that occur n times
- **HIGHER(n)** - only keep records with values that occur more than n times
- **LOWER(n)** - only keep records with values that occur less than n times

The selected records are written to the output data set identified by the TO operand. If appropriate, you can use the DISCARD operand with any of the other operands shown previously to save the records that are **not** selected in a separate output data set identified by the DISCARD operand. You can create just the TO data set, just the DISCARD data set, or both.

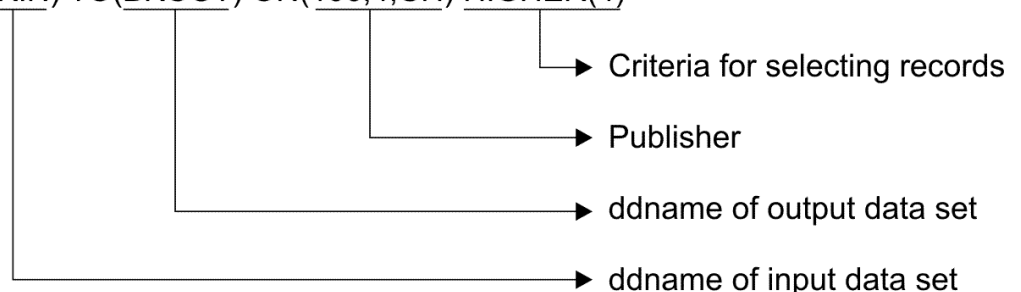
You can use a USING data set to specify DFSORT INCLUDE, OMIT, INREC, and OUTFIL statements for your SELECT operation. INCLUDE or OMIT and INREC statement processing is performed before SELECT processing. OUTFIL statement processing is performed after SELECT processing.

To create an output data set containing records for publishers with more than four different books in use, write the following SELECT statement:

* Separate output containing records for publishers

* with more than 4 books in use

```
SELECT FROM(BKIN) TO(BKOUT) ON(106,4,CH) HIGHER(4)
```



BKIN is the ddname for the sample bookstore data set. BKOUT is the ddname of the output data set that will contain the records for each publisher field value that occurs more than 4 times (all of the records for COR and VALD in this case).

Write a DD statement for the A123456.BOOKS1 data sets and place it at the end of the job:

```
//BKOUT DD DSN=A123456.BOOKS1,DISP=(NEW,CATLG,DELETE),
// SPACE=(CYL,(3,3)),UNIT=3390
```

Table 71 on page 148 shows the Book Title and Publisher fields for the records in the resulting output data set. The actual records contain all of the fields.

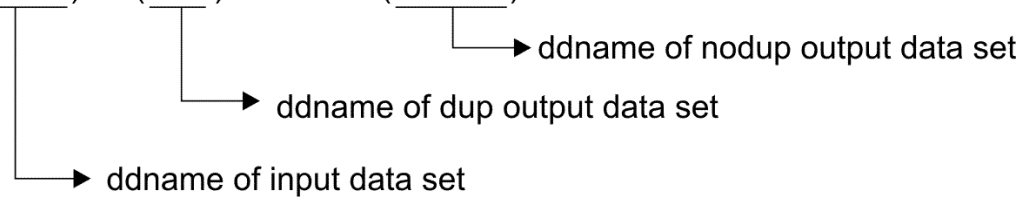
Table 71. Books from Publishers with More than Four Books in Use

Book Title		Publisher
1	75	106 109
LIVING WELL ON A SMALL BUDGET		COR
SUPPLYING THE DEMAND		COR
INKLINGS: AN ANTHOLOGY OF YOUNG POETS		COR
PICK'S POCKET DICTIONARY		COR
MODERN ANTHOLOGY OF WOMEN POETS		COR
INTRODUCTION TO PSYCHOLOGY		COR
CRISES OF THE MIDDLE AGES		COR
VIDEO GAME DESIGN		VALD
EDITING SOFTWARE MANUALS		VALD
STRATEGIC MARKETING		VALD
SHORT STORIES AND TALL TALES		VALD
INTRODUCTION TO BIOLOGY		VALD

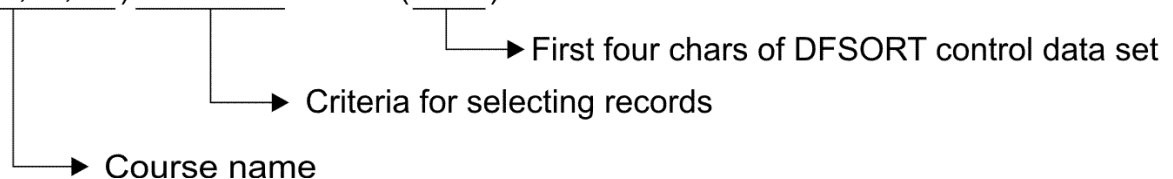
To create separate output data sets containing records with only the course name and author's last name, both for courses that use more than one book, and for courses that use only one book, write the following SELECT statement:

* Course name and author's last name for courses with more than one book
* and only one book

SELECT FROM(BKIN) TO(DUP) DISCARD(NODUP)



ON(120,25,CH) ALLDUPS USING(CTL1)



BKIN is the ddname for the sample bookstore data set. DUP is the ddname of the output data set to contain the records for courses with more than one book. NODUP is the ddname of the output data set to contain the records for courses with only one book.

Here is the complete JCL for the job, including control statements:

```
//SEL2 JOB A492,PROGRAMMER
//EXTRACT EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//BKIN DD DSN=A123456.SORT.SAMPIN,DISP=SHR
//TOOLIN DD *
* Course name and author's last name for courses with more than one book
* and only one book
SELECT FROM(BKIN) TO(DUP) DISCARD(NODUP) -
```



```

ON(120,25,CH) ALLDUPS USING(CTL1)
/*
//CTL1CNTL DD *
  OMIT COND=(120,25,CH,EQ,C' ')
  OUTFIL FAMES=DUP,OUTREC=(120,25,2X,145,15)
  OUTFIL FAMES=NODUP,OUTREC=(120,25,2X,145,15)
/*
//DUP DD SYSOUT=*
//NODUP DD SYSOUT=*

```

The OMIT statement removes records with a blank course name before SELECT processing. The OUTFIL statement for DUP reformats the selected records for courses with more than one book to have just the course name and author's last name. The OUTFIL statement for NODUP reformats the selected records for courses with only one book to have just the course name and author's last name.

Here are the DUP records exactly as they would appear:

INTRO TO COMPUTERS	CHATTERJEE
INTRO TO COMPUTERS	CHATTERJEE
INTRO TO COMPUTERS	CHATTERJEE
MODERN POETRY	FRIEDMAN
MODERN POETRY	FRIEDMAN
WORLD HISTORY	GOODGOLD
WORLD HISTORY	WILLERTON

Here are the NODUP records exactly as they would appear:

ADVANCED MARKETING	LORCH
BIOLOGY I	GREENBERG
DATA MANAGEMENT	SMITH
EUROPEAN HISTORY	BISCARDI
FICTION WRITING	BUCK
MARKETING	MAXWELL
PSYCHOANALYSIS	NAKATSU
PSYCHOLOGY I	ZABOSKI
TECHINCAL EDITING	MADRID
TECHNICAL EDITING	MADRID
VIDEO GAMES	NEUMANN

Note that "TECHINCAL EDITING" and "TECHNICAL EDITING" are both included in the NODUP data set because they are different ("TECHINCAL" is spelled incorrectly).

Suppose you want to use the SORT.BRANCH data set to display the three branches in each state with the highest number of employees. Write the following ICETOOL statement:

```

SELECT FROM(BRANCH) TO(HIGH3) ON(16,2,CH) FIRST(3) USING(HIGH)

```

Include the following DFSORT statements in HIGHCNTL:

```

SORT FIELDS=(16,2,CH,A,18,4,ZD,D)
OUTFIL FAMES=BRANCH,
  BUILD=(16,2,2X,18,4,ZD,EDIT=(IIT))

```

The SORT statement orders the records ascending by state and descending by number of employees. This brings the records for each state with the highest number of employees to the top. The intermediate result would be:

State	Employees
CA	35
CA	32
CA	29
CA	22
CA	21
CA	18
CA	15
CO	33
CO	32
CO	22
CO	20
CO	19

The FIRST(3) operand tells SELECT to keep the first 3 records for each ON field (the state). Finally, the OUTFIL statement extracts the state and employees values from the selected records.

The output records would look like this:

```
CA 35
CA 32
CA 29
CO 33
CO 32
CO 22
```

So far

So far in this chapter you have learned how to print statistics for numeric fields, create sorted and unsorted data sets, obtain a count of numeric fields in a range for a particular field, print fields from an input data set, print reports, print a count of field occurrences and select output records based on field occurrences. Next, you will learn how to use ICETOOL's SPLICE operator.

Joining fields from different data sets

You can use ICETOOL's SPLICE operator to create output records in a variety of ways by splicing together up to 50 fields from records that have the same ON field values, but different information. The output records are sorted by the ON field values. The records to be spliced can originate from different input data sets, making it possible to perform various "join" and "match" operations.

You can use up to 10 ON fields; all of the ON field values are used for sorting and splicing. For example, if you use ON(1,4,CH), 'ABCD+01' and 'ABCD-01' are sorted in that order (by 1,4,CH,A) and counted as two occurrences of 'ABCD'. However, if you use ON(1,4,CH) and ON(5,3,FS), 'ABCD-01' is sorted before 'ABCD+01' (by 1,4,CH,A and 5,3,FS,A) and counted as one occurrence of 'ABCD-01' and one occurrence of 'ABCD+01'.

To do a join or match operation on two input data sets, you need to have the fields aligned appropriately for each pair of records to be spliced. Typically, you accomplish that by copying and reformatting one or both input data sets to temporary data sets, so you can splice the temporary data sets together.

Suppose you have two input data sets, REGION.IN1 and REGION.IN2, as shown in [Table 72 on page 150](#) and [Table 73 on page 150](#).

Table 72. REGION.IN1 data set for join

Region	Headquarters	Regional Director
1 5	6 20	21 35
East	Philadelphia	C. Kent
West	San Jose	B. Wayne
North	Boston	P. Parker
South	Charlotte	D. Prince

Table 73. REGION.IN2 data set for join

Office	Region	Employees	Evaluation	Established
1 4	5 9	10 13	14 23	24 27

Table 73. REGION.IN2 data set for join (continued)

Office	Region	Employees	Evaluation	Established
0001	East	0050	Fair	1983
0001	South	0023	Good	1976
0002	South	0068	Fair	1978
0002	East	0125	Excellent	1986
0001	West	0052	Good	1995
0003	East	0028	Good	1994
0002	West	0105	Excellent	2001
0003	South	0054	Fair	1992
0001	North	0200	Fair	1991
0004	South	0070	Good	2002

From these two input data sets, you want to create an output data set, REGION.OUT. For each record in REGION.IN2, you want to look up the corresponding Region in REGION.IN1, and combine fields from the two records into one output record in REGION.OUT, as shown in Table 74 on page 151.

Table 74. REGION.OUT data set for join

Office	Region	Regional Director	Employees	Evaluation	Headquarters
1 4	5 9	10 24	25 28	29 38	39 53
0001	East	C. Kent	0050	Fair	Philadelphia
0002	East	C. Kent	0125	Excellent	Philadelphia
0003	East	C. Kent	0028	Good	Philadelphia
0001	North	P. Parker	0200	Fair	Boston
0001	South	D. Prince	0023	Good	Charlotte
0002	South	D. Prince	0068	Fair	Charlotte
0003	South	D. Prince	0054	Fair	Charlotte
0004	South	D. Prince	0070	Good	Charlotte
0001	West	B. Wayne	0052	Good	San Jose
0002	West	B. Wayne	0105	Excellent	San Jose

Write the following ICETOOL statements and JCL statements to create REGION.OUT from REGION.IN1 and REGION.IN2:

```
* Reformat REGION.IN1 to T1 so it can be spliced
COPY FROM(REGNIN1) TO(T1) USING(CTL1)
* Reformat REGION.IN2 to T1 so it can be spliced
COPY FROM(REGNIN2) TO(T1) USING(CTL2)
* Splice records in T1 with matching ON fields
SPICE FROM(T1) WITHALL -
  ON(5,5,CH) - Region
  WITH(1,4) - Office
  WITH(25,4) - Employees
  WITH(29,10) - Evaluation
  TO(REGNOUT)
/*
//REGNIN1 DD DSN=A123456.REGION.IN1,DISP=SHR
//REGNIN2 DD DSN=A123456.REGION.IN2,DISP=SHR
//T1 DD DSN=*&&T1,UNIT=3390,SPACE=(CYL,(5,5)),DISP=(MOD,PASS)
//REGNOUT DD DSN=A123456.REGION.OUT,DISP=(NEW,CATLG,DELETE),UNIT=3390,
// SPACE=(CYL,(5,5))
//CTL1CNTL DD *
* Move REGION.IN1 fields to their locations for the
* output data set
  OUTREC FIELDS=(5:1,5, Region
                  10:21,15, Regional Director
                  39:6,15) Headquarters
/*
//CTL2CNTL DD *
* Move REGION.IN2 fields to their locations for the
* output data set
  OUTREC FIELDS=(1:1,4, Office
                  5:5,5, Region
                  25:10,4, Employees
```

29:14,10, Evaluation
53:X)
/*

The first COPY operator writes records to temporary data set T1 with the fields from REGION.IN1 as shown in [Table 75 on page 152](#).

Table 75. T1 data set fields from REGION.IN1

Blanks	Region	Regional Director	Blanks	Blanks	Headquarters
1 4	5 9	10 24	25 28	29 38	39 53

The second COPY operator writes records at the end (MOD) of temporary data set T1 with the fields from REGION.IN2 as shown in [Table 76 on page 152](#).

Table 76. T1 data set fields from REGION.IN2

Office	Region	Blanks	Employees	Evaluation	Blanks
1 4	5 9	10 24	25 28	29 38	39 53

Because the Region field is the ON field you want SPLICE to use to match your records, you must move it to the same location in both types of reformatted records (positions 5-9). You must move all of the other fields that you want in your output records to the locations in which you want them to appear in the combined record. Put blanks in each record for the corresponding fields in the other type of record. For example, the Regional Director field appears in positions 10-24 of the reformatted REGION.IN1 records while blanks appear in the corresponding positions 10-24 of the reformatted REGION.IN2 records, and the Office field appears in positions 1-4 of the reformatted REGION.IN2 records, while blanks appear in the corresponding positions 1-4 of the reformatted REGION.IN1 records.

The FROM(T1) operand tells DFSORT that the input for the SPLICE operation is the T1 data set, which contains the reformatted REGION.IN1 and REGION.IN2 records, in that order. The order of the input records is important, because the first record for each matching ON field acts as the base record on to which the second or subsequent record is spliced.

In this case, all of the reformatted REGION.IN1 records come before all of the reformatted REGION.IN2 records, so the SPLICE operator splices the WITH fields from the reformatted REGION.IN2 records to the reformatted REGION.IN1 record whenever their ON fields match. WITHALL tells SPLICE to splice the first record with every other record for which the ON fields match. Without WITHALL, only the first and last records are spliced.

This is just one example of the many types of join operations you can do with SPLICE, using some of its available operands.

You can also use a USING data set to specify DFSORT INCLUDE, OMIT, INREC, and OUTFIL statements for your SPLICE operation. INCLUDE or OMIT and INREC statement processing is performed before SPLICE processing. OUTFIL statement processing is performed after SPLICE processing.

For complete details of all of the operands you can use with SPLICE, as well as many more examples of join operations, see [z/OS DFSORT Application Programming Guide](#).

Matching records from different data sets

SPLICE can be used for match operations as well as join operations. Suppose you want to look at the courses in SORT.SAMPIN and SORT.SAMPADD and produce three output data sets as follows:

- COURSE.MATCH to show the courses that appear in both SORT.SAMPIN and SORT.SAMPADD
- COURSE.INONLY to show the courses that appear only in SORT.SAMPIN
- COURSE.ADDONLY to show the courses that appear only in SORT.SAMPADD

The following JCL and ICETOOL statements create these three data sets:

```

//S1      EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG  DD SYSOUT=*
//F1IN     DD DSN=A123456.SORT.SAMPIN,DISP=SHR
//F2IN     DD DSN=A123456.SORT.SAMPADD,DISP=SHR
//T1       DD DSN=88&T1,DISP=(MOD,PASS),UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE)
//MATCH DD DSN=COURSE.MATCH,DISP=(NEW,CATLG,DELETE),
//        SPACE=(CYL,(5,5)),UNIT=3390
//F1ONLY DD DSN=COURSE.INONLY,DISP=(NEW,CATLG,DELETE),
//        SPACE=(CYL,(5,5)),UNIT=3390
//F2ONLY DD DSN=COURSE.ADDONLY,DISP=(NEW,CATLG,DELETE),
//        SPACE=(CYL,(5,5)),UNIT=3390
//TOOLIN   DD *
* Copy needed File1 fields to T1 and add '11' id
COPY FROM(F1IN) TO(T1) USING(MATA)
* Copy needed File2 fields to T1 and add '22' id
COPY FROM(F2IN) TO(T1) USING(MATB)
* Splice second id character from overlay to base for
* matching course numbers. Use resulting spliced or
* unspliced id to write records to MATCH, F1ONLY
* or F2ONLY as appropriate.
SPLICE FROM(T1) TO(MATCH) ON(1,5,CH) -
      WITH(32,1) KEEPNOUDUPS USING(MATC)
/*
//MATACTL DD *
* Extract needed File1 fields and add '11' id
OUTREC FIELDS=(1:115,5,      Course number
               6:120,25,     Course name
               31:C'11')     '11' id for File1 records
/*
//MATBCTL DD *
* Extract needed File2 fields and add '22' id
OUTREC FIELDS=(1:115,5,      Course number
               6:120,25,     Course name
               31:C'22')     '22' id for File2 records
/*
//MATCNTL DD *
* A '12' id indicates a match between File1 and File2.
* Remove the '12' id and write the records to MATCH.
OUTFIL FNAMES=MATCH,INCLUDE=(31,2,CH,EQ,C'12'),OUTREC=(1,30)
* A '11' id indicates a record in File1 only.
* Remove the '11' id and write the records to F1ONLY.
OUTFIL FNAMES=F1ONLY,INCLUDE=(31,2,CH,EQ,C'11'),OUTREC=(1,30)
* A '22' id indicates a record in File2 only.
* Remove the '22' id and write the records to F2ONLY.
OUTFIL FNAMES=F2ONLY,INCLUDE=(31,2,CH,EQ,C'22'),OUTREC=(1,30)
/*

```

The first COPY operator writes records to temporary data set T1 consisting of the course number and course name fields from SORT.SAMPIN, and an id of '11', as shown in [Table 77 on page 153](#).

Table 77. T1 data set fields from SORT.SAMPIN

Course Number	Course Name	'11'
1 5	6 30	31 32

The second COPY operator writes records at the end (MOD) of temporary data set T1 consisting of the course number and course name fields from SORT.SAMPADD, and an id of '22', as shown in [Table 78 on page 153](#).

Table 78. T1 data set fields from SORT.SAMPADD

Course Number	Course Name	'22'
1 5	6 30	31 32

The SPLICE operator matches the course numbers (ON field). When a match is found, the second id character is spliced into the base record. Because WITHALL is not specified, only the last record for each match is spliced with the first record for each match. KEEPNOUDUPS tells DFSORT to keep records that are not spliced. Without KEEPNOUDUPS, records for course numbers that appear once in one data set, but not in the other data set, would be deleted instead of being written to COURSE.INONLY or COURSE.ADDONLY.

Here's what happens for all of the possible types of matches and non-matches:

- For a course number that appears one or more times with id '11' and one or more times with id '22', the second '2' from the last '22' record is spliced into the first '11' record to get an id of '12'. Thus, an id of '12' represents a course number that appears in both SORT.SAMPIN and SORT.SAMPADD
- For a course number that appears only once with id '11' (and not with id '22') the id of '11' is not changed. For a course number that appears more than once with id '11' (and not with id '22') the second '1' from the last '11' record is spliced into the first '11' record to get an id of '11'. Thus, an id of '11' represents a course number that only appears in SORTIN.SAMPIN.
- For a course number that appears only once with an id of '22' (and not with an id of '11') the id of '22' is not changed. For a course number that appears more than once in with an id of '22' (and not with an id of '11') the second '2' from the last '22' record is spliced into the first '22' record to get an id of '22'. Thus, an id of '22' represents a course number that only appears in SORTIN.SAMPADD.

The output created for COURSE.MATCH containing the information for course numbers that appear in both SORT.SAMPIN and SORT.SAMPADD is shown in [Table 79 on page 154](#).

Table 79. COURSE.MATCH output

Course Number	Course Name
1 5	6 30
00103	DATA MANAGEMENT
00205	VIDEO GAMES
10054	FICTION WRITING
30016	PSYCHOLOGY I
30975	PSYCHOANALYSIS
50420	WORLD HISTORY
50521	WORLD HISTORY
50632	EUROPEAN HISTORY
70124	ADVANCED MARKETING
70251	MARKETING

The output created for COURSE.INONLY containing the information for course numbers that appear only in SORT.SAMPIN is shown in [Table 80 on page 154](#).

Table 80. COURSE.INONLY output

Course Number	Course Name
1 5	6 30
00032	INTRO TO COMPUTERS
10347	TECHNICAL EDITING
10856	MODERN POETRY
80521	BIOLOGY I

The output created for COURSE.ADDONLY containing the information for course numbers that appear only in SORT.SAMPADD is shown in [Table 81 on page 154](#).

Table 81. COURSE.ADDONLY output

Course Number	Course Name
1 5	6 30

Table 81. COURSE.ADDONLY output (continued)

Course Number	Course Name
70255	BUSINESS THEORY
80522	BIOLOGY II
80523	INTRO TO GENETICS

Sorting records between headers and trailers

You can use ICETOOL's DATASORT operator to sort the data records in a data set without sorting the header or trailer records. You use the following operands to tell DATASORT the number of header records, trailer records, or header and trailer records in your data set:

- **HEADER** or **FIRST** - the first record is a header record
- **HEADER(x)** or **FIRST(x)** - the first x records are header records
- **TRAILER** or **LAST** - the last record is a trailer record
- **TRAILER(y)** or **LAST(y)** - the last y records are trailer records

DATASORT does not require an "identifier" in the header or trailer records; it can treat the first x records as header records and the last y records as trailer records.

You must specify a USING data set to specify a DFSORT SORT statement that tells DATASORT the fields you want to use to sort your data records. You can use various other DFSORT statements as well; see [z/OS DFSORT Application Programming Guide](#) for details.

DATASORT sorts the data records between your header records (first x records of the data set) and trailer records (last y records of the data set) while keeping the header and trailer records in place.

Suppose you have input records that look like this:

```
2008/04/23
Geometry
Algebra
Trigonometry
Calculus
0004
End of data set
```

The first record is a header record containing a date, the last two records are trailer records containing a count and an end indicator, respectively, and the records between the header and trailer records are data records containing Subject names. You want to create an output data set with the Subject names sorted but the header and trailer records in their original places. Write the following ICETOOL statement:

```
DATASORT FROM(IN) TO(OUT) HEADER TRAILER(2) USING(CTL1)
```

Include the following DFSORT SORT statement in CTL1CNTL:

```
SORT FIELDS=(1,16,CH,A)
```

The output records would look like this:

```
2008/04/23
Algebra
Calculus
Geometry
Trigonometry
0004
End of data set
```

For complete details of DATASORT, as well as more examples, see [z/OS DFSORT Application Programming Guide](#).

Keeping or removing headers, trailers and relative records

You can use ICETOOL's SUBSET operator to create a subset of the input or output records with or without specific header, trailer or relative records. You use the following operands to tell SUBSET which records you want to keep or remove:

- **KEEP** - keep the specified records
- **REMOVE** - remove the specified records
- **INPUT** - keep or remove input records
- **OUTPUT** - keep or remove output records
- **HEADER** or **FIRST** - keep or remove the first record (header record)
- **HEADER(x)** or **FIRST(x)** - keep or remove the first x records (header records)
- **RRN(u)** - keep or remove relative record u
- **RRN(u,v)** - keep or remove relative records u through v
- **RRN(u,*)** - keep or remove relative records u through the last record
- **TRAILER** or **LAST** - keep or remove the last record (trailer record)
- **TRAILER(y)** or **LAST(y)** - keep or remove the last y records (trailer records)

The records that are selected (that is, kept or not removed) are written to the output data set identified by the TO operand. If appropriate, you can use the DISCARD operand to save the records that are not selected to a separate output data set identified by the DISCARD operand. You can create just the TO data set, just the DISCARD data set, or both.

You can use a USING data set to specify various DFSORT statements with SUBSET; see [z/OS DFSORT Application Programming Guide](#) for details.

Suppose you have input records that look like this:

```
MASTER03.IN
2008/04/23
Vicky
Frank
Regina
Viet
David
Dave
Carrie
Sam
Sri Hari
Martin
UPDATE03.OUT
```

You want to create an output data set with the the first two input records, the fifth through seventh input records, the eleventh input record, and the last input record. Write the following ICETOOL statement:

```
SUBSET FROM(INPUT) TO(OUTPUT) KEEP INPUT-
FIRST(2) RRN(5,7) RRN(11) LAST
```

The output records would look like this:

```
MASTER03.IN
2008/04/23
Regina
Viet
David
Sri Hari
UPDATE03.OUT
```

For complete details of SUBSET, as well as more examples, see [z/OS DFSORT Application Programming Guide](#).

Merging previously sorted data sets

You can use DFSORT's MERGE operator to create merged output data sets. A single MERGE operator can be used to merge up to 50 input data sets which were previously sorted by the same fields in the same locations. MERGE can create one output data set or up to 10 identical output data sets. Using INCLUDE or OMIT statements, you can select a subset of the input records. Using INREC or OUTREC statements, you can rearrange the fields of the input records. Using OUTFIL statements, you can create any number of output data sets with different subsets of records or arrangements of fields.

You can use up to 10 FROM operands to specify up to 50 input data sets. You must specify a USING data set to specify a DFSORT MERGE statement that tells MERGE the fields your input data sets are already sorted on. You can use various other DFSORT statements as well; see [z/OS DFSORT Application Programming Guide](#) for details.

Suppose you have two input data sets previously sorted by Branch Number as follows:

Input file1

```
0003 Sacramento
0005 Palo Alto
0008 Morgan Hill
```

Input file2

```
0002 Los Angeles
0006 Modesto
0009 San Jose
```

The following JCL and ICETOOL statements would merge the two input data sets on the Branch Number:

```
//MRG EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=*
//DFSMSG DD SYSOUT=*
//IN1 DD DSN=INPUT.FILE1,DISP=SHR
//IN2 DD DSN=INPUT.FILE2,DISP=SHR
//OUT DD DSN=OUTPUT.FILE,DISP=(NEW,CATLG,DELETE),
// UNIT=SYSDA,SPACE=(CYL,(5,5))
//TOOLIN DD *
MERGE FROM(IN1,IN2) TO(OUT) USING(CTL1)
/*
//CTL1CNTL DD *
MERGE FIELDS=(1,4,ZD,A)
/*
```

The output data set would have these merged records:

```
0002 Los Angeles
0003 Sacramento
0005 Palo Alto
0006 Modesto
0008 Morgan Hill
0009 San Jose
```

So far

So far in this chapter you have learned how to print statistics for numeric fields, create sorted and unsorted data sets, obtain a count of numeric fields in a range for a particular field, print fields from an input data set, print reports, print a count of field occurrences, select output records based on field occurrences, join fields from different data sets, match records from different data sets, sort records between headers and trailers, keep or remove headers, trailers or relative records, and merge previously sorted data sets. The last part of this chapter shows the complete main ICETOOL job and its resulting TOOLMSG output.

Complete ICETOOL job and TOOLMSG output

Here is the complete ICETOOL job you created in this chapter:

```

//EXAMP    JOB    A492,PROGRAMMER
//TOOL     EXEC   PGM=ICETOOL
//TOOLMSG  DD     SYSOUT=A
//DFSMSG   DD     SYSOUT=A
//TOOLIN   DD     *

* Statistics from all branches
STATS FROM(ALL) ON(18,4,ZD) ON(28,6,PD) ON(22,6,PD)
* Books from VALD and WETH
SORT FROM(BKS) TO(DAPUBS,PRPUBS) USING(S PUB)
* Separate output for California and Colorado branches
SORT FROM(ALL) USING(CACO)
* California branches profit analysis
  RANGE FROM(CADASD) ON(28,6,PD) HIGHER(-1500) LOWER(+8000)
* Branches with less than 32 employees
  RANGE FROM(ALL) ON(18,4,ZD) LOWER(32)
* Print profit, employees, and city for each Colorado branch
DISPLAY FROM(CODASD) LIST(OUT) ON(28,6,PD) ON(18,4,ZD) ON(1,15,CH)
* Print a report for the Colorado branches
DISPLAY FROM(CODASD) LIST(RPT) -
  DATE TITLE('Colorado Branches Report') PAGE -
  HEADER('City') HEADER('Profit') HEADER('Employees') -
  ON(1,15,CH) ON(28,6,PD) ON(18,4,ZD) BLANK BETWEEN(5) -
  TOTAL('Total') AVERAGE('Average') -
  MINIMUM('Lowest') COUNT('Number of cities')
* Print a report of books for individual publishers
DISPLAY FROM(DAPUBS) LIST(SECTIONS) -
  TITLE('BOOKS FOR INDIVIDUAL PUBLISHERS') PAGE -
  HEADER('TITLE OF BOOK') ON(1,35,CH) -
  HEADER('PRICE OF BOOK') ON(170,4,BI,C1,F'$') -
  BTITLE('PUBLISHER:') BREAK(106,4,CH) -
  B AVERAGE('AVERAGE FOR THIS PUBLISHER') -
  BTOTAL('TOTAL FOR THIS PUBLISHER') -
  AVERAGE('AVERAGE FOR ALL PUBLISHERS') -
  TOTAL('TOTAL FOR ALL PUBLISHERS')
* Print the count of books in use from each publisher
OCCUR FROM(BKIN) LIST(PUBCT) BLANK -
  TITLE('Books from Publishers') DATE(DMY.) -
  HEADER('Publisher') HEADER('Books Used') -
  ON(106,4,CH) ON(VALCNT,N05)
* Separate output containing records for publishers
* with more than 4 books in use
SELECT FROM(BKIN) TO(BKOUT) ON(106,4,CH) HIGHER(4)
* Reformat REGION.IN1 to T1 so it can be spliced
COPY FROM(REGNIN1) TO(T1) USING(CTL1)
* Reformat REGION.IN2 to T1 so it can be spliced
COPY FROM(REGNIN2) TO(T1) USING(CTL2)
* Splice records in T1 with matching ON fields
SPLICE FROM(T1) WITHALL -
  ON(5,5,CH) - Region
  WITH(1,4) - Office
  WITH(25,4) - Employees
  WITH(29,10) - Evaluation
  TO(REGNOUT)

/*
//ALL DD DSN=A123456.SORT.BRANCH,DISP=SHR
//BKS DD DSN=A123456.SORT.SAMPIN,DISP=SHR
// DD DSN=A123456.SORT.SAMPADD,DISP=SHR
//DAPUBS DD DSN=&DSRT,DISP=(,PASS),SPACE=(CYL,(2,2)),UNIT=SYSDA
//PRPUBS DD SYSOUT=A
//SPUBCNTL DD *
  SORT FIELDS=(106,4,A,1,75,A),FORMAT=CH
  INCLUDE COND=(106,4,EQ,C'VALD',OR,106,4,EQ,C'WETH'),
    FORMAT=CH

/*
//CACOCNTL DD *
  SORT FIELDS=(1,15,CH,A)
  OUTFIL FNAMES=(CADASD,CATAPE),INCLUDE=(16,2,CH,EQ,C'CA')
  OUTFIL FNAMES=(CODASD,COTAPE),INCLUDE=(16,2,CH,EQ,C'CO')

/*
//CADASD DD DSN=&CA,DISP=(,PASS),SPACE=(CYL,(2,2)),UNIT=3390
//CATAPE DD DSN=CA.BRANCH,UNIT=3480,VOL=SER=111111,
// DISP=(NEW,KEEP),LABEL=(,SL)
//CODASD DD DSN=&CO,DISP=(,PASS),SPACE=(CYL,(2,2)),UNIT=3390
//COTAPE DD DSN=CO.BRANCH,UNIT=3480,VOL=SER=222222,
// DISP=(NEW,KEEP),LABEL=(,SL)
//OUT DD SYSOUT=A
//RPT DD SYSOUT=A
//SECTIONS DD SYSOUT=A
//BKIN DD DSN=A123456.SORT.SAMPIN,DISP=SHR
//PUBCT DD SYSOUT=A

```

```

//BKOUT DD DSN=A123456.BOOKS1,DISP=(NEW,CATLG,DELETE),
// SPACE=(CYL,(3,3)),UNIT=3390
//REGNIN1 DD DSN=A123456.REGION.IN1,DISP=SHR
//REGNIN2 DD DSN=A123456.REGION.IN2,DISP=SHR
//T1 DD DSN=*&T1,UNIT=3390,SPACE=(CYL,(5,5)),DISP=(MOD,PASS)
//REGNOUT DD DSN=A123456.REGION.OUT,DISP=(NEW,CATLG,DELETE),UNIT=3390,
// SPACE=(CYL,(5,5))
//CTL1CNTL DD *
* Move REGION.IN1 fields to their locations for the
* output data set
  OUTREC FIELDS=(5:1,5,      Region
                  10:21,15,   Regional Director
                  39:6,15)    Headquarters
/*
//CTL2CNTL DD *
* Move REGION.IN2 fields to their locations for the
* output data set
  OUTREC FIELDS=(1:1,4,      Office
                  5:5,5,      Region
                  25:10,4,     Employees
                  29:14,10,    Evaluation
                  53:X)
/*

```

Here is the complete TOOLMSG data set produced by running this job:

```

ICE600I 0 DFSORT ICETOOL UTILITY RUN STARTED

ICE650I 0 VISIT http://www.ibm.com/storage/dfsor FOR ICETOOL PAPERS, EXAMPLES AND MORE

ICE632I 0 SOURCE FOR ICETOOL STATEMENTS: TOOLIN

ICE630I 0 MODE IN EFFECT: STOP

      * Statistics from all branches
      STATS FROM(ALL) ON(18,4,ZD) ON(28,6,PD) ON(22,6,PD)
ICE627I 0 DFSORT CALL 0001 FOR COPY FROM ALL TO E35 EXIT COMPLETED
ICE628I 0 RECORD COUNT: 000000000000012
ICE607I 0 STATISTICS FOR (18,4,ZD) :
ICE608I 0 MINIMUM: +000000000000015, MAXIMUM: +000000000000035
ICE609I 0 AVERAGE: +000000000000024, TOTAL : +000000000000298
ICE607I 0 STATISTICS FOR (28,6,PD) :
ICE608I 0 MINIMUM: -000000000004278, MAXIMUM: +000000000008276
ICE609I 0 AVERAGE: +000000000004222, TOTAL : +0000000000050665
ICE607I 0 STATISTICS FOR (22,6,PD) :
ICE608I 0 MINIMUM: +000000000012300, MAXIMUM: +0000000000042820
ICE609I 0 AVERAGE: +000000000027469, TOTAL : +0000000000329637
ICE602I 0 OPERATION RETURN CODE: 00

      * Books from VALD and WETH
      SORT FROM(BKS) TO(DAPUBS,PRPUBS) USING(SPUB)
ICE606I 0 DFSORT CALL 0002 FOR SORT FROM BKS TO OUTFIL USING SPUBCNTL COMPLETED
ICE602I 0 OPERATION RETURN CODE: 00

      * Separate output for California and Colorado branches
      SORT FROM(ALL) USING(CACO)
ICE606I 0 DFSORT CALL 0003 FOR SORT FROM ALL TO OUTFIL USING CACOCNTL COMPLETED
ICE602I 0 OPERATION RETURN CODE: 00

      * California branches profit analysis
      RANGE FROM(CADASD) ON(28,6,PD) HIGHER(-1500) LOWER(+8000)
ICE627I 0 DFSORT CALL 0004 FOR COPY FROM CADASD TO E35 EXIT COMPLETED
ICE628I 0 RECORD COUNT: 000000000000007
ICE631I 0 NUMBER OF VALUES IN RANGE FOR (28,6,PD) : 000000000000003
ICE602I 0 OPERATION RETURN CODE: 00

      * Branches with less than 32 employees
      RANGE FROM(ALL) ON(18,4,ZD) LOWER(32)
ICE627I 0 DFSORT CALL 0005 FOR COPY FROM ALL TO E35 EXIT COMPLETED
ICE628I 0 RECORD COUNT: 000000000000012
ICE631I 0 NUMBER OF VALUES IN RANGE FOR (18,4,ZD) : 000000000000008
ICE602I 0 OPERATION RETURN CODE: 00

* Print profit, employees, and city for each Colorado branch
  DISPLAY FROM(CODASD) LIST(OUT) ON(28,6,PD) ON(18,4,ZD) ON(1,15,CH)
ICE643I 0 WIDTH OF REPORT IS 0121 BYTES
ICE627I 0 DFSORT CALL 0006 FOR COPY FROM CODASD TO E35 EXIT COMPLETED
ICE603I 0 INFORMATION PRINTED IN OUT DATA SET
ICE628I 0 RECORD COUNT: 000000000000005
ICE602I 0 OPERATION RETURN CODE: 00

      * Print a report for the Colorado branches
      DISPLAY FROM(CODASD) LIST(RPT) -
      DATE TITLE('Colorado Branches Report') PAGE -
      HEADER('City') HEADER('Profit') HEADER('Employees') -
      ON(1,15,CH) ON(28,6,PD) ON(18,4,ZD) BLANK BETWEEN(5) -
      TOTAL('Total') AVERAGE('Average') -
      MINIMUM('Lowest') COUNT('Number of cities')
ICE643I 0 WIDTH OF REPORT IS 0121 BYTES
ICE627I 0 DFSORT CALL 0007 FOR COPY FROM CODASD TO E35 EXIT COMPLETED
ICE603I 0 INFORMATION PRINTED IN RPT DATA SET
ICE628I 0 RECORD COUNT: 000000000000005
ICE602I 0 OPERATION RETURN CODE: 00

```

Figure 4. Complete TOOLMSG Data Set. (Part 1 of 2)

```

* Print a report of books for individual publishers
DISPLAY FROM(DAPUBS) LIST(SECTIONS) -
  TITLE('BOOKS FOR INDIVIDUAL PUBLISHERS') PAGE -
  HEADER('TITLE OF BOOK') ON(1,35,CH) -
  HEADER('PRICE OF BOOK') ON(170,4,BI,C1,F'$') -
  BTITLE('PUBLISHER:') BREAK(106,4,CH) -
  BAVERAGE('AVERAGE FOR THIS PUBLISHER') -
  BTOTAL('TOTAL FOR THIS PUBLISHER') -
  AVERAGE('AVERAGE FOR ALL PUBLISHERS') -
  TOTAL('TOTAL FOR ALL PUBLISHERS')
ICE643I 0 WIDTH OF REPORT IS 0121 BYTES
ICE627I 0 DFSORT CALL 0008 FOR COPY FROM DAPUBS TO E35 EXIT COMPLETED
ICE603I 0 INFORMATION PRINTED IN SECTIONS DATA SET
ICE628I 0 RECORD COUNT: 000000000000019
ICE602I 0 OPERATION RETURN CODE: 00

* Print the count of books in use from each publisher
OCCUR FROM(BKIN) LIST(PUBCT) BLANK -
  TITLE('Books from Publishers') DATE(DMY.) -
  HEADER('Publisher') HEADER('Books Used') -
  ON(106,4,CH) ON(VALCNT)
ICE643I 0 WIDTH OF REPORT IS 0121 BYTES
ICE627I 0 DFSORT CALL 0009 FOR SORT FROM BKIN TO E35 EXIT COMPLETED
ICE603I 0 INFORMATION PRINTED IN PUBCT DATA SET
ICE628I 0 RECORD COUNT: 000000000000020
ICE638I 0 NUMBER OF RECORDS RESULTING FROM CRITERIA: 000000000000004
ICE602I 0 OPERATION RETURN CODE: 00

* Separate output containing records for publishers
* with more than 4 books in use
SELECT FROM(BKIN) TO(BKOUT) ON(106,4,CH) HIGHER(4)
ICE627I 0 DFSORT CALL 0010 FOR SORT FROM BKIN TO BKOUT COMPLETED
ICE628I 0 RECORD COUNT: 000000000000020
ICE638I 0 NUMBER OF RECORDS RESULTING FROM CRITERIA: 000000000000012
ICE602I 0 OPERATION RETURN CODE: 00

* Reformat REGION.IN1 to T1 so it can be spliced
COPY FROM(REGIN1) TO(T1) USING(CTL1)
ICE606I 0 DFSORT CALL 0011 FOR COPY FROM REGIN1 TO T1 USING CTL1CNTL COMPLETED
ICE602I 0 OPERATION RETURN CODE: 00

ICE601I 0 DFSORT ICETOOL UTILITY RUN ENDED - RETURN CODE: 00
* Reformat REGION.IN2 to T1 so it can be spliced
COPY FROM(REGIN2) TO(T1) USING(CTL2)
ICE606I 0 DFSORT CALL 0012 FOR COPY FROM REGIN2 TO T1 USING CTL2CNTL COMPLETED
ICE602I 0 OPERATION RETURN CODE: 00

* Splice records in T1 with matching ON fields
SPLICE FROM(T1) WITHALL -
  ON(5,5,CH) - Region
  WITH(1,4) - Office
  WITH(25,4) - Employees
  WITH(29,10) - Evaluation
  TO(REGNOUT)
ICE627I 0 DFSORT CALL 0013 FOR SORT FROM T1 TO REGNOUT COMPLETED
ICE628I 0 RECORD COUNT: 000000000000014
ICE638I 0 NUMBER OF RECORDS RESULTING FROM CRITERIA: 000000000000010
ICE602I 0 OPERATION RETURN CODE: 00

```

Figure 5. Complete TOOLMSG Data Set. (Part 2 of 2)

Summary

This topic introduced the 17 ICETOOL operators, and showed some of the ways you can use them to perform various tasks. For complete information on DFSORT's ICETOOL, refer to [z/OS DFSORT Application Programming Guide](#).

Part 4. Learning to use symbols

Chapter 13. Defining and using symbols

A *symbol* is a name (preferably something meaningful) you can use to represent a field or a constant. Sets of symbols, also called mappings, can be used to describe a group of related fields and constants such as the information in a particular type of record. Such mappings allow you to refer to fields, constants, and output columns by their symbols, freeing you from having to know the position, length and format of fields, the values of constants or the position of the output column you want to use.

DFSORT's symbol processing feature allows you or your site to create symbols for the fields in your own records and for constants associated with those fields. You can then use those symbols in DFSORT control statements and ICETOOL operators.

In addition, you can obtain predefined sets of symbols to use for the records created by other products such as RACF®, DCOLLECT and DFSMSrmm. Visit the [DFSORT Home Page \(www.ibm.com/storage/dfsor\)](http://www.ibm.com/storage/dfsor) for information about obtaining DFSORT symbol mappings for records produced by other products, and examples that use these symbols.

The chapter in this section explains how to define symbols for the bookstore data set and use them in several DFSORT control statements and ICETOOL operators.

Although not described in this book, you can also use:

- System symbols (for example, &SYSPLEX. and &JOBNAME.) in your symbol constants. For a full description of how symbols (including system symbols) can be used with DFSORT and ICETOOL, see [*z/OS DFSORT Application Programming Guide*](#).
- SET and PROC symbols included in the DFSORT and ICETOOL EXEC PARM string. For a full description of how SET and PROC symbols can be used with DFSORT and ICETOOL, see [*z/OS DFSORT Application Programming Guide*](#).

Creating the SYMNAMES data set

DFSORT and ICETOOL obtain the symbols to be used from the data set specified in a SYMNAMES DD statement. Create the SYMNAMES data set you want to use with RECFM=FB and LRECL=80 in the same way you would create a data set containing DFSORT JCL and control statements. Then use an editor, such as ISPF EDIT, to write the SYMNAMES statements defining your symbols, as described later in this section.

After you create the SYMNAMES data set, you can use it in any DFSORT or ICETOOL application for which you want to use the symbols you defined. You can add, delete and change SYMNAMES statements in the SYMNAMES data set at any time using the editor.

For this chapter, we will assume you have created a data set called SORT.SYMBOLS to put your SYMNAMES statements in.

Defining symbols for fields

Appendix B, “Descriptions of the sample data sets,” on page 173 shows the fields of the bookstore data set. To define the symbols for these fields, write a SYMNAMES statement for each one in SORT.SYMBOLS. The SYMNAMES statements for the symbols Title and Author_Last_Name look like this:

```
* Symbols for fields
Title,1,75,CH
Author_Last_Name,*,15,CH
```

To write these SYMNAMES statements that define the symbol *Title* for the Title field and the symbol *Author_Last_Name* for the Author's Last Name field, follow these steps:

Table 82. Steps to Define Symbols for Fields

Step	Action
1	Write a comment statement (optional): * Symbols for fields
2	Type Title followed by a comma. This is the symbol you will use for the Title field. A symbol can be 1 to 50 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 (-). However, the first character must not be a number. Title, TITLE, and title are three different symbols.
3	Type the position of the Title field followed by a comma. The position of the Title field is 1 .
4	Type the length of the Title field followed by a comma. The length of the Title field is 75 .
5	Type the format of the Title field followed by a blank. The format of the Title field is CH .
6	Type Author_Last_Name followed by a comma. This is the symbol you will use for the Author's Last Name field.
7	Type * for the position followed by a comma. An asterisk (*) for the position shows that this field immediately follows the previous field. The * here will automatically assign 76 as the position of the Author_Last_Name symbol. You could specify 76 instead of *, but * is preferable when fields are adjacent. * allows symbols for fields to be inserted without changes to symbols for other fields.
8	Type the length of the Author's Last Name field followed by a comma. The length of the Author's Last Name field is 15 .
9	Type the format of the Author's Last Name field followed by a blank. The format of the Author's Last Name field is CH .

The SYMNAMES statements to define the symbols for the other fields in the bookstore data set look like this:

```
Author_First_Name,*,15,CH
Publisher,*,4,CH
Course_Department,*,5,CH
Course_Number,*,5,CH
Course_Name,*,25,CH
Instructor_Last_Name,*,15,CH
Instructor_Initials,*,2,CH
Number_in_Stock,*,4,BI
Number_Sold_YTD,*,4,BI
Price,*,4,BI
```

So far

So far, you have learned how to create a SYMNAMES data set and use it to define symbols for your fields. Now you will learn how to use the symbols for the fields you defined.

Using symbols for fields in DFSORT statements

Now that you have your symbols for the bookstore data set defined in SORT.SYMBOLS, you can use those symbols in DFSORT control statements wherever fields can appear.

Here's an example of a DFSORT application that uses symbols. It selects the books for courses 00032 and 10347 from SORT.SAMPIN and sorts them by title, instructor and price:

```
//SYM1 JOB A492,PROGRAMMER
//SORTIT EXEC PGM=SORT
//SYSOUT DD SYSOUT=A
//SYMNAMES DD DSN=A123456.SORT.SYMBOLS,DISP=SHR
//SYMNOUT DD SYSOUT=*
//SORTIN DD DSN=A123456.SORT.SAMPIN,DISP=SHR
//SORTOUT DD DSN=A123456.SORT.SAMPOUT,DISP=SHR
//SYSIN DD *
    INCLUDE COND=(Course_Number,EQ,C'00032',OR,
                  Course_Number,EQ,C'10347')
    SORT FIELDS=(Title,A,
                  Instructor_Last_Name,A,Instructor_Initials,A,
                  Price,A)
/*
```

The SYMNAMES DD statement specifies the SYMNAMES data set to be used for this application.

The SYMNOUT DD statement specifies a data set in which you want DFSORT to list your original SYMNAMES statements and the symbol table constructed from them. You can omit the SYMNOUT data set if you don't want to see that information. For this SYMNAMES data set, the SYMNOUT listing looks like this:

```
----- ORIGINAL STATEMENTS FROM SYMNAMES -----
* Symbols for fields
Title,1,75,CH
Author_Last_Name,*,15,CH
Author_First_Name,*,15,CH
Publisher,*,4,CH
Course_Department,*,5,CH
Course_Number,*,5,CH
Course_Name,*,25,CH
Instructor_Last_Name,*,15,CH
Instructor_Initials,*,2,CH
Number_in_Stock,*,4,BI
Number_Sold_YTD,*,4,BI
Price,*,4,BI

----- SYMBOL TABLE -----
Title,1,75,CH
Author_Last_Name,76,15,CH
Author_First_Name,91,15,CH
Publisher,106,4,CH
Course_Department,110,5,CH
Course_Number,115,5,CH
Course_Name,120,25,CH
Instructor_Last_Name,145,15,CH
Instructor_Initials,160,2,CH
Number_in_Stock,162,4,BI
Number_Sold_YTD,166,4,BI
Price,170,4,BI
```

The INCLUDE and SORT statements use symbols for the fields instead of position, length and format.

You can define symbols for any %nnn, %nn or %n parsed fields you use. For example, you could specify these SYMNAMES statements to define symbols for %01 and %02:

```
First_name,%01
Last_name,%02
```

and then use these symbols in the following OUTREC statement:

```
OUTREC PARSE=(First_name=(ABSP0S=9,FXLEN=12,ENDBEFR=C','),
               Last_name=(FIXLEN=15,ENDBEFR=C',')),
        BUILD=(First_name,15:Last_name)
```

Using symbols for fields in ICETOOL operators

You can also use the symbols from SORT.SYMBOLS in ICETOOL operators and their associated DFSORT control statements wherever fields can appear. Here's an example of an ICETOOL application that uses

symbols for fields. It does what the DFSORT example in [“Using symbols for fields in DFSORT statements”](#) on page 166 does, but also shows the number of selected books priced below 7 dollars and the number priced above 20 dollars:

```
//SYM2 JOB A492,PROGRAMMER
//TOOL EXEC PGM=ICETOOL
//TOOLMSG DD SYSOUT=A
//DFSMSG DD SYSOUT=A
//SYMNAMES DD DSN=A123456.SORT.SYMBOLS,DISP=SHR
//SYMNOUT DD SYSOUT=*
//IN DD DSN=A123456.SORT.SAMPIN,DISP=SHR
//OUT DD DSN=A123456.SORT.SAMPOUT,DISP=OLD
//TOOLIN DD *
  SORT FROM(IN) TO(OUT) USING(CTL1)
  RANGE FROM(OUT) ON(Price) LOWER(+700)
  RANGE FROM(OUT) ON(Price) HIGHER(+2000)
//CTL1CNTL DD *
  INCLUDE COND=(Course_Number,EQ,C'00032',OR,
                Course_Number,EQ,C'10347')
  SORT FIELDS=(Title,A,
                Instructor_Last_Name,A,Instructor_Initials,A,
                Price,A)
/*
```

The SYMNAMES DD statement specifies the SYMNAMES data set to be used for this application.

The SYMNOUT DD statement specifies a data set in which you want ICETOOL to list your original SYMNAMES statements and the symbol table constructed from them. Because the SYMNAMES data set is the same as for the DFSORT application shown in [“Using symbols for fields in DFSORT statements”](#) on page 166, the SYMNOUT listing will be the same as well.

The RANGE operators and the INCLUDE and SORT statements use symbols for the fields instead of position, length and format.

So far

You have now learned how to use symbols for fields in DFSORT control statements and ICETOOL operators. Next, you will learn how to define and use symbols for your constants.

Defining and using symbols for constants

You can use symbols wherever decimal constants, character constants, hexadecimal constants or bit constants can appear in DFSORT control statements and ICETOOL operators.

The ICETOOL example in [“Using symbols for fields in ICETOOL operators”](#) on page 167 uses the following RANGE operators:

```
RANGE FROM(OUT) ON(Price) LOWER(+700)
RANGE FROM(OUT) ON(Price) HIGHER(+2000)
```

and the following INCLUDE statement:

```
INCLUDE COND=(Course_Number,EQ,C'00032',OR,
              Course_Number,EQ,C'10347')
```

+700 and +2000 are decimal constants. C'00032' and C'10347' are character constants. You can use symbols for these constants to make them more understandable in the ICETOOL operators and DFSORT control statements you write. The SYMNAMES statements for these constants look like this:

```
* Symbols for constants
Discount,+700
Premium,+2000
Beginning_Economics,C'00032'
Advanced_Sociology,C'10347'
```

To add these symbols for the four constants to the SORT.SYMBOLS data set, follow these steps:

Table 83. Steps to define symbols for constants

Step	Action
1	Write a comment statement (optional): * Symbols for constants
2	Type Discount followed by a comma. This is the symbol you will use for the decimal constant +700.
3	Type the constant followed by a blank. The constant is +700 . You can also use 700 , but we recommend using a + sign for positive decimal constants.
4	Type Premium followed by a comma. This is the symbol you will use for the decimal constant +2000 .
5	Type the constant followed by a blank. The constant is +2000 . You can also use 2000 , but we recommend using a + sign for positive decimal constants.
6	Type Beginning_Economics followed by a comma. This is the symbol you will use for the character constant C'00032'.
7	Type the constant followed by a blank. The constant is C'00032' . You can also use '00032' or c'00032' .
8	Type Advanced_Sociology followed by a comma. This is the symbol you will use for the character constant C'10347'.
9	Type the constant followed by a blank. The constant is C'10347' . You can also use '10347' or c'10347' .

The RANGE operators can now be written as:

```
RANGE FROM(OUT) ON(Price) LOWER(Discount)
RANGE FROM(OUT) ON(Price) HIGHER(Premium)
```

The INCLUDE statement can now be written as:

```
INCLUDE COND=(Course_Number,EQ,Beginning_Economics,OR,
              Course_Number,EQ,Advanced_Sociology)
```

Summary

This chapter covered how to define and use symbols for fields and constants in DFSORT control statements and ICETOOL operators. For more information on DFSORT symbols, see [z/OS DFSORT Application Programming Guide](#).

This completes the [z/OS DFSORT: Getting Started](#) tutorials. The appendixes that follow contain important information about the Sample Data Sets and the order in which DFSORT processes its various control statements.

Appendix A. Creating the sample data sets

Many of the examples in this document use A123456.SORT.SAMPIN, A123456.SORT.SAMPADD, and A123456.SORT.BRANCH as input data sets, and A123456.SORT.SAMPOUT as an output data set. If you want to try the examples in this document that use these data sets, create your own userid.SORT.SAMPIN, userid.SORT.SAMPADD, userid.SORT.BRANCH and userid.SORT.SAMPOUT data sets with your own userid by running the ICEDCRE job that follows.

Substitute your own JOB statement for the ICEDCRE JOB statement. Substitute your own userid for 'userid' in the four DD statements.

If you use a JOBLIB or STEPLIB to run DFSORT jobs, add it to the ICEDCRE job. If you prefer a value for UNIT other than SYSALLDA, change that parameter in the DD statements. Add a VOL=SER=vvvvvv parameter to the DD statements, if appropriate.

```
//ICEDCRE JOB A492,PROGRAMMER
//SAMPCOPY EXEC PGM=ICESAMP,PARM=(BOOKS)
//SYSPRINT DD SYSOUT=*
//SAMPLE DD DSN=userid.SORT.SAMPIN,DISP=(NEW,CATLG),
//        SPACE=(TRK,(1,1),RLSE),
//        DCB=(RECFM=FB,LRECL=173,BLKSIZE=1730),
//        UNIT=SYSALLDA
//*
//ADD DD DSN=userid.SORT.SAMPADD,DISP=(NEW,CATLG),
//      SPACE=(TRK,(1,1),RLSE),
//      DCB=(RECFM=FB,LRECL=173,BLKSIZE=1730),
//      UNIT=SYSALLDA
//*
//OUTPUT DD DSN=userid.SORT.SAMPOUT,DISP=(NEW,CATLG),
//         SPACE=(TRK,(1,1),RLSE),
//         DCB=(RECFM=FB,LRECL=173,BLKSIZE=1730),
//         UNIT=SYSALLDA
//*
//BRANCH DD DSN=userid.SORT.BRANCH,DISP=(NEW,CATLG),
//        SPACE=(TRK,(1,1),RLSE),
//        DCB=(RECFM=FB,LRECL=33,BLKSIZE=330),
//        UNIT=SYSALLDA
//*
//SYSIN DD DUMMY
```

Once you run this job successfully, you can substitute your userid for A123456 in the data set names of these four data sets. For example, if you see a DD statement like this:

```
//SORTIN DD DSN=A123456.SORT.SAMPIN,DISP=SHR
```

you can use your own DD statement like this:

```
//SORTIN DD DSN=userid.SORT.SAMPIN,DISP=SHR
```

Some of the examples use data sets other than A123456.SORT.SAMPIN, A123456.SORT.SAMPOUT, A123456.SORT.SAMPADD, and A123456.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.

Appendix B. Descriptions of the sample data sets

The fields in the records of the sample data sets SORT.SAMPIN, SORT.SAMPADD and SORT.BRANCH are described on this page, and the contents of the records are described on the following pages.

Table 84 on page 173 shows the length and data format of each field in the sample bookstore data set SORT.SAMPIN, and in the additional bookstore data set SORT.SAMPADD. Both of these data sets have fixed-length records (RECFM=FB) that are 173 bytes long (LRECL=173).

Table 84. SORT.SAMPIN and SORT.SAMPADD Field Descriptions

Field	Length	Data Format
Title	75	CH
Author's Last Name	15	CH
Author's First Name	15	CH
Publisher	4	CH
Course Department	5	CH
Course Number	5	CH
Course Name	25	CH
Instructor's Last Name	15	CH
Instructor's Initials	2	CH
Number In Stock	4	BI
Number Sold Y-to-D	4	BI
Price	4	BI

Table 85 on page 173 shows the length and data format of each field in the sample branch data set SORT.BRANCH. This data set has fixed-length records (RECFM=FB) that are 33 bytes long (LRECL=33).

Table 85. SORT.BRANCH Field Descriptions

Field	Length	Data Format
City	15	CH
State	2	CH
Employees	4	ZD
Revenue	6	PD
Profit	6	PD

The contents of the records in the sample data sets are shown on the following pages. Each horizontal line represents a record, and each column represents a field. Headings show the field names and the starting and ending position of each field. Of course, these headings do not actually appear in the data sets. Numeric fields (BI, ZD, PD) would not actually be readable if displayed, but are shown as their equivalent readable values.

Sample Data Set - SORT.SAMPIN

Sample Bookstore Data Sets

Sample Data Set - SORT.SAMPIN

Book Title		Author's Last Name	Author's First Name	Publisher	Course Department
1	75	76 90	91 105	106 109	110 114
COMPUTER LANGUAGES		MURRAY	ROBERT	FERN	COMP
LIVING WELL ON A SMALL BUDGET		DEWAN	FRANK	COR	
SUPPLYING THE DEMAND		MILLER	TOM	COR	BUSIN
VIDEO GAME DESIGN		RASMUSSEN	LORI	VALD	COMP
INKLINGS: AN ANTHOLOGY OF YOUNG POETS		WILDE	KAREN	COR	ENGL
COMPUTERS: AN INTRODUCTION		DINSHAW	JOKII	WETH	COMP
PICK'S POCKET DICTIONARY		GUSTLIN	CAROL	COR	
EDITING SOFTWARE MANUALS		OJALVO	VICTOR	VALD	ENGL
NUMBERING SYSTEMS		BAYLESS	WILLIAM	FERN	COMP
STRATEGIC MARKETING		YAEGER	MARK	VALD	BUSIN
THE INDUSTRIAL REVOLUTION		GROSS	DON	WETH	HIST
MODERN ANTHOLOGY OF WOMEN POETS		COWARD	PETER	COR	ENGL
INTRODUCTION TO PSYCHOLOGY		DUZET	LINDA	COR	PSYCH
THE COMPLETE PROOFREADER		GREEN	ANN	FERN	ENGL
SYSTEM PROGRAMMING		CAUDILLO	RAUL	WETH	COMP
SHORT STORIES AND TALL TALES		AVRIL	LILIANA	VALD	ENGL
INTRODUCTION TO BIOLOGY		WU	CHIEN	VALD	BIOL
ADVANCED TOPICS IN PSYCHOANALYSIS		OSTOICH	DIANNE	FERN	PSYCH
EIGHTEENTH CENTURY EUROPE		MUNGER	ALICE	WETH	HIST
CRISES OF THE MIDDLE AGES		BENDER	GREG	COR	HIST

Sample Data Set - SORT.SAMPADD

Sample Data Set - SORT.SAMPADD

Book Title		Author's Last Name	Author's First Name	Publisher	Course Department
1	75	76 90	91 105	106 109	110 114
GUNTHER'S GERMAN DICTIONARY		WILLIS	GUNTER	WETH	
COMPLETE SPANISH DICTIONARY		ROBERTS	ANGEL	VALD	
ANOTHER ITALIAN DICTIONARY		UNDER	JOAN	COR	
FRENCH TO ENGLISH DICTIONARY		JONES	JACK	FERN	
GUIDE TO COLLEGE LIFE		LAMB	CHARLENE	WETH	
THE ANIMAL KINGDOM		YOUNG	KEVIN	COR	BIOL
A SMALLER WORLD: MICROBES		BEESELY	GEORGE	FERN	BIOL
DNA: BLUEPRINT FOR YOU		IAVERS	ILSE	FERN	BIOL
CELLS AND HOW THEY WORK		JETTS	PETER	VALD	BIOL
KNOW YOUR CONSUMER		ZANE	JENNIFER	COR	BUSIN
ANTICIPATING THE MARKET		ALLEN	CLYDE	WETH	BUSIN
ZEN BUSINESS		WILLIAMS	KATIE	VALD	BUSIN
THE ART OF TAKEOVERS		HUNT	ROBERT	FERN	BUSIN
THE TOY STORE TEST		LITTLE	MARIE	COR	COMP
NOVEL IDEAS		PETERS	SETH	VALD	ENGL
POLITICS AND HISTORY		TOMPSOM	KEN	FERN	HIST
CIVILIZATION SINCE ROME FELL		PIERCE	NICOLE	WETH	HIST
REBIRTH FROM ITALY		FISH	JOHN	WETH	HIST
FREUD'S THEORIES		GOOLE	APRIL	VALD	PSYCH
MAP OF THE HUMAN BRAIN		WINTER	POLLY	COR	PSYCH
QUEUE THEORY		FOX	THAD	FERN	BUSIN
DESIGNING APPLICATIONS		STEVENS	NOAH	COR	COMP

Sample Data Set - SORT.SAMPIN (continued)

Sample Data Set - SORT.SAMPIN (continued)

Course Number	Course Name	Instructor's Last Name	Instructor's Initials	Number In Stock	Number Sold Year-to-Date	Price
115 119	120 144	145 159	160 161	162 165	166 169	170 173

Sample Data Set - SORT.SAMPIN (continued) (continued)

Course Number	Course Name	Instructor's Last Name	Instructor's Initials	Number In Stock	Number Sold Year-to-Date	Price
00032	INTRO TO COMPUTERS	CHATTERJEE	CL	5	29	2600
70251	MARKETING	MAXWELL	RF	14	1	9900
00205	VIDEO GAMES	NEUMANN	LB	0	32	1925
10856	MODERN POETRY	FRIEDMAN	KR	10	10	2199
00032	INTRO TO COMPUTERS	CHATTERJEE	CL	2	32	595
				20	26	1899
				46	38	295
10347	TECHNICAL EDITING	MADRID	MM	13	32	1450
00032	INTRO TO COMPUTERS	CHATTERJEE	AN	6	27	360
70124	ADVANCED MARKETING	LORCH	MH	3	35	2350
50420	WORLD HISTORY	GOODGOLD	ST	15	9	795
10856	MODERN POETRY	FRIEDMAN	KR	1	26	450
30016	PSYCHOLOGY I	ZABOSKI	RL	26	15	2200
10347	TECHINICAL EDITING	MADRID	MM	7	19	625
00103	DATA MANAGEMENT	SMITH	DC	4	23	3195
10054	FICTION WRITING	BUCK	GR	10	9	1520
80521	BIOLOGY I	GREENBERG	HC	6	11	2350
30975	PSYCHOANALYSIS	NAKATSU	FL	1	12	2800
50632	EUROPEAN HISTORY	BISCARDI	HR	23	21	1790
50521	WORLD HISTORY	WILLERTON	DW	14	17	1200

Sample Data Set - SORT.SAMPADD (continued)

Sample Data Set - SORT.SAMPIN (continued)

Course Number	Course Name	Instructor's Last Name	Instructor's Initials	Number In Stock	Number Sold Year-to-Date	Price
115 119	120 144	145 159	160 161	162 165	166 169	170 173
				3	16	1088
				8	4	650
				26	6	925
				7	17	1100
				20	1	2000
80522	BIOLOGY II	HAROLD	LM	2	30	3000
80522	BIOLOGY II	HAROLD	LM	9	20	1955
80523	INTRO TO GENETICS	ABRAHAM	NG	15	21	2195
80523	INTRO TO GENETICS	ABRAHAM	NG	8	46	2495
70251	MARKETING	MAXWELL	RF	16	3	4500
70124	ADVANCED MARKETING	LORCH	MH	20	25	2000
70255	BUSINESS THEORY	SCHOFFE	KN	12	12	1200
70255	BUSINESS THEORY	SCHOFFE	KN	17	15	615
00205	VIDEO GAMES	NEUMANN	LB	15	12	2600
10054	FICTION WRITING	BUCK	GR	11	25	2450
50521	WORLD HISTORY	WILLERTON	DW	3	17	995
50420	WORLD HISTORY	GOODGOLD	ST	6	15	1350
50632	EUROPEAN HISTORY	BISCARDI	HR	10	20	2560
30975	PSYCHOANALYSIS	NAKATSU	FL	12	15	1250
30016	PSYCHOLOGY I	ZABOSKI	RL	6	9	895
70255	BUSINESS THEORY	SCHOFFE	KN	2	20	1500
00103	DATA MANAGEMENT	SMITH	DC	7	15	1435

Sample Data Set - SORT.BRANCH

Sample Data Set - SORT.BRANCH

City	State	Employees	Revenue	Profit
1 15	16 17	18 21	22 27	28 33

Sample Bookstore Data Sets

Sample Data Set - SORT.BRANCH (*continued*)

City	State	Employees	Revenue	Profit
Los Angeles	CA	32	22530	-4278
San Francisco	CA	35	42820	6832
Fort Collins	CO	22	12300	-2863
Sacramento	CA	29	42726	8276
Sunnyvale	CA	18	16152	-978
Denver	CO	33	31876	6288
Boulder	CO	32	33866	7351
Morgan Hill	CA	15	18200	3271
Vail	CO	19	23202	5027
San Jose	CA	21	27225	8264
San Diego	CA	22	32940	8275
Aspen	CO	20	25800	5200

Appendix C. Processing order of control statements

The following flowchart shows the order in which control statements are processed. (SUM is processed at the same time as SORT or MERGE. It is not used with COPY.)

Although you can write the statements in any order, DFSORT always processes the statements in the following order.

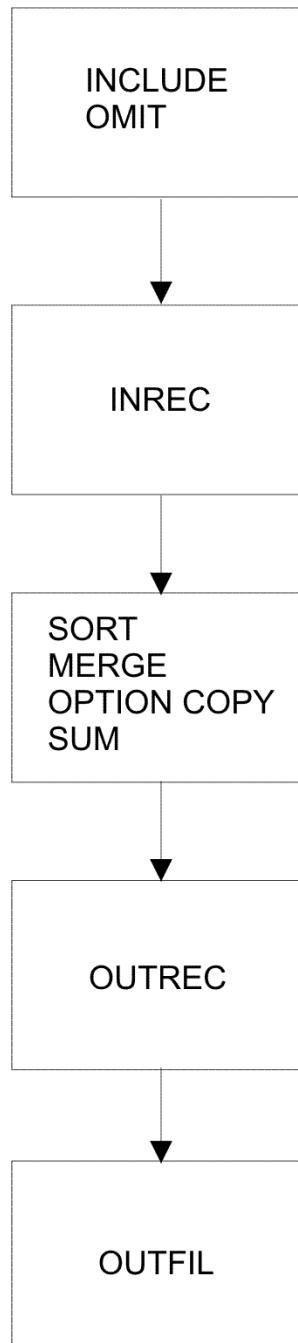


Figure 6. Processing Order of Control Statements

Appendix D. Accessibility

Accessible publications for this product are offered through [IBM Documentation for z/OS \(www.ibm.com/docs/en/zos\)](http://www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS documentation see [How to Send Feedback to IBM](#) to leave documentation feedback.

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, 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 \(www.ibm.com/software/support/systemsz/lifecycle\)](http://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](http://www.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 \(www.ibm.com/legal/copytrade.shtml\)](http://www.ibm.com/legal/copytrade.shtml).

Index

A

- accessibility
 - contact IBM [179](#)
- allowable comparisons [26](#)
- Arithmetic [48](#)
- ascending order
 - example [4](#)
 - sorting [11](#)
- ASCII data format code [4](#)
- assistive technologies [179](#)

B

- BI format [4](#)
- binary data [4](#)
- binary display [48](#)
- binary zeros [39](#)
- blanks [40](#)

C

- calling DFSORT
 - COBOL program [109](#)
 - PL/I program [113](#)
- carriage control, ANSI [88](#), [89](#), [93](#)
- CH format [4](#)
- character data [4](#)
- character strings, format [41](#)
- COBOL
 - calling DFSORT [109](#)
 - FASTSORT compile-time option [119](#)
 - MERGE statement [109](#)
 - passing DFSORT statements [109](#)
 - sample program [110](#), [111](#)
 - SORT statement [109](#)
- comparison
 - allowable [26](#)
 - field-to-constant [23](#), [26](#)
 - field-to-field [23](#), [26](#)
 - numeric [29](#)
 - operators [23](#)
- constant
 - comparisons [28](#)
 - reformatting [41](#)
- contact
 - z/OS [179](#)
- continuing a statement
 - DFSORT [15](#)
 - ICETOOL [127](#)
- control fields
 - combining [14](#)
 - deleting duplicate records [35](#)
 - duplicate [33](#)
 - equally collating [8](#), [33](#)
 - general information [7](#)
 - multiple [13](#)

- control fields (*continued*)
 - overlapping [13](#)
 - reordering [39](#)
 - summing [33](#)
- control statements
 - DFSORT [6](#)
 - ordering [24](#)
 - processing order flowchart [177](#)
- converting numeric fields [44](#)
- COPY operator (ICETOOL) [133](#)
- COPY option [21](#)
- copying records [21](#)
- COUNT operator (ICETOOL) [123](#)
- creating a DFSORT job [6](#)

D

- data formats [4](#)
- data set
 - copying
 - definition [5](#)
 - with DFSORT [21](#)
 - with ICETOOL [133](#)
 - joining
 - definition [5](#)
 - making multiple copies [79](#), [80](#)
 - merging
 - definition [4](#)
 - with DFSORT [17](#)
 - sorting
 - ascending order [11](#)
 - by multiple control fields [13](#)
 - definition [4](#)
 - descending order [11](#)
 - with DFSORT [11](#)
 - with ICETOOL [128](#)
- DATASORT operator (ICETOOL) [123](#)
- dataspace sorting [117](#)
- defaults
 - order of equal records [8](#)
 - overriding
 - using OPTION statement [116](#)
 - using PARM parameter [115](#)
- DEFAULTS operator (ICETOOL) [123](#)
- deleting fields [38](#)
- deleting records [33](#)
- descending order
 - example [4](#)
 - sorting [11](#)
- DFSORT web site [3](#)
- DFSPARM [16](#)
- discards, saving [82](#)
- DISPLAY operator (ICETOOL) [135](#)
- duplicate fields
 - OCCUR operator (ICETOOL) [144](#)
 - SELECT operator (ICETOOL) [146](#)
 - SPLICE operator (ICETOOL) [150](#)

duplicate fields (*continued*)
SUM statement [112](#)
UNIQUE operator (ICETOOL) [124](#)

E

EBCDIC data format code [4](#)
edit masks
ICETOOL [138](#)
OUTFIL [46–48](#)
editing fields [37, 138](#)
equal control fields [8](#)
excluding records [26](#)

F

FASTSORT
COBOL [112](#)
compile-time option [119](#)
FB to VB [98](#)
FI format [5](#)
field-to-constant comparison [23, 26](#)
field-to-field comparison [23, 26](#)
fields
character and numeric, printing [135](#)
deleting unnecessary fields [38](#)
editing [39, 138](#)
printing statistics for numeric [126](#)
reformatting [39](#)
fixed fields [37](#)
Fixed-point data [5](#)
Floating sign data [5](#)
format [20](#)
formats [5](#)
formats for writing constants [28](#)
formatting fields [37, 138](#)
FS format [5](#)

H

headers
sorting records between [155](#)
hexadecimal display [48](#)
hexadecimal strings, format [41](#)
high-speed disk [117](#)
Hipersorting [117](#)

I

ICETOOL
complete sample job [157](#)
COPY operator [133](#)
COUNT operator [123](#)
creating a job [124](#)
DATASORT operator [123](#)
DEFAULTS operator [123](#)
definition [123](#)
DISPLAY operator [135](#)
JCL statements [125](#)
job
elements [124](#)
sample [157](#)
MERGE operator [123](#)

ICETOOL (*continued*)
MODE operator [123](#)
OCCUR operator [144](#)
operator summary [123](#)
RANGE operator [123, 133](#)
requirements
input data sets [124](#)
JCL [124](#)
RESIZE operator [124](#)
SELECT operator [146](#)
SPLICE operator [150](#)
statements
blank [125](#)
comment [125](#)
operator, continuing [127](#)
STATS operator [126](#)
SUBSET operator [124](#)
symbols [167](#)
TOOLMSG output, sample [157](#)
UNIQUE operator [124](#)
VERIFY operator [124](#)
improving performance
allocating main storage [117](#)
FASTSORT compile-time option [119](#)
high-speed disk [117](#)
Hiperspace [117](#)
INCLUDE statement [118](#)
INREC statement [118](#)
JCL (job control language) [119](#)
OMIT statement [118](#)
options to avoid [119](#)
SKIPREC option [118](#)
STOPAFT option [118](#)
SUM statement [118](#)
INCLUDE statement [23](#)
INREC statement [37](#)
installation defaults, overriding [115](#)

J

JCL (job control language)
calling DFSORT from a program [109, 111, 113](#)
executing a copy [21](#)
executing a merge [19](#)
executing a sort [16](#)
general information [16](#)
ICETOOL [125](#)
selecting input data sets [16](#)
selecting output data sets [16](#)
statements [16](#)
JOB statement [16](#)
joining [150](#)
justifying data [52](#)

K

keyboard
navigation [179](#)
PF keys [179](#)
shortcut keys [179](#)

L

lookup and change [51](#)
lowercase to uppercase [43](#)

M

main storage, allocating [117](#)
making multiple data set copies [79](#)
mapping
 symbols [165](#)
matching [152](#)
memory object sorting [117](#)
MERGE operator (ICETOOL) [123](#)
MERGE statement [18](#)
MERGE statement, COBOL [109](#)
merging records [17](#)
MODE operator (ICETOOL) [123](#)
multiple control fields [13](#)
multiple data sets [79](#)

N

navigation
 keyboard [179](#)
non-duplicate fields
 OCCUR operator (ICETOOL) [144](#)
 SELECT operator (ICETOOL) [146](#)
 SPLICE operator (ICETOOL) [150](#)
 SUM statement [112](#)
 UNIQUE operator (ICETOOL) [124](#)
number of digits
 ICETOOL [140](#)
numeric tests [29](#)

O

OCCUR operator (ICETOOL) [144](#)
occurrences
 OCCUR operator (ICETOOL) [144](#)
 SELECT operator (ICETOOL) [146](#)
 SPLICE operator (ICETOOL) [150](#)
OMIT statement [23](#)
OPTION COPY [21](#)
OPTION statement [21](#), [116](#)
order of control statements [24](#)
ordering
 ascending order [4](#)
 control statements [24](#)
 descending order [4](#)
 DFSORT defaults [8](#)
OUTFIL
 edit mask patterns [46–48](#)
OUTFIL statement [79](#)
output records, reformatting [37](#)
OUTREC statement [37](#)
overflow
 explanation [36](#)
 preventing [66](#)
overriding installation defaults [115](#)

P

packed decimal data [4](#)
PARM parameter [115](#)
parsing records [73](#)
passing DFSORT statements
 from a COBOL program [109](#)
 from a PL/I program [109](#)
PD format [4](#)
PL/I
 calling DFSORT [113](#)
 passing DFSORT statements [109](#)
 sample program [113](#)
printing reports
 DISPLAY operator (ICETOOL) [135](#)
 OUTFIL statement [88](#)
processing order
 differences between INREC and OUTREC [37](#)
 flowchart [177](#)
 special considerations for INREC [65](#), [66](#)

R

RANGE operator (ICETOOL) [123](#), [133](#)
record length, changing [38](#)
RECORD statement [113](#)
records
 considerations when reordering with INREC and OUTREC [65](#), [66](#)
 copying
 definition [5](#)
 writing the COPY statement [21](#)
 joining
 definition [5](#)
 merging
 definition [4](#)
 using program control statements [17](#)
 reformatting with OUTREC [23](#)
 reordering and reformatting with OUTREC [39](#)
 reordering with OUTREC [23](#)
 selecting by occurrences [146](#)
 sorting
 ascending order [11](#)
 between headers and trailers [155](#)
 by multiple control fields [13](#)
 definition [4](#)
 descending order [11](#)
 writing the SORT statement [11](#)
 summing [33](#)
reformatting records [37](#), [73](#)
regular expressions [30](#)
reordering fields [39](#)
repeating [85](#)
reports
 DISPLAY operator (ICETOOL) [135](#)
 OUTFIL statement [88](#)
RESIZE operator (ICETOOL) [124](#)
running a DFSORT job [6](#)

S

sample data sets [7](#), [80](#), [124](#), [171](#), [173](#)
sampling [80](#)

- saving discards [82](#)
- search
 - substring [30](#)
- sections [94](#), [142](#)
- SELECT operator (ICETOOL) [146](#)
- selecting records [146](#)
- sequence numbers [67](#)
- SFF format [4](#)
- short fields [20](#), [32](#), [36](#)
- shortcut keys [179](#)
- SKIPREC option [118](#)
- SORT operator (ICETOOL) [128](#)
- SORT statement [11](#)
- SORT statement, COBOL [109](#)
- SORT-CONTROL special register (COBOL) [110](#)
- SORT-RETURN special register (COBOL) [110](#)
- sorting records
 - DFSORT [11](#)
 - ICETOOL [128](#), [155](#)
- specifying COPY
 - COPY operator (ICETOOL) [133](#)
 - MERGE statement [21](#)
 - OPTION statement [21](#)
 - SORT statement [21](#)
- SPLICE operator (ICETOOL) [150](#)
- splitting [86](#)
- squeezing data [54](#)
- statistics, printing for numeric fields [91](#), [126](#), [136](#)
- STATS operator (ICETOOL) [126](#)
- STOPAFT option [118](#)
- SUBSET operator (ICETOOL) [124](#)
- substring search [30](#)
- SUM statement [33](#)
- summary fields [33](#)
- summary of changes [xix](#)
- symbols [165](#)

T

- trailers
 - sorting records between [155](#)

U

- UFF format [4](#)
- Unicode comparisons [31](#)
- UNIQUE operator (ICETOOL) [124](#)
- uppercase to lowercase [43](#)
- user interface
 - ISPF [179](#)
 - TSO/E [179](#)

V

- variable fields [73](#)
- VB data sets [20](#), [32](#), [36](#), [69](#), [93](#), [98](#), [99](#), [127](#)
- VB to FB [99](#)
- VERIFY operator (ICETOOL) [124](#)

W

- web site [3](#)
- work storage data sets

- work storage data sets (*continued*)
 - devices for efficient use [117](#)
 - number needed [16](#)

Z

- ZD format [4](#)
- zeros [39](#)
- zoned decimal data [4](#)



Product Number: 5655-ZOS

SC23-6880-70

