

PDP-11 SORT/MERGE User's Guide

Order Number: AA-CI67B-TC

August 1990

This manual describes the use of the PDP-11 SORT/MERGE Utility to reorder and combine files on RSTS/E, RSX-11M, RSX-11M-PLUS, and Micro/RSX.

Revision/Update Information: This revised manual supersedes the *PDP-11 SORT/MERGE User's Guide*, Version 3.0. (Order No. AA-CI67A-TC)

Operating System and Version: Micro/RSX 4.2 or a higher version
RSTS/E 10.0 or a higher version
RSX-11M 4.5 or a higher version
RSX-11M-PLUS 4.3 or a higher version

Software Version: PDP-11 SORT/MERGE Version 3.1

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation.

Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

Any software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license. No responsibility is assumed for the use or reliability of software or equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.


Restricted Rights: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

© Digital Equipment Corporation 1984, 1990.

All rights reserved.
Printed in U.S.A.

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

ALL-IN-1	EduSystem	RT
DEC	IAS	ULTRIX
DEC/CMS	MASSBUS	UNIBUS
DEC/MMS	PDP	VAX
DECnet	PDT	VAXcluster
DECmate	P/OS	VMS
DECsystem-10	Professional	VT
DECSYSTEM-20	Q-bus	Work Processor
DECUS	Rainbow	
DECwriter	RSTS	
DIBOL	RSX	

Contents

Preface	vii
---------------	-----

Chapter 1 Getting Started

1.1	Sorting Records	1-1
1.1.1	The SORT Command	1-2
1.1.1.1	Input and Output Files	1-2
1.1.1.2	Qualifiers and Switches	1-3
1.2	Identifying a Key Field for SORT/MERGE	1-3
1.3	Merging Records	1-5
1.3.1	The MERGE Command	1-6
1.3.1.1	Input and Output Files	1-6
1.3.1.2	Qualifiers and Switches	1-7
1.4	Invoking the SORT/MERGE Utility from the MCR Command Line Interpreter ..	1-7
1.5	Default File-Name Extensions	1-7
1.6	Continuing a Long Command Line to a Second Line	1-8
1.7	Running SORT and MERGE Batch Jobs	1-8

Chapter 2 Defining SORT and MERGE Operations

2.1	SORT and MERGE Commands	2-1
	SORT	2-2
	MERGE	2-8
2.2	Specifying Key-Field Attributes	2-14
2.2.1	Order	2-14
2.2.2	Type of Data	2-15
2.2.2.1	Size Limitations of Key Fields According to Data Type	2-16
2.2.2.1.1	Obtaining Size Information for Key Fields	2-17
2.2.2.2	Representing Data Other Than Character or ASCII in Input Files	2-17
2.2.3	Collating Sequence	2-17
2.2.4	Specifying Key-Field Information in MCR	2-18
2.3	Using Multiple Key Fields	2-18
2.4	Dealing with Equal Key Fields	2-19

2.5	The Sorting Process	2-19
2.6	Specifying File Attributes	2-22
2.6.1	Input File Attributes	2-22
2.6.1.1	File Organization	2-22
2.6.1.2	Record Format and Longest Record Length	2-22
2.6.1.3	File Size	2-23
2.6.1.4	File Shareability	2-23
2.6.2	Output File Attributes	2-24
2.6.2.1	File Organization	2-24
2.6.2.2	Record Format and Longest Record Length	2-24
2.6.2.3	File Size	2-25
2.7	Chaining to a SORT or MERGE image (RSTS/E only)	2-25
2.8	Merging Files	2-26
2.9	Optimizing the SORT/MERGE Work Area	2-27

Chapter 3 Using a Specification File

3.1	Creating a Specification File	3-1
3.2	Processing a Specification File	3-2
3.3	Specification File Qualifiers	3-3
3.4	Identifying Record Fields	3-4
3.4.1	Specifying Key Fields	3-6
3.4.2	Formatting Data for the Output File	3-6
3.4.3	Defining and Using Conditions	3-7
3.4.3.1	Changing the Contents of a Field	3-8
3.4.3.2	Specifying Records for Inclusion or Omission	3-9
3.4.3.3	Representing Data Other Than Character or ASCII in Conditions	3-9
3.5	Sorting Files with More than One Record Format	3-10
3.6	Specifying a Collating Sequence	3-11
3.6.1	Defining Your Own Collating Sequence	3-11
3.6.2	FOLD and TIE_BREAK Subqualifiers	3-12
3.6.3	Modifying the Collating Sequence	3-13
3.6.4	Example of a User-Defined Collating Sequence	3-15
3.7	Reassigning Work Files	3-15
3.8	Specifying a New Pad Character	3-16
3.9	Format of Qualifiers in a Specification File	3-16
3.10	Sample Specification File	3-18

Chapter 4	Using SORT and MERGE in a Program	
4.1	Language Support	4-1
4.2	Accessing Callable SORT and MERGE	4-2
4.3	Specifying Your Own Routines	4-3
4.4	Calling the SORT Subroutines	4-3
4.4.1	Using the File Interface	4-4
4.4.2	Using the Record Interface	4-5
4.4.3	Using Mixed-Mode Interface	4-5
4.4.4	Passing File Names and Initializing the Sort Process	4-5
4.4.5	Passing Records to SORT	4-11
4.4.6	Returning Records to Your Program	4-11
4.4.7	Sorting Records	4-12
4.4.8	Ending a Sort Operation	4-13
4.5	Calling the MERGE Subroutines	4-13
4.5.1	File Interface	4-14
4.5.2	Record Interface	4-15
4.5.3	Mixed-Mode Interface	4-15
4.5.4	Initializing the Merge Process	4-15
4.5.5	Summary of SORT Subroutine Calls	4-18
4.6	Task Building	4-21
4.6.1	Overlay Descriptor Language Files	4-21
4.6.2	Task Building with User-Defined Routines	4-23
4.6.3	Usage of Logical Unit Numbers	4-25

Chapter 5	Customizing SORT	
5.1	SORT and MERGE Internal Operation	5-1
5.1.1	Initialization Phase	5-1
5.1.2	Sort Phase	5-2
5.1.3	Merge Phase	5-3
5.1.4	Cleanup Phase	5-3
5.2	Understanding and Using SORT/MERGE Statistics	5-4
5.2.1	Using Statistics with Callable SORT/MERGE	5-6
5.3	Modifications the User Can Make	5-7
5.3.1	Work Files	5-7
5.3.2	Input File Allocation	5-8
5.3.3	Output File Preallocation	5-8
5.3.4	Process	5-9
5.4	Modifications the System Manager Can Make	5-10

Appendix A Error Messages

Appendix B Sample Programs

B.1	BASIC-PLUS-2 Program Using the MERGE File Interface	B-2
B.2	BASIC-PLUS-2 Program Using Both SORT and MERGE Mixed-Mode Interfaces	B-3
B.3	COBOL-81 Program Using the MERGE Record Interface	B-7
B.4	COBOL-81 Program Using the SORT Record interface	B-9
B.5	FORTRAN Program Using the MERGE File Interface	B-12
B.6	FORTRAN Program Using the SORT File Interface	B-14

Appendix C DIGITAL Multinational and ASCII Collating Sequences

Index

Tables

2-1	SORT Processes	2-20
4-1	SORT Subroutines	4-4
4-2	Parameters for SRTINI, SRTINB, and SRTINC	4-5
4-3	Parameters for SRTRLS, SRTRLB, and SRTRLC	4-11
4-4	Parameters for SRTRTN, SRTRTB, and SRTRTC	4-12
4-5	MERGE Subroutines	4-14
4-6	Parameters for MRGINI, MRGINB, and MRGINC	4-16
4-7	Summary of SORT Subroutine Calls for the File Interface	4-18
4-8	Summary of SORT Subroutine Calls for the Record Interface	4-19
4-9	Summary of SORT Subroutine Calls for File-Interface Input and Record-Interface Output	4-20
4-10	Summary of SORT Subroutine Calls for Record-Interface Input and File-Interface Output	4-21
4-11	SORT/MERGE ODL Files	4-22
A-1	SORT/MERGE Utility Error Messages	A-1
C-1	DIGITAL Multinational Collating Sequence	C-1
C-2	ASCII Collating Sequence	C-7

Preface

This manual describes the use of PDP-11 SORT/MERGE to reorder and combine files on the following systems: RSTS/E, RSX-11M, RSX-11M-PLUS, and Micro/RSX.

Intended Audience

This manual is intended for all users of PDP-11 SORT/MERGE.

Conventions

The following conventions are used in this manual:

Conventions	Meaning
<code>CTRLx</code>	The symbol <code>CTRLx</code> indicates that you hold down the key labeled CTRL while simultaneously pressing the specified letter key; for example, <code>CTRL/C</code> , <code>CTRL/O</code> .
<code>RETURN</code>	A symbol with an abbreviation indicates that you must press a key on the terminal; for example, <code>RETURN</code> and <code>TAB</code> indicate that you press the RETURN key and the TAB key on your terminal.
[]	Brackets usually indicate optional syntax. Brackets that are part of directory names, however, do not indicate optional syntax.
{ }	Braces indicate that you may select only one of several choices.
.	A vertical ellipsis indicates that you can include additional syntax between the listed elements.
()	Parentheses indicate that you must enclose the choices that you select in parentheses.
Color	Color is used to show user input.

PDP-11 SORT/MERGE is a utility that accepts as input up to 10 RMS-11 formatted files and produces as output one reordered RMS-11 formatted file. Records may be sequenced in ascending or descending order by as many as 16 key fields, with a total key-field size of 512 bytes. (The key field is the field of a record on which SORT/MERGE performs an operation.) The utility supports all PDP-11 RMS files.

SORT reorders data, and MERGE combines data. For example, with SORT you can arrange a computer file of employee records by employee name or by identification number. With MERGE you can combine two or more files into a single file.

This chapter shows the syntax required for simple sort and merge operations. It includes formats for both the DIGITAL Command Language (DCL) and Monitor Console Routine (MCR) command line interpreters.

1.1 Sorting Records

PDP-11 SORT reads records from as many as 10 input files, sorts them according to the field or fields you specify, and generates one reordered output file. Suppose a file named DATA.DAT contains the data records of a magazine subscription list. One record exists for each subscriber, and each record has 5 fields, as follows:

- Name
- Street
- City
- State
- Expiration date of the subscription

Name	Street	City	State	Exp Date
Yellen Mark	90 Lynwood Lane	Westfield	MA	901231
Germont Alfredo	15 Town House Dr	Waltham	MA	910501
Thompson Lynda	395 N Main St	Easton	MA	931130
Fallon Curtis	56 Juniper Lane	Lenox	MA	941101
Tosca Floria	108 Winfield Dr	Rome	NY	920630
Weaver Stephen	72 Newton Ave	Hyde Park	NY	940509
Marsh Beverly	305 Cambridge St	Pittsfield	MA	901015
Ling Kemlo	81 River St	Belmont	NY	941031

To create a new file with the subscription list ordered alphabetically by subscriber name, use the SORT command as follows:

```
DCL> SORT DATA.DAT NAME.DAT
MCR> SRT NAME.DAT=DATA.DAT
```

The output file appears as follows:

Name	Street	City	State	Exp Date
Fallon Curtis	56 Juniper Lane	Lenox	MA	941101
Germont Alfredo	15 Town House Dr	Waltham	MA	910501
Ling Kemlo	81 River St	Belmont	NY	941031
Marsh Beverly	305 Cambridge St	Pittsfield	MA	901015
Thompson Lynda	395 N Main St	Easton	MA	931130
Tosca Floria	108 Winfield Dr	Rome	NY	920630
Weaver Stephen	72 Newton Ave	Hyde Park	NY	940509
Yellen Mark	90 Lynwood Lane	Westfield	MA	901231

The input file is DATA.DAT, and the output file is NAME.DAT. The entire file is sorted by default in alphabetical order (A to Z), which is the default. The data in DATA.DAT is arranged in fields separated by space, such as *Yellen Mark*, *90 Lynwood Lane*, and *Westfield*. The field according to which the data is sorted, in this example the subscriber name, is the key field.

1.1.1 The SORT Command

The SORT command invokes the SORT Utility. It requires an input file name and an output file name. The function of the command is modified by qualifiers and subqualifiers that describe the key field, specify options, or define other aspects of the sort operation or the input and output files. The format of the command line is as follows:

DCL:

```
SORT [/qualifier:n/qualifier=(subqualifier, subqualifier:n)] input-file[/qualifier=(subqualifier)]
output-file[/qualifier=(subqualifier)]
```

MCR:

```
SRT [output-file[/switch:n/switch:subswitch:n]= input-file[/switch:n])
```

1.1.1.1 Input and Output Files

The input file is the file to be sorted. SORT processes up to 10 input files and places the sorted data in one output file in a single operation. If you have more than 10 files to sort and you want to have the output data in a single file, you can sort 10 files at a time and then use the MERGE command to create a single file from the output files. If you do not supply an input file name, SORT prompts you for it.

The output file is the file that SORT creates. It contains the sorted data from the input file or files. You can specify only one name for the output file. If you do not supply an output file name, SORT prompts you for it.

When you have more than one input file for a sort operation, use a comma to separate the individual input file specifications. For example, if you have three input files named SALES1.DAT, SALES2.DAT, and SALES3.DAT that you want to sort into a single output file named SALES.DAT, use the following command line:

```
DCL> SORT SALES1.DAT, SALES2.DAT, SALES3.DAT SALES.DAT
MCR> SRT SALES.DAT=SALES1.DAT, SALES2.DAT, SALES3.DAT
```

1.1.1.2 Qualifiers and Switches

DCL qualifiers and MCR switches following the command or file name allow you to modify the sort operation, for example, by identifying key fields or specifying the format of the input or output files. A slash (/) is the first character in the name of every qualifier and switch. Both SORT and MERGE provide default values for qualifiers. You need only use a qualifier when you want to override the default value.

Subqualifiers, subswitches, or values following a qualifier or switch modify the action of the qualifier or switch.

An equal sign (=) separates the DCL qualifier from the subqualifier and value, as in /PROCESS=TAG, and /BUCKET_SIZE=10. If you use more than one subqualifier, enclose them in parentheses and separate them by commas, as in /KEY=(POSITION:10,SIZE:5). When a subqualifier takes a value, as in /KEY=POSITION:10, separate the subqualifier from its value with a colon (:). You can use an equal sign instead of the colon between a subqualifier and its value; however, it is good practice to use a colon in order to distinguish between qualifiers and subqualifiers. The negative form of a qualifier consists of the letters *NO* before the qualifier, as in /NOSTABLE.

MCR switches are two letters in length. A colon separates the switch from both the subswitch and value, as in /CS:ASCII and /BU:10. A colon also separates the subswitch and value, as in /FO:VARIABLE:71. The negative form of a switch includes a minus sign between the slash and the two-letter mnemonic (for example, /-ST).

Chapter 2 lists and describes the SORT qualifiers and switches. Chapter 5 describes the qualifiers that you can use to customize SORT for your own environment.

1.2 Identifying a Key Field for SORT/MERGE

On the magazine subscription list, we performed a sort operation that reordered the file according to the field that occupies the first position in the record, the default key field. Suppose, however, that you want to sort the magazine subscription list by the expiration date. To do so, you must identify the key field. This section describes how to identify key fields other than the default for the SORT Utility. The procedure is the same for the MERGE utility, except that for the MERGE Utility the default key field is the entire record.

Following is the DATA.DAT file again, this time with the fields numbered for purposes of explanation.

Name (1-19)	Street (20-39)	City (40-51)	State (52-59)	Exp Date (60-65)
Yellen Mark	90 Lynwood Lane	Westfield	MA	941231
Germont Alfredo	15 Town House Dr	Waltham	MA	910501
Thompson Lynda	395 N Main St	Easton	MA	931130
Fallon Curtis	56 Juniper Lane	Lenox	MA	941101
Tosca Floria	108 Winfield Dr	Rome	NY	920630
Weaver Stephen	72 Newton Ave	Hyde Park	NY	940509
Marsh Beverly	305 Cambridge St	Pittsfield	MA	901015
Ling Kemlo	81 River St	Belmont	NY	941031

The numbers in parentheses indicate the position of each field in the record. For example, the Name field begins at position 1 and ends at position 19. Its size is 19: each position holds one character, and one character, including the space character, is equivalent to 1 byte. (Note that this equivalent relationship between characters and bytes is not true for non-ASCII numerical data, which is discussed

in Chapter 2. If you use the TAB character in a file, it is treated as a single ASCII character.

The Expiration Date field begins at position 60 and its size is 6. You identify the Expiration Date field as the key field by means of the DCL /KEY qualifier or the MCR /KE switch. In DCL you also use two subqualifiers to the /KEY qualifier, POSITION:n and SIZE:n, which indicate the position and size of the field, as follows:

```
DCL> SORT/KEY=(POSITION:60,SIZE:6) DATA.DAT EXPDAT.DAT
```

The POSITION:60 subqualifier identifies the position of the first character in the key field, and the SIZE subqualifier (SIZE:6) indicates the number of characters in the key field.

In MCR you indicate the position and size of the key field by adding values to the /KE qualifier. The position value comes after the colon and is separated from the size value by a decimal point (.), as follows:

```
MCR> SRT EXPDAT.DAT=DATA.DAT/KE:60.6
```

Following is the EXPDAT.DAT file with the data arranged by expiration date.

Name (1-19)	Street (20-39)	City (40-51)	State (52-59)	Exp Date (60-65)
Marsh Beverly	305 Cambridge St	Pittsfield	MA	901015
Germont Alfredo	15 Town House Dr	Waltham	MA	910501
Tosca Floria	108 Winfield Dr	Rome	NY	920630
Thompson Lynda	395 N Main St	Easton	MA	931130
Weaver Stephen	72 Newton Ave	Hyde Park	NY	940509
Ling Kemlo	81 River St	Belmont	NY	941031
Fallon Curtis	56 Juniper Lane	Lenox	MA	941101
Yellen Mark	90 Lynwood Lane	Westfield	MA	941231

The expiration dates are now in ascending order, beginning with the expiration date that is earliest (has the lowest number) 901015, and ending with the date that is latest (has the highest number) 941231. Note that you cannot use the SIZE subqualifier with the DCL D_FLOATING and F_FLOATING data types (discussed in Chapter 2), because these types of data have implicit sizes.

A primary key field is the first field by which records are sorted. However, if some of your records have identical data in the primary key field, you may want to specify a secondary key field. For example, suppose you want to arrange the information in DATA.DAT by state. You choose the state field as the primary key field, but five records have MA as the state and three have NY as the state. To solve this problem, you can sort the records first alphabetically by state and then alphabetically by name. The name field is the secondary key field. To specify a secondary key field in DCL, use the /KEY qualifier twice. The first /KEY qualifier specifies the position and size of the primary key field, and the second /KEY qualifier specifies the position and size of the secondary key field, as follows:

To specify a secondary key field in MCR, you use a single /KE qualifier and add a second colon and values after the position and size values specified for the first key field.

```
DCL> SORT/KEY=(POSITION:52,SIZE:8)/KEY=(POSITION:1,SIZE:19) DATA.DAT EXPDAT.DAT  
MCR> SRT EXPDAT.DAT=DATA.DAT/KE:52.8:1.19
```

Following is the EXPDAT.DAT file with the names listed alphabetically by Massachusetts subscribers, then alphabetically by New York subscribers.

Name	Street	City	State	Exp Date
Fallon Curtis	56 Juniper Lane	Lenox	MA	941101
Germont Alfredo	15 Town House Dr	Waltham	MA	910501
Marsh Beverly	305 Cambridge St	Pittsfield	MA	901015
Thompson Lynda	395 N Main St	Easton	MA	931130
Yellen Mark	90 Lynwood Lane	Westfield	MA	941231
Ling Kemlo	81 River St	Belmont	NY	941031
Tosca Floria	108 Winfield Dr	Rome	NY	920630
Weaver Stephen	72 Newton Ave	Hyde Park	NY	940509

1.3 Merging Records

The MERGE Utility combines files that have previously been sorted by the same field or fields specified in the merge operation. In addition, MERGE can determine whether or not a file has been sorted according to a specified set of keys. The MERGE Utility accepts up to 10 input files and combines them according to the fields specified. Like SORT, MERGE generates a single output file.

The following files, BILL1.DAT and BILL2.DAT, which have been sorted alphabetically by name, are merged to create one output file, MAIL.DAT, which contains all the records from both files. The following MERGE Utility commands, containing the input and output file specifications, create the output file MAIL.DAT:

```
DCL> MERGE BILL1.DAT,BILL2.DAT MAIL.DAT
MCR> MGE MAIL.DAT=BILL1.DAT,BILL2.DAT
```

```

BILL1.DAT
Coolidge Sue    98034
Erickson Sam   72931
McKee Michael  64388

BILL2.DAT
Brown Thomas   23581
Waters Mary    44567
Woo Lee        99807

=====>
MAIL.DAT
Brown Thomas   23581
Coolidge Sue   98034
Erickson Sam   72931
McKee Michael  64388
Waters Mary    44567
Woo Lee        99807
```

In this example, no key field is specified, because the default behavior of the MERGE command is to merge the entire record.

The following files, named PLANT1.DAT and PLANT2.DAT, contain quality-control data from two plants in a manufacturing company:

```

PLANT1.DAT
Date      Plant Part   Qty   Qty   Percent
          Code Num   Manuf Rej   Usable
(1-6)    (11) (17-20) (25-28) (33-34) (42-45)

901109    1     0275   1000   37     96.3
901109    1     7820   1200   28     97.6
901109    1     2064    800   12     98.5
901109    1     4016    950   11     98.8
901109    1     3198   1500   11     99.3

PLANT2.DAT
Date      Plant Part   Qty   Qty   Percent
          Code Num   Manuf Rej   Usable
(1-6)    (11) (17-20) (25-28) (33-34) (42-45)
```

821109	2	4016	1300	33	97.4
821109	2	0275	700	13	98.1
821109	2	2064	1800	25	98.6
821109	2	3198	1650	21	98.7
821109	2	7820	1400	14	99.0

Each of the records in the two files contains six fields (Date, Plant Code, Part Number, Quantity Manufactured, Quantity Rejected, and Percent Usable). Each of the files has been sorted by the field containing the percent usable figure; this field begins in position 42 and is 4 characters long.

To merge these files according to the Percent Usable field, use the following command line:

```
DCL> MERGE/KEY=(POSITION:41,SIZE:4) PLANT1.DAT,PLANT2.DAT REPORT.DAT
MCR> MGE REPORT.DAT=PLANT1.DAT,PLANT2.DAT/KE:42.4
```

In this merge operation, the /KEY qualifier is required because the key field is not the default key field, which in the MERGE Utility is the entire record. The POSITION and SIZE subqualifiers are required whenever you use the DCL /KEY qualifier. Likewise, the position and size values are required whenever you use the MCR /KE: qualifier.

The following is the merged file, REPORT.DAT:

Date	Plant Code	Part Num	Qty Manuf	Qty Rej	Percent Usable
821109	1	0275	1000	37	96.3
821109	2	3198	1300	33	97.4
821109	1	7820	1200	28	97.6
821109	2	2064	700	13	98.1
821109	1	2064	800	12	98.5
821109	2	7820	1800	25	98.6
821109	1	0275	1650	21	98.7
821109	2	4016	950	11	98.8
821109	1	4016	1400	14	99.0
821109	2	3198	1500	11	99.3

1.3.1 The MERGE Command

The MERGE command invokes the MERGE Utility. It requires an input file name and an output file name. The function of the command is modified by qualifiers and subqualifiers that describe the key field, specify options, or define other aspects of the sort operation or the input and output files. The format of the command line, which is identical to that for SORT, is as follows:

DCL:

```
MERGE [/qualifier:n/qualifier=(subqualifier, subqualifier:n)] input-file[/qualifier=(subqualifier)]
output-file[/qualifier=subqualifier]
```

MCR:

```
MGE [output-file=/switch/switch:n:subswitch:n] input-file[/switch:n]
```

1.3.1.1 Input and Output Files

The input file parameter identifies the file or files that you want to merge. In a single operation, MERGE allows you to merge up to 10 presorted input files into one output file. A comma must separate multiple input file names. The key field specified must be the same in each of the input files. If you do not supply an input file name, MERGE prompts you for it.

The output file parameter identifies the name of the file that MERGE creates; this file contains the data from the input file or files. You can identify various characteristics of the output file by using qualifiers, which are described in Chapter 2. You can specify only one output file; if you do not supply an output file name, MERGE prompts you for it.

1.3.1.2 Qualifiers and Switches

DCL qualifiers and MCR switches following the command or file name allow you to modify the merge operation, for example, by identifying key fields or specifying the format of the input or output files. A slash (/) is the first character in the name of every qualifier and switch. Both SORT and MERGE provide default values for qualifiers. You need only use a qualifier when you want to override the default value.

Subqualifiers, subswitches, or values following a qualifier or switch modify the action of the qualifier or switch.

The DCL qualifier is separated from subqualifiers and values by an equal sign (=), as in /PROCESS=TAG, and /BUCKET_SIZE=10. If you use more than one subqualifier, enclose them in parentheses and separate them by commas, as in /KEY=(POSITION:10,SIZE:5). When a subqualifier takes a value, in /KEY=POSITION:10, separate the subqualifier from its value with a colon. You can use an equal sign instead of the colon between a subqualifier and its value; however, it is good practice to use a colon in order to distinguish between qualifiers and subqualifiers. The negative form of a qualifier consists of the letters *NO* before the qualifier, as in /NOSTABLE.

MCR switches are two letters in length. The MCR switch is separated from both subswitches and values by a colon, as in /CS:ASCII and /BU:10. Subswitches and values are also separated by a colon, as in /FO:VARIABLE:71. The negative form of a switch includes a minus sign between the slash and the two-letter mnemonic (for example, /-ST).

Chapter 2 lists and describes the MERGE qualifiers and switches.

1.4 Invoking the SORT/MERGE Utility from the MCR Command Line Interpreter

Using the MCR command line interpreter, you can invoke the SORT/MERGE Utility by typing the SRT or MGE commands without a file name. When you do so, the operating system displays the SRT> or MGE> prompts, respectively. You can then enter the remainder of the command line as follows:

```
SRT> output-file/switch:subswitch=input-file/switch:subswitch
```

Invoking SORT/MERGE in this way makes it unnecessary to retype the command each time you enter a new command line. You can also enter the command and file names together, as shown in the examples in this manual for display purposes.

1.5 Default File-Name Extensions

The SORT/MERGE Utility provides default file-name extensions when you do not specify a file-name extension in your command line. With the DCL command line interpreter, the default extension for SORT and MERGE input and output file names is DAT.

If you use DCL, you must provide a file name for at least one input file and for the output file. If you do not enter an input file specification in your command line, the following prompt is displayed on your terminal:

```
INPUT FILE?
```

SORT also prompts you if you do not provide an output file specification.

In **MCR** the default name for the input file in the sort or merge operation is **SRT.DAT**. If you have more than one input file, you must specify all file names. The default file name for the output file is **OUT.DAT**. If you do not specify a file-name extension, the system uses the **DAT** default for input and output files.

When you use **MCR** and enter one or more characters after the **SRT>** or **MGE>** prompt and then press **RETURN**, **SORT** or **MERGE** substitutes default values for any missing values in the command line. For example, if you enter only an equal sign (=), and then press **RETURN** at the **SRT>** prompt, **SORT** substitutes the default value of **SRT.DAT** for the input file and **OUT.DAT** for the output file.

1.6 Continuing a Long Command Line to a Second Line

When a command is too long to fit on one line on the screen, use the hyphen (-) continuation character to finish it on another line, as follows:

```
DCL> SORT/KEY=(POSITION:52,SIZE:8)/KEY=(POSITION:1,SIZE:19) -  
_DCL> EXPDAT.DAT  
MCR> SRT EXPDAT.DAT=DATA -  
_MCR> .DAT/KE=:52.8:1.19
```

When you type the hyphen and press **RETURN**, the operating system responds with a special prompt, which allows you to finish typing the command on another line.

1.7 Running SORT and MERGE Batch Jobs

You can run a sort or merge operation as a batch job on the **RSTS/E** and **RSX-11M-PLUS** operating systems. Batch processing frees your terminal for other work and is particularly useful when you are performing frequent or lengthy sort or merge operations. See the documentation for your operating system for more information about creating and submitting batch jobs.

Defining SORT and MERGE Operations

This chapter describes the order in which data is reordered in sort and merge operations, the different ways in which key-field data may be stored in records, and the method the SORT/MERGE Utility uses to process key fields. It lists the DIGITAL Command Language (DCL) qualifiers and the Monitor Console Routine (MCR) switches that are used to provide information to the utility and to modify aspects of the sort or merge operation. Examples show syntax for both the DCL and MCR command line interpreters.

2.1 SORT and MERGE Commands

This section describes the SORT and MERGE commands and their qualifiers.

SORT

SORT

The SORT command reads up to 10 input files, reorders the data, and produces 1 output file.

Format (DCL)

SORT *[/qualifier=(subqualifier:n, subqualifier) /qualifier=n]
input-file.DAT [/qualifier] output-file.DAT[/qualifier]*

Format (MCR)

SRT *output-file.DAT[/switch:subswitch:n/switch] =
input-file.DAT[/switch:n]*

DCL Qualifiers	MCR Switches
/ALLOCATION	/AL
/BUCKET_SIZE	/BU
/COLLATING_SEQUENCE	/CS
/CONTIGUOUS	/CO
/NODUPLICATES	/ND
/FORMAT	/FO, /BK, /BL, /SI
/INDEXED_SEQUENTIAL	/IN
/KEY	/KE
/LOAD_FILL	/LO
/OVERLAY	/OV
/PROCESS	/PR
/RELATIVE	/RE
/SEQUENTIAL	/SE
/SHAREABLE	/SH
/SPECIFICATION	/SF
/[NO]STABLE	/ST
/STATISTICS	/SS
/TREE_SPACE	/PT
/WORK_FILES	/FI, /DE
Not available in DCL	/CN

The following qualifier definitions for the SORT command are grouped alphabetically by Command Language Interpreter, DCL qualifier followed by MCR switch.

Qualifiers

/ALLOCATION=1–(2³²–1)

/AL:0–(2³²–1)

Specifies file size in blocks for use in optimization, described in Chapter 5. In DCL this is an output file qualifier; in MCR it is both an input and output file switch. As an input file switch, it specifies file size for the initial allocation of work files.

/BUCKET_SIZE=1–15

1–32

/BU:1–15

1–32

Specifies RMS bucket size for disk files, for use in optimization, described in Chapter 5: 1–15 is for RSTS/E systems, 1–32 for RSX systems. This is an output file qualifier (switch) in both DCL and MCR.

Not available in DCL

/CN[:n]

Identifies the file to be chained to, where *n* is a decimal number that represents the line number in the program being chained to (RSTS/E only).

/COLLATING_SEQUENCE=(ASCII)

(EBDIC)

(MULTINATIONAL)

/CS=ASCII

EBDIC

MULTINATIONAL

Specifies ordering systems for character data; ASCII is the default. In DCL this is a command qualifier, in MCR an input file switch.

/[NO]CONTIGUOUS

/CO

Specifies contiguous allocation of data for use in optimization, described in Chapter 5. In DCL this is an output file qualifier, in MCR both an input and output file switch.

/NODUPLICATES

/ND

Determines that when two or more key fields are equal, only the first that SORT/MERGE encounters is preserved. /STABLE (/ST) and /NODUPLICATES (/ND) cannot be used in the same operation. In DCL /NODUPLICATES is a command qualifier; in MCR /ND is an input file switch. /DUPLICATES is the default.

/FORMAT=([FILE_SIZE: 1–(2³²–1)])

([RECORD_SIZE: 1–32767])

([VARIABLE[:1–32767]])

([RMS_STREAM])

([STREAM])

([CONTROLLED[:1–32767]])

[,FSZ:1–255]

/FO:CONTROLLED:n

SORT

FIXED:n
RMS_STREAM:n
STREAM:n
UNKNOWN:n
VARIABLE:n
/BK:1-(2³²-1)
/BL:18-8192
/SI:n

Defines input and output file format and record format: **FILE_SIZE** is required for files not on disk or magnetic tape; **RECORD_SIZE** is required for files not on disk or magnetic tape or files whose longest record length is unavailable. **/BK** specifies file size for files not on disks or magnetic tapes. **/BL** specifies block size for nonstandard magnetic tapes. The **CONTROLLED**, **FIXED**, **RMS_STREAM**, **STREAM**, **UNKNOWN**, and **VARIABLE** subqualifiers specify record format. **CONTROLLED** is for use with controlled records. **RMS_STREAM** and **STREAM** are synonymous. The **/FORMAT** qualifier (switch) is also used in optimization, described in Chapter 5. In both DCL and MCR this is an input and output file qualifier (switch).

In MCR specifies record format and maximum record size (from 1 to 32767). **/BK** is an MCR input file switch that specifies file size for files not on disk or magnetic tape. **/BL** is an MCR output file switch that specifies block size for nonstandard magnetic tapes. **/SI**, an MCR input and output file switch, specifies the cluster size for RSTS/E or the retrieval window size for RSX-11M/M-PLUS.

/INDEXED_SEQUENTIAL=1-255
/IN:[1-255]

Defines input or output file organization, required for indexed-sequential files. The number of index keys defaults to 1 if a number is not specified. The output file must already exist and be empty when this qualifier (switch) is used as an output file qualifier; by default the output file is overlaid. This is an input and output file qualifier (switch) in both DCL and MCR.

/KEY=POSITION:1-255
SIZE:1-255
1, 2, 4, or 8
1-31

[ASCENDING]
[DESCENDING]

[ASCII_FLOATING]
[ASCII_ZONED]
[BINARY]
[SIGNED]
[UNSIGNED]
[CHARACTER]
[DECIMAL]
[SIGNED]
[UNSIGNED]
[TRAILING_SIGN]
[LEADING_SIGN]
[OVERPUNCHED_SIGN]
[SEPARATE_SIGN]

**[DIBOL_ZONED]
 [D_FLOATING]
 [F_FLOATING]
 [PACKED_DECIMAL]**

/KE=A

B

C

D

F

I

J

K

L

P

S

U

Z

N

O

1-65535

1-255

Describes key fields, including position, size, and data type of the field and the order of the sort operation (ASCENDING is the default order). A key field of character data can be 1-255 characters in length; a key field of binary data can be 1, 2, 4, or 8 bytes; and a key field of decimal data can be 1-31 bytes. Omit the SIZE subqualifier with the DCL D_FLOATING and F_FLOATING data types, because these data types have implicit sizes. See Section 2.2.2 for information on default data types.

In MCR specify key-field information in this order: data type, sort order, key position, and key-field size. The following is a key to the MCR abbreviations:

A (ASCII Floating)

B (COBOL COMP—6 word, signed binary)

C (Character)

D (Decimal, unsigned or trailing, overpunched sign)

F (Floating-point)

I (Decimal, leading separate sign)

J (Decimal, trailing separate sign)

K (Decimal, leading overpunched sign)

L (Decimal, DIBOL zoned)

P (Packed decimal)

S (Signed binary)

U (Unsigned binary)

Z (ASCII zoned)

N (Ascending)

O (Descending)

SORT

1-65535 (Position)

1-255 (Size)

In MCR **/KE** is an input file switch and in DCL a command qualifier.

/LOAD_FILL

/LO

Specifies fill factor when used with **INDEXED_SEQUENTIAL** files in optimization, described in Chapter 5. This is an output file qualifier (switch) in both DCL and MCR.

/[NO]OVERLAY

/[-]OV

Specifies that the output file is to be overlaid on, or written to, an existing empty file. In both DCL and MCR, **/OVERLAY** is an output file qualifier (switch).

/PROCESS=ADDRESS

INDEX

RECORD

TAG

/PR=A

I

R

T

Defines the sort process; choose one only. **RECORD (R)** is the default. In DCL this is a command qualifier, whereas in MCR it is an input file switch.

/RELATIVE

/RE

Requests relative organization; **/SEQUENTIAL** is the default. In both DCL and MCR, this is an output file qualifier (switch).

/SEQUENTIAL

/SE

Requests sequential file organization, which is the default. In both DCL and MCR, this is an output file qualifier (switch).

/[NO]SHAREABLE

/SH

Specifies that the input file is to be opened in a write-shareable mode. Specify for each shareable file. **/NOSHAREABLE** is the default.

/SPECIFICATION=file-specification

/SF

Identifies a specification file, described in Chapter 3. In MCR this is an input file switch and in DCL a command qualifier.

/[NO]STABLE

/[-]ST

Maintains the order of the input file when two or more key fields are equal. **/NOSTABLE (/ST)**, which is the default, causes the order to be unpredictable. In MCR this is an input file switch and in DCL a command qualifier.

/[NO]STATISTICS***/[-]SS***

Displays a statistical summary of the operation, primarily for help with optimization, described in Chapter 5. */NOSTATISTICS* (*/-SS*) is the default. In MCR this is an input file switch and in DCL a command qualifier.

/TREE_SPACE=0-100***/PT:0-100***

Specifies the percentage of available work area assigned to SORT/MERGE data structures in optimization, described in Chapter 5. This is an input file qualifier (switch) in both DCL and MCR.

/WORK_FILES=(NUMBER:0,3-10)***(DEVICE:ddnn:)******(ALLOCATION:1-(2³²-1))******([NO]CONTIGUOUS)******(SIZE:1-255)******/DE:ddnn:******/FI:0******3-10***

Specifies the number of work files for purposes of optimization, described in Chapter 5. In MCR the */DE* switch places work files on an alternate device, and the */FI* switch specifies the maximum number of work files to be used. Both MCR switches are input file switches. In DCL */WORK_FILES* is a command qualifier.

MERGE

MERGE

The **MERGE** command reads two or more previously sorted input files, combines the data, and produces one output file.

Format (DCL)

MERGE *[/qualifier=(subqualifier:n, subqualifier) /qualifier=n]*
input-file.DAT [/qualifier] output-file.DAT[/qualifier]

Format (MCR)

MGE *output-file.DAT[/switch:subswitch:n/switch] =*
input-file.DAT[/switch:n]

DCL Qualifiers	MCR Switches
/ALLOCATION	/AL
/BUCKET_SIZE	/BU
/[NO]CHECK_SEQUENCE	/[-]CH
/COLLATING_SEQUENCE	/CS
/CONTIGUOUS	/CO
/NODUPLICATES	/ND
/FORMAT	/FO, /BK, /BL, /SI
/INDEXED_SEQUENTIAL	/IN
/KEY	/KE
/LOAD_FILL	/LO
/OVERLAY	/OV
/RELATIVE	/RE
/SEQUENTIAL	/SE
/SHAREABLE	/SH
/SPECIFICATION	/SF
/[NO]STABLE	/ST
/STATISTICS	/SS
/TREE_SPACE	/PT
Not available in DCL	/CN

The following qualifier definitions for the **Sort** command are grouped according to Command Language Interpreter, DCL qualifier followed by MCR switch.

Command Qualifiers

/ALLOCATION=1-(2³²-1)

/AL:0-(2³²-1)

Specifies file size in blocks for use in optimization, described in Chapter 5. In DCL this is an output file qualifier in MCR both an input and output file switch. As an input file switch, it specifies file size for the initial allocation of work files.

/BUCKET_SIZE=1-15

1-32

/BU:1-15

1-32

Specifies RMS bucket size for disk files, for use in optimization, described in Chapter 5: 1-15 is for RSTS/E systems, 1-32 for RSX systems. This is an output file qualifier (switch) in both DCL and MCR.

Not available in DCL

/CN[:n]

Identifies the file to be chained to, where *n* is a decimal number that represents the line number in the program being chained to (RSTS/E only).

/[NO]CHECK_SEQUENCE

/[-]CH

Verifies that the input files have been sorted. The positive form is the default; the negative form causes sequence checking to be waived. In MCR this is an input file qualifier and in DCL a command qualifier.

/COLLATING_SEQUENCE=(ASCII)

(EBDIC)

(MULTINATIONAL)

/CS=ASCII

EBDIC

MULTINATIONAL

Specifies ordering systems for character data; ASCII is the default. In DCL this is a command qualifier. In MCR this is an input file switch, and in DCL a command qualifier.

/[NO]CONTIGUOUS

/CO

Specifies contiguous allocation of data; used in optimization, described in Chapter 5. In DCL this is an output file qualifier; in MCR it is both an input and output file switch.

/NODUPLICATES

/ND

Determines that when two or more key fields are equal, only the first that SORT/MERGE encounters is preserved. /STABLE (/ST) and /NODUPLICATES (/ND) cannot be used in the same operation. In DCL /NODUPLICATES is a command qualifier; in MCR /ND is an input file switch. /DUPLICATES is the default.

MERGE

```
/FORMAT=([FILE_SIZE: 1-(232-1)])  
          ([RECORD_SIZE: 1-32767])  
          ([VARIABLE[:1-32767]])  
          ([RMS_STREAM])  
          ([STREAM])  
          ([CONTROLLED[:1-32767]])  
                  [,FSZ:1-255]
```

```
/FO:CONTROLLED:n  
      FIXED:n  
      RMS_STREAM:n  
      STREAM:n  
      UNKNOWN:n  
      VARIABLE:n
```

```
/BK:1-(232-1)
```

```
/BL:18-8192
```

```
/SI:n
```

Defines input and output file format and record format: **FILE_SIZE** is required for files not on disk or magnetic tape; **RECORD_SIZE** is required for files not on disk or magnetic tape or files whose longest record length is unavailable. **/BK** specifies file size for files not on disks or magnetic tapes. **/BL** specifies block size for nonstandard magnetic tapes. The **CONTROLLED**, **FIXED**, **RMS_STREAM**, **STREAM**, **UNKNOWN**, and **VARIABLE** subqualifiers specify record format. **CONTROLLED** is for use with controlled records. **RMS_STREAM** and **STREAM** are synonymous. The **/FORMAT** qualifier (switch) is also used in optimization, described in Chapter 5.) In both DCL and MCR this is an input and output file qualifier (switch).

In MCR specifies record format and maximum record size (from 1 to 32767.) **/BK** is an MCR input file switch that specifies file size for files not on disk or magnetic tape. **/BL** is an MCR output file switch that specifies block size for nonstandard magnetic tapes. **/SI**, an MCR input and output file switch, specifies the cluster size for RSTS/E or the retrieval window size for RSX-11M/M-PLUS.

```
/INDEXED_SEQUENTIAL=1-255
```

```
/IN:[1-255]
```

Defines input or output file organization; required for indexed-sequential files. The number of index keys defaults to 1 if a number is not specified. The output file must already exist and be empty when this qualifier (switch) is used as an output file qualifier; by default the output file is overlaid. This is an input and output file qualifier (switch) in both DCL and MCR.

```
/KEY=POSITION:1-255
```

```
      SIZE:1-255
```

```
          1, 2, 4, or 8
```

```
          1-31
```

```
      [ASCENDING]
```

```
      [DESCENDING]
```

```
      [ASCII_FLOATING]
```

```
      [ASCII_ZONED]
```

```
      [BINARY]
```

```
          [SIGNED]
```

```
          [UNSIGNED]
```

[CHARACTER]
[DECIMAL]
[SIGNED]
[UNSIGNED]
[TRAILING_SIGN]
[LEADING_SIGN]
[OVERPUNCHED_SIGN]
[SEPARATE_SIGN]
[DIBOL_ZONED]
[D_FLOATING]
[F_FLOATING]
[PACKED_DECIMAL]

/KE=A

B
C
D
F
I
J
K
L
P
S
U
Z

N
O

1-65535

1-255

Describes key fields, including position, size, and data type of the field and the order of the sort operation (ASCENDING is the default order). A key field of character data can be 1-255 characters in length; a key field of binary data can be 1, 2, 4, or 8 bytes; and a key field of decimal data can be 1-31 bytes. Omit the SIZE subqualifier with the DCL D_FLOATING and F_FLOATING data types, because these data types have implicit sizes. See Section 2.2.2 for information on default data types.

In MCR specify key-field information in this order: data type, sort order, key position, and key-field size. The following is a key to the MCR abbreviations:

A (ASCII Floating)
 B (COBOL COMP—6 word, signed binary)
 C (Character)
 D (Decimal, unsigned or trailing, overpunched sign)
 F (Floating-point)
 I (Decimal, leading separate sign)
 J (Decimal, trailing separate sign)
 K (Decimal, leading overpunched sign)
 L (Decimal, DIBOL zoned)
 P (Packed decimal)
 S (Signed binary)

MERGE

U (Unsigned binary)
Z (ASCII zoned)

N (Ascending)
O (Descending)

1-65535 (Position)
1-255 (Size)

In MCR **/KE** is an input file switch and in DCL a command qualifier. See Section 2.2.2 for information on default data types.

/LOAD_FILL

/LO

Specifies fill factor when used with **INDEXED_SEQUENTIAL** files in optimization, described in Chapter 5. This is an output file qualifier (switch) in both DCL and MCR.

/[NO]OVERLAY

/[-]OV

Specifies that the output file is to be overlaid on, or written to, an existing empty file. In both DCL and MCR, **/OVERLAY** is an output file qualifier (switch).

/RELATIVE

/RE

Requests relative organization; **/SEQUENTIAL** is the default. In both DCL and MCR, this is an output file qualifier (switch).

/SEQUENTIAL

/SE

Requests sequential file organization, which is the default. In both DCL and MCR, this is an output file qualifier (switch).

/[NO]SHAREABLE

/SH

Specifies that the input file is to be opened in a write-shareable mode. Specify for each shareable file. **/NOSHAREABLE** is the default.

/SPECIFICATION=file-specification

/SF

Identifies a specification file, described in Chapter 3. In MCR this is an input file switch and in DCL a command qualifier.

/[NO]STABLE

/[-]ST

Maintains the order of the input file when two or more key fields are equal. **/NOSTABLE** (**/-ST**), which is the default, causes the order to be unpredictable. In MCR this is an input file switch and in DCL a command qualifier.

/[NO]STATISTICS

/[-]SS

Displays a statistical summary of the operation, primarily for help with optimization, described in Chapter 5. **/NOSTATISTICS** (**/-SS**) is the default. In MCR this is an input file switch and in DCL a command qualifier.

MERGE

/TREE_SPACE=0-100

/PT:0-100

Specifies the percentage of available work area assigned to SORT/MERGE data structures in optimization, described in Chapter 5. This is an input file qualifier (switch) in both DCL and MCR.

2.2 Specifying Key-Field Attributes

In addition to size and position, discussed in Chapter 1, you can also specify other attributes of the key field, such as the order in which it is sorted or merged and the type of data it contains.

2.2.1 Order

Following is a file named SALES.DAT, which contains quarterly sales totals for individual salespersons in thousands of dollars:

Store (1-2)	Dept (6-7)	Name (12-27)	Q1 (34-36)	Q2 (42-44)	Q3 (50-52)	Q4 (58-60)	Total (66-69)
1E	B1	Emery Patrick	6.5	6.2	5.9	6.7	25.3
2E	B1	Applebaum George	6.9	7.3	6.4	6.8	27.4
1E	B1	Kilpatrick Karyn	6.3	5.8	6.7	6.2	25.0
1E	B1	Hoffman Cheryl	6.8	6.4	6.9	7.0	27.1
2E	A1	Sterling Martha	8.3	7.9	7.8	8.1	32.1
1E	A1	Griffen Michael	7.5	7.3	7.4	7.6	29.8
2E	B1	Fenster Barbara	6.7	6.4	6.6	6.5	26.2
2E	A1	Gates Stephen	7.1	7.0	6.9	7.1	28.1
2E	A1	Ling Kemlo	6.7	6.6	6.8	6.7	26.8
1E	A1	Albertson Ronald	6.9	6.7	6.8	6.8	27.2

The positions of the fields are indicated in parentheses. To sort the figures in the Total field in descending order, highest total sales to lowest, enter one of the following commands:

```
DCL> SORT/KEY=(POSITION:66,SIZE:4,DESCENDING) SALES.DAT DOLLAR.DAT
MCR> SRT DOLLAR.DAT=SALES.DAT/KE:O66.4
```

The key field is the Total field, which is in position 66 and is 4 characters long. The DCL DESCENDING subqualifier to the /KEY qualifier is added after the POSITION and SIZE subqualifiers and preceded by a comma. To sort this file in ascending order, lowest to highest total sales, you would not need to specify the ASCENDING subqualifier, because it is the default. Alphabetical (A to Z) order and lowest to highest number are ascending order. Sequence is determined by the number that a character or digit equates to in the collating sequence you are using. For example, uppercase A is equivalent to 65 in the DIGITAL Multinational Collating Sequence, and uppercase Z is equivalent to 90. Appendix C lists the DIGITAL Multinational Collating Sequence and the ASCII Collating Sequence.

The MCR letter *O* in *KE:O66.4*, represents *opposite* and specifies that the data is to be sorted in *descending* order. To sort the data in ascending order, you can either omit the letter designation (since ascending order is the default) or include *N* (for *normal*) after the colon. You must indicate the sort order before you list the numbers indicating position and size.

Following is the output file, DOLLAR.DAT, with the total sales figures in descending order.

2E	A1	Sterling Martha	8.3	7.9	7.8	8.1	32.1
1E	A1	Griffen Michael	7.5	7.3	7.4	7.6	29.8
2E	A1	Gates Stephen	7.1	7.0	6.9	7.1	28.1
2E	B1	Applebaum George	6.9	7.3	6.4	6.8	27.4
1E	B1	Hoffman Cheryl	6.8	6.4	6.9	7.0	27.1
1E	A1	Albertson Ronald	6.9	6.7	6.8	6.8	27.2
2E	A1	Ling Kemlo	6.7	6.6	6.8	6.7	26.8
2E	B1	Fenster Barbara	6.7	6.4	6.6	6.5	26.2
1E	B1	Emery Patrick	6.5	6.2	5.9	6.7	25.3
1E	B1	Kilpatrick Karyn	6.3	5.8	6.7	6.2	25.0

Specify a descending sort order when you have numeric data that you want to order from highest to lowest, or when you have alphabetic character data that you want in reverse alphabetic order.

2.2.2 Type of Data

In the examples thus far, we have sorted and merged only text, or character data. The SORT/MERGE Utility accepts 17 types of data, which have the following syntax in DCL (the default data types are in boldface):

```

CHARACTER
ASCII_FLOATING
ASCII_ZONED
BINARY
SIGNED
UNSIGNED
DECIMAL
SIGNED
UNSIGNED
TRAILING_SIGN
LEADING_SIGN
OVERPUNCHED_SIGN
SEPARATE_SIGN
DIBOL_ZONED
DECIMAL
TRAILING
OVERPUNCHED_SIGN
D_FLOATING
F_FLOATING
PACKED_DECIMAL

```

The MCR syntax for data types is listed separately in this section. If the data in your key field is of any type but one of these defaults, it is necessary to identify the data type as a subqualifier to the /KEY qualifier (/KE switch) in the command line, as follows:

```

DCL> SORT/KEY=(POS:66,SIZ:4,DESC,DECIMAL,UNSIGNED) -[RETURN]
_DCL> SALES.DAT DOLLAR.DAT

```

This command specifies a descending sort operation on the field with the total sales figures in SALES.DAT, which is a numeric field and contains unsigned decimal data. As this example shows, you can abbreviate qualifiers and sub-qualifiers as long as the abbreviations are unique. You can also abbreviate MCR subswitches.

If the data in your key field is binary and signed, you need specify only BINARY, since BINARY,SIGNED is the default. However, if your key data type is binary and unsigned, you must specify BINARY,UNSIGNED.

If key-field data is decimal and its sign is trailing and overpunched (the default), you may specify `DECIMAL` only. If your key-field data is decimal but unsigned, you must specify `DECIMAL,UNSIGNED`. Note that you must use a comma between the data type and its optional arguments.

If your key-field data is decimal and its sign is trailing but separate, specify `TRAILING_SIGN` and `SEPARATE_SIGN`. If your data is decimal and its sign is leading and overpunched, specify `LEADING_SIGN` and `OVERPUNCHED_SIGN`. If your data is decimal and its sign is leading and separate, specify `LEADING_SIGN` and `SEPARATE_SIGN`.

In MCR the names of the data types are abbreviated, as follows:

- A (ASCII floating string)
- B (Signed two's complement binary, COBOL COMP-6)
- C (Character, ASCII, EBCDIC, or MULTINATIONAL)
- D (Decimal, unsigned, trailing overpunched sign)
- F (Floating point)
- I (Decimal, leading separate sign)
- J (Decimal, trailing separate sign)
- K (Decimal, leading overpunched sign)
- L (DIBOL zoned decimal, trailing overpunched sign)
- P (Packed decimal)
- S (Signed binary, COBOL COMP or FORTRAN integer)
- U (Unsigned binary)
- Z (ASCII zoned)

To specify a decimal, unsigned data type for a descending sort operation on the field with the total sales figures in the `SALES.DAT` file, use the following MCR command line:

```
MCR> SRT DOLLAR.DAT=SALES.DAT, KE:DO66.4
```

Note that the MCR data type is specified after the colon on the `/KE:` switch and before the order of the operation. Section 2.2.4 summarizes the methods of specifying key-field information in MCR.

2.2.2.1 Size Limitations of Key Fields According to Data Type

You must always specify the size of a key field that is not the default key field. The size specification of a key field, given in bytes, depends on the data type.

The following rules apply to the size of a key field according to the type of data it contains:

- With `CHARACTER` data, the size of the key field cannot exceed 255 bytes (equivalent to 255 characters for the `CHARACTER` data type).
- With `BINARY` data, you must specify the size of the key field as 1, 2, 4, or 8 bytes.
- With `DECIMAL` data, the maximum size of the key field is 31 digits; if a decimal number has a plus sign (+) or minus sign (-), the sign must be counted in the total number of digits.
- With floating-point data, the size of the key field must be either 4 or 8 bytes except that with the `ASCII_FLOATING`, `D_FLOATING`, and `F_FLOATING` data types, the size of the key field is implicit and cannot be specified.

2.2.2.1.1 Obtaining Size Information for Key Fields

You can determine the size of key fields of CHARACTER and ASCII data by counting the characters in the field. This is so because a single character or ASCII digit is stored in a single byte. Numerical data other than ASCII is not stored in the same manner and is not collated by SORT/MERGE as ASCII data is collated. For example, three characters of PACKED-DECIMAL data are compressed into the space normally occupied by two ASCII characters.

To obtain the key-field size that you supply to SORT, consult the Data Division map that is listed in the map file generated when you compile your program.

2.2.2.2 Representing Data Other Than Character or ASCII in Input Files

If the data in the field being sorted is numerical and non-ASCII, care must be taken to ensure that the data is represented correctly. SORT/MERGE translates character and ASCII data into its machine-language representation, which is necessary for correct processing by the Central Processing Unit (CPU). The utility does not translate other types of data. Therefore, to sort other types of data, you must represent it in octal, decimal, or hexadecimal form, which can be read by the CPU. The octal representation must be preceded by %O, the decimal by %D, and the hexadecimal by %X. For further information on the use of other types of data, consult the user's guide to your programming language.

2.2.3 Collating Sequence

Depending on the type of data you are sorting, you may want to specify a collating sequence. By default, SORT/MERGE arranges characters in American Standard Code for Information Exchange (ASCII) sequence. The utility also allows you to use either the Extended Binary Coded Decimal Interchange Code (EBCDIC) or MULTINATIONAL collating sequence. You might use EBCDIC, for example, as input to a program that requires EBCDIC sequence; you might use MULTINATIONAL if your records use the DIGITAL Multinational Character Set. When you select EBCDIC, input files are sorted as if the EBCDIC key field were translated into ASCII and then sorted as an ASCII key field. Records do not change.

The DCL command qualifiers for specifying a collating sequence are as follows:

```
/COLLATING_SEQUENCE=(ASCII)
/COLLATING_SEQUENCE=(EBCDIC)
/COLLATING_SEQUENCE=(MULTINATIONAL)
```

The MCR input file switches for specifying a collating sequence are as follows:

```
/CS:ASCII
/CS:EBCDIC
/CS:MULTINATIONAL
```

When you use the MULTINATIONAL collating sequence, the following ordering procedures are applied.

- All diacritical forms of a character (that is, all forms of a character that include any accent mark) are given the collating value of the character; for example, *A'*, *A"*, and *A* all collate as *A*.
- Lowercase characters are given the collating value of their uppercase equivalents; for example, *a* collates as *A* and *a"* collates as *A"*.

- If two strings compare as equal, tie-breaking is performed. The strings are compared to detect differences due to diacritical marks, ignored characters, or characters that collate as equal although they are actually different. If the strings still compare as equal, another comparison is done based on the numeric codes of the characters. In this final comparison, lowercase characters are ordered before uppercase.

NOTE

Exercise care when you use the MULTINATIONAL collating sequence for records and files that will be processed later by a program. Sequence-checking procedures in most programming languages compare the numeric values that represent the individual characters. Because MULTINATIONAL is based on actual graphic characters, and not the codes representing those characters, normal sequence checking will not work.

Appendix C lists the DIGITAL Multinational Collating Sequence and the ASCII Collating Sequence).

2.2.4 Specifying Key-Field Information in MCR

The order for specifying key information with the /KE: switch in MCR is as follows:

1. **Data type** is the first element specified after the colon on the /KE switch. In the MCR example in Section 2.2.2, the data type is unsigned decimal, represented by *D* in /KE:DO66.4.
2. **Sort order** is the second element after the colon. In the same example, the order is descending (opposite), represented by *O*, in KE:DO68.4.
3. **Position of the field** is the third element after the colon and is specified as an integer represented by 66 in KE:DO66.4.
4. **Size of the field** is the fourth element after the colon and is specified as an integer separated from position by a decimal point (.), as in KE:DO66.4.

2.3 Using Multiple Key Fields

You can specify up to 16 key fields in a sorting operation, with a total key-field size of up to 512 bytes. You must choose an order of priority for multiple key fields and list them in that order in the command string: primary key followed by secondary key and so on.

For example, to arrange the sales records in SALES.DAT by store, by department, and in descending order of total sales, specify the store field as the primary key, the department field as the secondary key, and the total sales field as the tertiary key, as follows:

```
DCL> SORT/KEY=(POS:1, SIZ:2)/KEY=(POS:6, SIZ:2) - [RETURN]
_DCL> /KEY=(POS:66, SIZ:4, DESC) SALES.DAT STORES.DAT
MCR> SRT STORES.DAT=SALES.DAT/KE:1.2:6.2:O66.4
```

In DCL you must use a separate /KEY qualifier for each sort key. If SORT finds the POSITION and SIZE subqualifiers repeated after a single /KEY qualifier, it does not treat them as specifications for multiple key fields. Instead, SORT causes the subqualifiers most recently encountered to override previous subqualifiers.

In MCR you must repeat the colon and the values for each of the subsequent key fields after the /KE switch for the primary key, but you do not need to repeat the /KE switch.

After this multiple-key operation is completed, the sorted sales records appear as follows in STORES.DAT:

Store	Dept	Name	Q1	Q2	Q3	Q4	Total
1E	A1	Griffen Michael	7.5	7.3	7.4	7.6	29.8
1E	A1	Albertson Ronald	6.9	6.7	6.8	6.8	27.2
1E	B1	Hoffman Cheryl	6.8	6.4	6.9	7.0	27.1
1E	B1	Emery Patrick	6.5	6.2	5.9	6.7	25.3
1E	B1	Kilpatrick Karyn	6.3	5.8	6.7	6.2	25.0
2E	A1	Sterling Martha	8.3	7.9	7.8	8.1	32.1
2E	A1	Gates Stephen	7.1	7.0	6.9	7.1	28.1
2E	A1	Ling Kemlo	6.7	6.6	6.8	6.7	26.8
2E	B1	Applebaum George	6.9	7.3	6.4	6.8	27.4
2E	B1	Fenster Barbara	6.7	6.4	6.6	6.5	26.2

Note that the records in the output file are ordered first by store, then by department, and finally by the highest total sales figure.

2.4 Dealing with Equal Key Fields

Your input files may contain records with key fields that are equal. These records will be grouped together in the output file, and, by default, their sorted order (with reference to each other) will be unpredictable. However, you can use one of two qualifiers to modify the sort order of equal key fields: /STABLE (/ST switch) and /NODUPLICATES (/ND switch). The /STABLE qualifier causes records with equal keys to be directed to the output file in the order in which they were input to SORT/MERGE, and /NODUPLICATES causes SORT/MERGE to retain only the first of the equal records it encounters. The default is /NOSTABLE (/–ST). Note that you cannot use both the /NODUPLICATES and the /STABLE qualifiers in the same operation.

If you specify /STABLE (/ST) when sorting multiple input files, the output file will contain records with equal keys from the first file preceding those from the second file, and so on.

To specify which of the duplicate records SORT/MERGE is to keep, use SORT /MERGE in a program, write your own equal-key routine, and name it SRTCLB. Then either pass your equal-key routine address to the callable SORT/MERGE initialization program (SRTINI or MRGINI) or link the program with your equal-key routine. Chapter 4 discusses calling SORT from a program.

The qualifiers and switches that can be used to modify sort and merge operations are described at the end of this chapter.

2.5 The Sorting Process

The examples thus far have used only the record sorting process, which produces an output file containing the complete records from the input file or files. It is also possible to reorder records from one file in several ways for different purposes. SORT provides four processing methods for sorting data: record, tag, address, or index.

Record sort is usually the most appropriate choice if you want to print your output file, if your record size is not large, and if adequate temporary storage space is available. If you want to print your output file, but adequate temporary storage space is not available, tag sort is the recommended choice. If, instead of

printing them, you want to use the sorted records in a program that performs calculations, for example, then address sort would be the appropriate choice. If you want to use the sorted records in a program that needs to access key-field data, then index sort would be the preferred choice. Table 2-1 summarizes information about these processes.

Table 2-1: SORT Processes

Process	Input Device	Output Device	Description
Address	Disk only	Any device that accepts binary data	Address sorts only key fields. The output file contains only a list of pointers to the records in the input file. The list consists of 3-word record file addresses (RFAs) in binary format, and 1-word input file numbers if multiple input files are being sorted.
Index	Disk only	Any device that accepts binary data	Index sorts only key fields. The output file contains only a list of pointers to the records in the input file. The list consists of key fields, 3-word RFAs in binary format, and 1-word input file numbers if multiple input files are being sorted.
Record	Any RSTS/E or RSX-11M/M-PLUS input device	Any RSTS/E or RSX-11M/M-PLUS output device	Record keeps record intact throughout the sort operation. The output file contains complete records.
Tag	Disk only	Any RSTS/E or RSX-11M/M-PLUS input device	Tag sorts only key fields, then reaccesses the input file records to create the output file. The output file contains complete records.

To select a sort process, consider the following factors:

1. How you will use the output file:
 - Because record and tag sorts generate output files containing entire sorted records, the output files are ready for use.
 - Both address- and index-sorted output files can be processed by a program written in native-mode BASIC, MACRO, or BLISS.
 - Address sort creates a list of pointers to the records in the input file. This list consists of file addresses of binary records, plus a file number when sorting multiple input files. A program accesses the records by means of the pointers.
 - Index sort creates an output file containing both record file addresses (RFAs) and key fields, plus a file number when sorting multiple files. The format of these key fields is the same as it is in the input files. If the program needs key-field content for a decision during future processing, select index sort rather than address sort. (See the *RMS-11 User's Guide* and *RMS-11 MACRO-11 Reference Manual* for more information about RFAs.) Note that the index sort process is unrelated to the RMS indexed file organization.

If you need to reorder records from one file in several ways for different purposes, store several output files from address or index sort and use the files to access the records in the main file in the sorted order you want.

2. The temporary storage space available for the sort operation:
 - Tag sort uses less temporary storage space than record sort. Because record sort keeps the record intact during the sort operations, it uses much more work space when the files are large.
 - Address and index sort use little temporary storage space.
3. The type of input and output device used:
 - Record sort is the only process that can accept input from cards, magnetic tape, and disk.
 - Output from tag and record sorts can go to any output device; output from address and index sort must go to a device that accepts binary data.
4. Differences in speed:
 - If you plan to retrieve the sorted records at some point in the operation, record sort is usually the fastest process.
 - Because tag sort moves only key fields instead of complete records, it can be faster than record sort when record size is very large and key-field size is small. Tag sort can also be faster for extremely large files and devices with short seek times (the time required to position the record pointer to the correct record in the input file). In most cases, however, the time that tag sort takes to reaccess the input file to create the output file makes it slower than record sort.
 - Address and index sort are the fastest processes.

To specify a sort process with DCL, use one of the following command qualifiers:

```
RECORD
/PROCESS= TAG
          ADDRESS
          INDEX
```

Note that the /PROCESS qualifier is applicable to SORT only.

To specify a sort process with MCR, use one of the following input file switches:

```
R (record)
/PR: T (tag)
     A (address)
     I (index)
```

None of the examples in this chapter has specified the sort process, so, by default, each has used record sort. To use a tag sort for the descending sort operation on the SALES.DAT file, use one of the following command lines:

```
DCL> SORT/KEY=(POS:66,SIZ:4,DESC)/PROCESS=TAG SALES.DAT DOLLAR.DAT
MCR> SRT DOLLAR.DAT = SALES.DAT/KE:066.4/PR:T
```

2.6 Specifying File Attributes

Under certain circumstances, you are required to specify file attributes and under others you may choose to change file attributes. This section explains how to provide information on file attributes to SORT/MERGE.

PDP-11 SORT/MERGE accepts all PDP-11 Record Management Services (RMS) files; that is, it accepts sequential, relative, or indexed-sequential data files on one or more mass-storage devices, containing records of fixed, variable, variable with fixed-length control (VFC), or RMS stream format. You can specify up to 10 input files; the input files need not have the same record formats and file organizations.

2.6.1 Input File Attributes

This section discusses the input file attributes: file organization, longest record length, file format, and file size.

2.6.1.1 File Organization

The three types of file organization are relative, sequential, and indexed-sequential. If the organization of your input file is indexed-sequential, you must specify this organization and indicate the number of key fields in the indexed file. If you do not specify the number of key fields, the SORT/MERGE Utility uses a default of 1. It is necessary to indicate this number so that SORT/MERGE can allocate sufficient RMS space. You need not specify relative or sequential file organization, as SORT/MERGE allocates sufficient RMS space for these types of files by default.

Specify indexed-sequential organization with the input file qualifier /INDEXED_SEQUENTIAL[=*n*] (/IN[*n*] switch), where *n* equals the number of key fields, as follows:

```
DCL> SORT/KEY=(POS:10,SIZ:2)/KEY=(POS:20,SIZ:4) LIST.DAT-RETURN
_DCL> /INDEXED_SEQUENTIAL=2 STAT.DAT
MCR> SRT STAT.DAT=LIST.DAT/KE:10.2:20.4/IN:2
```

2.6.1.2 Record Format and Longest Record Length

If you are sorting files not residing on disk or standard ANSI magnetic tape (for example, if you are passing a file from a program and the file is in memory), you must specify the size of the longest record in your input files as well as the size of your files. To determine the longest record length, consult the documentation for your programming language. The record size that you specify overrides the size defined in the file header or label.

You specify the the longest record length in bytes. The longest record length allowed for the three types of file organization is as follows:

File Organization	Longest Record Length
Sequential	32,765
Relative	16,381
Indexed-sequential	16,369

These totals include control bytes for variable records with VFC format. For multiple input files, the longest record length is the length of the longest record among all of the files.

In DCL, to specify the longest record length in the input file, use the /FORMAT=(RECORD_SIZE:n) input-file qualifier.

In MCR you must also specify the record format when you specify the longest record length. Use one of the following input file switches:

```
/FO:CONTROLLED:n
  FIXED:n
  RMS_STREAM:n
  STREAM:n
  VARIABLE:n
  UNKNOWN:n
```

RMS_STREAM and STREAM are duplicates of each other: Version 3 and higher versions of PDP-11 SORT/MERGE support the STREAM syntax for compatibility with Version 2. If the longest record length cannot be obtained from RMS (that is, if the file was created by a version of RMS that does not include information on the longest record length), you must specify it. To specify the longest record length for a descending sort operation on the total sales figures in SALES.DAT, use the following DCL command:

```
DCL> SORT/KE=(POS:66,SIZ:4,DESC) SALES.DAT/FORMAT=(RECORD_S-)
_DCL> IZE:71) DOLLAR.DAT
```

The same command line in MCR is as follows:

```
MCR> SRT DOLLAR.DAT=SALES.DAT/KE:O66.4/FO:V:71
```

As this example shows, you can abbreviate MCR subswitches to one letter.

2.6.1.3 File Size

You specify file size in blocks. (To determine file size, use the DCL DIRECTORY command or MCR PIP/LI command.) SORT uses file-size information to estimate the file size of the temporary files used for the sort operation. The maximum file size accepted is 4,294,967,295 or $(2^{32}-1)$ blocks. For multiple input files, the size is the sum of the sizes of the individual files. SORT allocates 1000 blocks by default if you do not specify the file size.

In DCL specify input file size with the /FORMAT=FILE_SIZE:n input file qualifier. In MCR specify input file size with the /BK:n input-file switch.

The DCL command line for a descending sort on the sales figures in SALES.DAT with the longest record length and file-size specified is as follows:

```
DCL> SORT/KEY=(POS:66,SIZ:4,DESC) SALES.DAT/FORMAT=(RECORD_S-)
_DCL> IZE:71,FILE_SIZE:3) DOLLAR.DAT
```

The same command line in MCR is as follows:

```
MCR> SRT DOLLAR.DAT=SALES.DAT/KE:O66.4/FO:V:71/BK:3
```

2.6.1.4 File Shareability

If you want to sort files that may be updated by another user during the sort operation, then you must specify that your input files be opened in write-shareable mode. By default, the files are not shareable. For each shareable file, use the DCL input file qualifier /SHAREABLE or the MCR file switch /SH. The default is /NOSHAREABLE.

2.6.2 Output File Attributes

You can specify the file organization and record format of the output file. If you direct the output file to a magnetic tape device, you can also specify file size.

2.6.2.1 File Organization

The default file organization for the output file in a sort or merge operation is sequential. You can override this default in either of the following ways:

1. Use one of the following qualifiers or switches to specify the file organization:

- DCL output file qualifier

```
/SEQUENTIAL  
/RELATIVE  
/INDEXED_SEQUENTIAL[=n]
```

- MCR output file switch

```
/SE (sequential)  
/RE (relative)  
/IN[:n] (indexed-sequential)
```

If you specify indexed-sequential organization, an empty indexed-sequential file of the same name must already exist. SORT writes over the existing file. SORT/MERGE does not create an indexed output file if no such file exists. You can optionally specify the number of key fields in the indexed-sequential file.

2. Use the /OVERLAY output file qualifier in DCL (/OV switch in MCR). When you use the /OVERLAY qualifier, the output file must already exist and be empty, and its file organization must have been previously defined. If you use the /OVERLAY qualifier, you cannot use any of the file organization qualifiers.

In general, to write sorted records to an existing empty file, you should use the /OVERLAY qualifier.

2.6.2.2 Record Format and Longest Record Length

If you want the record format in the output file to differ from that of your first input file, you must specify the output format. You can specify fixed-length records, variable-length records, variable with fixed-length control (VFC) records, or stream records (for RMS stream files only). If you do not specify the record format for the output file, with the record or tag sort processes it defaults to the record format of the first input file, and with address or index sort it defaults to fixed-record format.

In specifying the output record format, you can optionally indicate the longest record length (in bytes) of the output records. The default maximum record length is a length long enough to hold the longest record. The longest record lengths allowed for the three types of file organization are as follows:

File Organization	Longest Record Length
Sequential	32,765
Relative	16,381
Indexed-sequential	16,369

Indicate the new record format by using one of the following qualifiers or switches, where *n* is the size of the longest record:

- DCL output-file qualifier:
 FIXED:*n*
 VARIABLE:*n*
/FORMAT= RMS_STREAM:*n*
 STREAM:*n*
 CONTROLLED:*n*,FSZ:*m*

The FSZ subqualifier is used with VFC (CONTROLLED) records. It specifies the size in bytes of the fixed portion of the record, up to a maximum of 255 bytes. If you specify this size as 0, RMS uses a default value of 2 bytes. If you do not specify FSZ, the default is the maximum size of the fixed-control portions of all VFC input files. If you do not specify FSZ and there are no VFC input files, the default is 2 bytes.

- MCR output-file switch:
 FIXED:*n*
 VARIABLE:*n*
/FO: RMS_STREAM:*n*
 STREAM:*n*
 CONTROLLED:*n*:*m*

The STREAM subswitch exists for compatibility with Version 2 and is the same as RMS_STREAM. You can specify a second value *m* for CONTROLLED records to give the size in bytes of the fixed portion of the record, up to a maximum of 255 bytes. If you specify this size as 0, RMS uses a default value of 2 bytes. If you do not specify this size, the default is the maximum size of the fixed control portions of all VFC input files. If you do not specify this size and there are no VFC input files, the default is 2 bytes.

In both DCL and MCR, you can truncate record format values to the first letter.

2.6.2.3 File Size

If you direct your output file to magnetic tape, you can specify the block size of the file in bytes or you can accept the default. If one or more of the input files is a tape file, the block size of the output file defaults to the maximum of the block sizes of all tape input files. If the input file is a disk file, the default value is 512 bytes.

In DCL specify the block size of a file with the BLOCK_SIZE:*n* subqualifier in the /FORMAT output file qualifier (for example, /FORMAT=(FIXED,BLOCK_SIZE:800)).

In MCR specify the block size of a file with the output file switch /BL:*n*.

2.7 Chaining to a SORT or MERGE Image (RSTS/E only)

If you are using the RSTS/E operating system, you can chain from one executable image to another. PDP-11 SORT/MERGE supports RSTS/E chaining from the MCR command line interpreter with the /CN output switch. Chaining is not available with the DCL command line interpreter.

For example, you can request that the SORT Utility chain to an image MYPROG.TSK generated by BASIC-PLUS-2 upon completion of the requested ordering operation. The MCR command is as follows:

```
SRT> OUT,MYPROG/CN=INP.DAT
```

The name of the chain image is listed as a second output file, but with the /CN switch appended. The switch is required. In addition, you can specify the line number of the image where execution is to begin. For example, the following command line causes the image MYPROG.TSK to begin at line 1400:

```
SRT> OUT,MYPROG/CN:1400=INP.DAT
```

Compilers or programs other than the BASIC-PLUS-2 compiler or programs may interpret the meaning of the chain value (for example, 1400 in the above example) in different ways. SORT or MERGE places the value in the FQNTENT field of the FIRQB block. See the *RSTS/E Systems Directive Manual* for details.

It is also possible to chain into SORT or MERGE from other tasks. To accomplish this, place the sort command line into the RSTS/E core common, place the value 30000 in the FQNTENT field of the FIRQB, and then chain to the desired SORT or MERGE task file. Consult the *RSTS/E System Directives Manual* and the *RSTS/E Programming Manual* for your language for details. For example, the following BASIC-PLUS-2 program chains into the sort image SRTUTL.TSK:

```
10 V$ = SYS (CHR$(8
%)+"SOR OUT=INPUT/FO:V:3/SS")
15 CHAIN "LB:[1,2]SRTUTL.TSK" LINE 30000
20 STOP
```

(To chain to MERGE, replace SRTUTL.TSK with MGEUTL.TSK.) Line 10 in the program places the command line in core common. Line 15 first causes the value 30000 to be placed into the FIRQB and then chains to the SORT image.

2.8 Merging Files

The MERGE Utility allows you to combine up to 10 presorted files. All your input files must have been already sorted by the same key fields according to which you intend to merge. You specify the same key-field information and file attributes for MERGE as for SORT. The MERGE Utility prompt is MGE>.

However, you do not specify any processes or work files with MERGE. Also, a function unique to MERGE called sequence checking (invoked with the /CHECK_SEQUENCE qualifier in DCL and /CH switch in MCR) verifies that your input files are sorted.

You can use qualifiers to indicate explicitly whether or not sequence checking should be performed. Use these qualifiers if you do not want sequence checking performed (to override the default) or if you want to ensure that sequence checking is performed (for example, to override an instruction in a specification file that cancels sequence checking).

In DCL specify whether or not sequence checking is done with the command qualifier /CHECK_SEQUENCE or /NOCHECK_SEQUENCE. The default is CHECK_SEQUENCE.

In MCR specify whether or not sequence checking is done with the input file switch /CH or /-CH. The default is /CH.

When you use the /NOCHECK_SEQUENCE qualifier (/CH switch), the records are not checked for order. If you have only one input file, the records are listed in the output file in the same order as they are listed in the input file; if you have more than one input file, the order of the records on output may be unpredictable.

When you use sequence checking to verify that the records have been sorted, the records are still merged into an output file, which you must specify. If you are checking that records are sorted on a key field that is other than the entire record

(the default), then you must specify key-field information along with requesting sequence checking.

In addition to sequence checking, you can use `MERGE` on one or more files to change file characteristics such as format, organization, record size, or VFC size. For example, the following MCR command changes the file `SALARY.DAT` from a variable sequential file with a 50-byte maximum record size to a fixed relative file with 80-byte maximum record size (null-filled with binary 0 where necessary).

```
MGE> SALARY.DAT/RE/FO:F:80 = SALARY.DAT/-CH/FO:V:50
```

2.9 Optimizing the SORT/MERGE Work Area

By default, `SORT/MERGE` divides available work area between tree-related data structures and input/output-related data structures in such a way as to ensure the best performance for a typical sort operation. However, in some instances the input/output (I/O) requirements of your job may require more space than the default provides. The `/TREE_SPACE` qualifier (in DCL) or `/PT` switch (in MCR) allows you to override this default and choose the distribution of available work area between `SORT/MERGE` data tree structures and I/O data structures.

For `SORT`, the default division is 55 percent to the tree and 45 percent to I/O. For `MERGE`, the default division is 30 percent to the merge list and 70 percent to I/O. If you use a consistently large number of input files, or if the majority of the files you are sorting (for example, an `INDEXED` file with many key fields) require a large number of I/O data structures, you may want to alter the ratio so that there will be enough room for the I/O requirements. For example, if you are sorting several indexed files, each having many key fields, it may be desirable to allow `SORT` a smaller tree, thereby allocating more room for RMS-required structures.

In DCL, to allocate the work area, use the input file qualifier `/TREE_SPACE=n`, where *n* is the percentage of work space allocated to data tree structures.

In MCR use the input file switch `/PT:n`, where *n* is the percentage of work space allocated to data tree structures.

Using a Specification File

This chapter describes how to create and use a specification file. A specification file provides parameters and qualifiers for a sort or merge operation, supplementing and extending the SORT or MERGE command line. You can use a specification file to give you added control over your sort and merge operations. It can include instructions to perform the following functions:

- Change the format and length of the records in the output file
- Conditionally alter record order and data fields
- Specify certain records for the sort or merge process to include or omit
- Modify one of the predefined collating sequences or specify one of your own
- Reassign work files
- Specify an alternate record-padding character

To perform any of the following functions, it is necessary to use a specification file:

- Reformat the output records
- Use conditional key fields or data
- Specify multiple record formats
- Create or modify a collating sequence

You can also use a specification file to reassign work files, to define sort or merge operations that you use frequently, and to execute many of the sort or merge operations described in Chapter 2.

3.1 Creating a Specification File

To create a specification file, use a text editor. It is good practice to place each qualifier on a separate line. Generally, you can specify the qualifiers in a specification file in any order. The order becomes significant, however, when you use a specification file to perform the following tasks:

- Sort on more than one key field
- Describe the output format
- Define multiple record types
- Modify the collating sequence using subqualifiers with the /COLLATING_SEQUENCE keyword

To begin your specification file, enter whatever instructions you require for the sort process, equal key fields, and sequence checking. The default values for qualifiers in a specification file are the same as the default values for the corresponding command qualifiers.

You can use specification-file instructions in combination with SORT/MERGE command-line instructions, whether entered interactively or passed at the program level, but instructions entered at either command or program level override corresponding entries in the specification file. If you specify any /KEY qualifier in the DCL command line (or /KE switch in MCR), for example, SORT /MERGE ignores all /KEY, /DATA, /INCLUDE, and /OMIT qualifiers in the specification file.

Note that in the specification file syntax, whenever you have quotation marks within a quoted string, you must double each quotation mark. For example, "A""B" specifies the three-character string A"B.

One special use of the specification file is in combination with MERGE on a single file that may or may not have been previously sorted. The specification file gives you access to such features as record omission and record reformatting. For a single-file merge operation, you can specify /NOCHECK_SEQUENCE, so that MERGE will not check the order of the input records.

To include comments anywhere on a line in a specification file, use an exclamation point (!) as follows:

```
/KEY=YEAR           !Primary key field
/KEY=MONTH          !Secondary key field
/KEY=DAY            !Tertiary key field
```

NOTE

If you intend to place a specification file in a program, do not use comment characters (!), because the SORT/MERGE Utility may not interpret them correctly when the program is executed.

3.2 Processing a Specification File

To process a specification file, use one of the following command line formats:

DCL:

```
SORT /SPECIFICATION=specification-file input-file, input-file output-file
```

MCR:

```
SRT output-file = input-file,specification-file/SF
```

The following command lines sort the input files SALES1.DAT and SALES2.DAT into an output file named SALES.DAT, according to the instructions in the specification file named SPEC1.SRT.

```
DCL> SORT/SPECIFICATION = SPEC1 SALES1.DAT,SALES2.DAT SALES.DAT
MCR> SRT SALES.DAT = SALES1.DAT,SALES2.DAT,SPEC1/SF
```

The default file extension for a specification file is SRT.

When using a specification file, you can still include in the command line any qualifiers that you might use without a specification file. Any qualifier that you use in the DCL or MCR command line overrides any corresponding qualifier in the specification file.

When you call SORT or MERGE from a program, you can use a specification file either by identifying a specification file in your command line or by placing the specification file within a program and then passing it to SORT or MERGE.

Chapter 4 discusses using Callable SORT and MERGE from a program.

3.3 Specification File Qualifiers

Many of the qualifiers used in specification files are the same as the DCL qualifiers and subqualifiers listed in Chapter 2.

Qualifier	Restriction
/[NO]CHECK_SEQUENCE	MERGE only
/COLLATING_SEQUENCE	
/CONDITION	
/DATA	
/FIELD	
/INCLUDE	
/KEY	
/OMIT	
/PAD	
/PROCESS	SORT only
/[NO]STABLE	
/WORK_FILES	SORT only

Qualifiers

/[NO]CHECK_SEQUENCE

Verifies that the input files have been sorted. The positive form is the default; the negative form causes sequence checking to be waived.

/COLLATING_SEQUENCE=(ASCII)
(EBDIC)
(MULTINATIONAL)
(user-defined)

Specifies one of three predefined ordering systems, or a user-defined ordering system, for character data; ASCII is the default.

/CONDITION

Defines conditions for key and data handling and for record selection.

/DATA

Specifies the fields in the output file.

/FIELD

Defines the fields in the input file or files.

/INCLUDE

Selects records, as well as multiple record formats, for inclusion.

/OMIT

Selects records, as well as multiple record formats, for omission.

/KEY

Identifies key fields, including position, size, and data type of the field and the order of the sort operation.

/PAD

Specifies a new record-padding character. The default is the null character.

/PROCESS=ADDRESS

INDEX
RECORD
TAG

Defines the sort process; choose one only. RECORD (R) is the default.

/[NO]STABLE

Maintains the order of the input file when two or more key fields are equal. /NOSTABLE, which is the default, causes the order to be unpredictable.

/WORK_FILES=(NUMBER:0,3-10)

(DEVICE:ddnn)
(ALLOCATION:1-(2³²-1))
([NO]CONTIGUOUS)
(SIZE:1-255)

Specifies the number of work files for purposes of optimization, described in Chapter 5.

3.4 Identifying Record Fields

Whenever you wish to override the default values for fields, you must provide information about each field in the records. You must always provide the following information:

- A name that you assign to each field
- The position in the record and size of the field
- The data type of the field

To supply this information, you include a line for each field in the specification file, as follows:

```
/FIELD=(NAME=field-name,POSITION=n,SIZE=n,data-type)
```

The field name must begin with an alphabetic character and can include only 8 characters, which must be letters, numbers, or underscores. The SORT/MERGE Utility does not accept other characters or blank spaces in field names.

The POSITION subqualifier identifies the position of the field when used with the /KEY qualifier as described in Chapter 2; that is, the position is equal to the number of characters (bytes) from the beginning of the record.

The SIZE subqualifier gives the length of the field in bytes. You determine the size of the field exactly as when you use the SIZE subqualifier with the /KEY qualifier (as explained in Chapter 2).

The default data type with the /FIELD qualifier is character. The following data types are recognized by PDP-11 SORT/MERGE (the default data types are in boldface):

CHARACTER
ASCII_FLOATING
ASCII_ZONED
BINARY
SIGNED
UNSIGNED
DECIMAL

SIGNED
UNSIGNED
TRAILING_SIGN
LEADING_SIGN
OVERPUNCHED_SIGN
SEPARATE_SIGN
DIBOL_ZONED
DECIMAL
TRAILING
OVERPUNCHED SIGN
D_FLOATING
F_FLOATING
PACKED_DECIMAL

For example, in the sample magazine subscription file in Chapter 1, the data is arranged as follows:

Name (1-19)	Street (20-39)	City (40-51)	State (52-59)	Exp Date (60-65)
Yellen Mark	90 Lynwood Lane	Westfield	MA	901231
Germont Alfredo	15 Town House Dr	Waltham	MA	910501
Thompson Lynda	395 N Main St	Easton	MA	931130
Fallon Curtis	56 Juniper Lane	Lenox	MA	941101
Tosca Floria	108 Winfield Dr	Rome	NY	920630
Weaver Stephen	72 Newton Ave	Hyde Park	NY	940509
Marsh Beverly	305 Cambridge St	Pittsfield	MA	901015
Ling Kemlo	81 River St	Belmont	NY	941031

To identify the fields, use the following lines in a specification file:

```

/FIELD = (NAME=CUSTNAME, POSITION=1, SIZE=19)
/FIELD = (NAME=STREET, POSITION=20, SIZE=20)
/FIELD = (NAME=CITY, POSITION=40, SIZE=12)
/FIELD = (NAME=STATE, POSITION=52, SIZE=8)
/FIELD = (NAME=EXP_DATE, POSITION=60, SIZE=6)

```

In this example, all of the fields have a character data type, so it is not necessary to specify the data type within the /FIELD qualifier.

When you use the SIZE subqualifier, PDP-11 SORT/MERGE reads the size of all data types as byte lengths, and VAX SORT/MERGE also reads the size of all but decimal data as byte lengths. VAX SORT/MERGE reads the size of decimal data as digits. Therefore, if you want to use your files with VAX SORT/MERGE, you must specify the size of decimal data types in digits by using the DIGITS subqualifier instead of SIZE; for example:

```

/FIELD= (NAME=PERCENT, POSITION=28, DIGITS=4, DECIMAL)

```

When you use the DIGITS subqualifier, PDP-11 SORT/MERGE makes the conversion to byte lengths.

The size of a field that contains character data cannot exceed 255 characters (255 bytes). Specify a value of 1, 2, 4, or 8 for the size of a field containing binary data. The size of a field containing decimal data cannot exceed 31 bytes (or 31 digits with VAX SORT/MERGE). Do not specify size for the DCL D_FLOATING and F_FLOATING data types, because these data types have implicit sizes. Chapter 2 lists the default values for sizes and describes how to determine field sizes for data types other than CHARACTER or ASCII.

You can shorten any qualifier or subqualifier to its unique abbreviation (for example, /FIELD to /FI or POSITION to POS). Enclose the set of subqualifiers for each /FIELD qualifier entry in parentheses; separate the subqualifiers (for example, POS=60,SIZ=6) with commas.

3.4.1 Specifying Key Fields

If you are sorting the entire record and your data is characters, you need not specify a key field. Otherwise, you must use a `/KEY` qualifier for each of the keys by which you want to sort, in the order of their priority. You can sort by as many as 16 key fields. The `/KEY` qualifier provides information about how to sort a particular field. You identify a field to the `/KEY` qualifier by using the name that you assigned in the `/FIELD` qualifier.

To indicate multiple key fields to `SORT/MERGE`, use a series of `/KEY` qualifiers. The first key field that you list is the primary key field, the next is the secondary key field, and so on. For example, suppose your specification file includes the following three `/KEY` qualifiers:

```
/KEY=CUSTNAME  
/KEY=CITY  
/KEY=EXP_DATE
```

The primary key field is `CUSTNAME`, the secondary key field is `CITY`, and the tertiary key field is `EXP_DATE`.

The default sorting order is ascending; you must specify sort order for a key field only if you want the field sorted in descending order. Indicate descending order for a key field in the `/KEY` qualifier, for example:

```
/KEY=(EXP_DATE, DESCENDING)
```

Separate the subqualifiers in a `/KEY` qualifier with commas.

3.4.2 Formatting Data for the Output File

By default, the format of data for an output file is the same as that for the input file. For example, suppose you identify the data as we did in the magazine subscription file, as follows:

```
/FIELD = (NAME=CUSTNAME, POSITION=1, SIZE=19)  
/FIELD = (NAME=STREET, POSITION=20, SIZE=20)  
/FIELD = (NAME=CITY, POSITION=40, SIZE=12)  
/FIELD = (NAME=STATE, POSITION=52, SIZE=8)  
/FIELD = (NAME=EXP_DATE, POSITION=60, SIZE=6)
```

When you sort the file, the data in the output file will be arranged as indicated here. However, you can override this default arrangement with the `/DATA` qualifier. For example, suppose that you want the output data from this example arranged as follows:

State	City	Customer Name	Street
(1-2)	(6-20)	(24-43)	(47-66)

To ensure that the output-file data is arranged as it is in this example, you would use the `/DATA` qualifier in your specification file, as follows:

```
/DATA = STATE  
/DATA = CITY  
/DATA = CUSTNAME  
/DATA = STREET
```

The order in which you list the `/DATA` qualifiers, using the field names defined by previous `/FIELD` qualifiers, determines the ordering of fields in your output file. If you use the `/DATA` qualifier to change the formatting of your output file records, you must have a `/DATA` qualifier for each field that you direct to your output file.

If you want to have blank spaces between fields, you can use the /DATA statement with a pair of quotation marks to include spaces between fields. The number of spaces between the quotation marks is the number of spaces that is inserted between the fields; for example:

```
/DATA = STATE
/DATA = "  "
/DATA = CITY
/DATA = "  "
/DATA = CUSTNAME
/DATA = "  "
/DATA = STREET
```

3.4.3 Defining and Using Conditions

When you use a specification file, you can sort your records based upon certain conditions that you specify with the /CONDITION qualifier. The /CONDITION qualifier is used after a /FIELD qualifier; it can establish a means of reordering a field based on data that does not exist by itself in any specific field.

For example, suppose that you have a series of customer records in the magazine subscription list, as follows:

Name (1-19)	Street (20-39)	City (40-51)	State (52-59)	Exp Date (60-65)
Ling Kemlo	81 River St	Belmont	NY	941031
Campbell Aidan	16 Newheart Rd	Nashua	NH	911229
Tosca Floria	108 Winfield Dr	Rome	NY	920630
Weaver Stephen	72 Newton Ave	Hyde Park	NY	940509
Sacajewea	9 Slippery Rock Rd	Clifden	NJ	890103
Washington Paul	10 Mountain St	Johnstown	PA	900214
Kauffman Beverly	12 Steele St	Clinton	DE	911130
O'Brien Nelson	1324 Cherry St	Baltimore	MD	920529
Szymczak Pat	45 Hartford St	Roanoke	VA	930614

Now suppose that your sales area is divided into three regions, depending on the customer's home state. You can use the /CONDITION qualifier, followed by a /KEY qualifier and optionally a /DATA qualifier, to sort your records by sales region even though you do not have a field devoted specifically to sales region. Use the /CONDITION qualifier as follows:

```
/FIELD=(NAME=STATE, POS=52, SIZ=8)
/CONDITION=(NAME=REGION1, TEST=(STATE EQ "NY"))
/CONDITION=(NAME=REGION2, TEST=(STATE EQ "NJ" OR STATE EQ "PA"))
/CONDITION=(NAME=REGION3, TEST=(STATE EQ "DE" OR
STATE EQ "MD" OR
STATE EQ "VA"))

/KEY = (IF REGION1 THEN 1 ELSE
IF REGION2 THEN 2 ELSE
IF REGION3 THEN 3 ELSE
4)

/DATA = (IF REGION1 THEN "REGION 1" ELSE
IF REGION2 THEN "REGION 2" ELSE
IF REGION3 THEN "REGION 3" ELSE
"ERROR ")

/DATA = STATE
/DATA = "  "
/DATA = CITY
/DATA = "  "
/DATA = CUSTNAME
/DATA = "  "
/DATA = STREET
```

The information supplied with the /FIELD qualifier identifies the field on which the conditional testing is to be done. The /CONDITION qualifier tests for matches between record data and the values that you specify. When data in a record field matches a value in a /CONDITION qualifier, a sorting tag (such as "1") is given to the record field by means of the /KEY qualifier.

In this example, REGION1 is the name of a test that succeeds when the value "NY" is in the STATE field; REGION2 is a test that succeeds when "NJ" or "PA" is in the STATE field; and REGION3 is a test that succeeds when "DE," "MD," or "VA" is in the STATE field. The /KEY qualifier then assigns sorting values to the tested field; in this case, the digits 1 through 3 for the three regions, and the digit 4 for all other values). When your records are sorted with this /KEY statement, the values that you assign with the /KEY qualifier are the basis for sorting; that is, all of the records with value "1" (which was defined as the test REGION1, which was in turn defined as the value "NY" in the STATE field) are listed, followed by the records with value "2," and so on.

The /DATA qualifier attaches a text string to the output file. In this example, the records with the sort tag "1" have the text string "REGION 1," and so on. Thus, the output for each record includes the customer's name, street address, city, and state, as well as a designation of "REGION 1," "REGION 2," or "REGION 3." Any customer address that is not entered with an acceptable state code (NY, NJ, PA, DE, MD, or VA), such as Aidan Campbell's, is output with an error message.

To define conditionals, use the TEST subqualifier with the following two-letter operators:

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

In collating terms, *less than* means coming before in sequence, and *greater than* means coming after in sequence. Sequence is determined by the number that a character or digit equates to in the collating sequence you are using. For example, uppercase *A* is equivalent to 65 in the DIGITAL Multinational Collating Sequence, and uppercase *Z* is equivalent to 90. Appendix C lists the DIGITAL Multinational Collating Sequence and the ASCII Collating Sequence.

Use AND and OR to include more than one conditional test with a TEST subqualifier, and enclose TEST and its parameters in parentheses. If the data in the field is alphabetic and you use the operators GT, GE, LT, or LE, then by default the ASCII value of the alphabetic data is compared to the ASCII value of the text you supplied with the TEST subqualifier.

3.4.3.1 Changing the Contents of a Field

You can also use the /CONDITION qualifier to change the contents of a field. For example, suppose that some of your records contain a spelling error, such as "CINNCINATTI," which should be spelled "CINCINNATI.") You could use the /CONDITION qualifier to correct the error as follows:

```
/FIELD = (NAME = CITY, POS = 6, SIZ = 15)
/CONDITION = (NAME = CINCI
              TEST = (CITY EQ "CINCINNATI" OR
                     CITY EQ "CINNCINATTI"))
```

Next, you define the text that you want to replace either of the two conditions by using the /DATA qualifier:

```
/DATA=(IF CINCI THEN "CINCINNATI")
```

When the data is sorted, all references in the field CITY to either CINCINNATI or CINNCINATTI will appear in the output file as CINCINNATI. Since no /KEY qualifier was used with the /CONDITION qualifier, this operation does not affect the order of the sorted output.

3.4.3.2 Specifying Records for Inclusion or Omission

To select records that you want to include or exclude from the sort or merge operation, use the /CONDITION qualifier as a subqualifier to an /INCLUDE or /OMIT qualifier. For example, if you wanted to sort only the records of customers in California, you could use the following syntax:

```
/FIELD = (NAME = STATE, POS = 56, SIZ = 2)
/CONDITION = (NAME = CALIF
              TEST = (STATE EQ "CA"))
/INCLUDE=(CONDITION = CALIF)
```

In this example, only those records that satisfy the condition named CALIF (that is, only those records with "CA" in the STATE field) are included in the sorting or merging process defined in the remainder of the specification file. If you use the /CONDITION qualifier with an /OMIT qualifier, all records satisfying the named condition are excluded from the sorting or merging process.

The order in which you list /INCLUDE or /OMIT statements is the order in which they are evaluated. Thus, if you exclude a record with an /OMIT qualifier and subsequently include the record with an /INCLUDE qualifier, the record is included in the output file. However, only the key fields that were indicated prior to the /OMIT statement and after the /INCLUDE statement are sorted.

When you use /OMIT or /INCLUDE, you can either specify the criteria for omission or inclusion (/OMIT=(CONDITION=condition-name) or not specify any criteria (/OMIT without a CONDITION statement). If the last instruction that specifies criteria is /OMIT, then everything not specifically omitted is included; if the last instruction that specifies criteria is /INCLUDE, then everything not specifically included is omitted. If the last instruction that does not specify criteria is /INCLUDE or /OMIT, then everything not specifically omitted is included (with /INCLUDE), or everything not specifically included is omitted (with /OMIT).

3.4.3.3 Representing Data Other Than Character or ASCII in Conditions

Note that the data supplied with the TEST subqualifier to the /CONDITION qualifier must be of the same type as the data in the field being tested. In the sales region example in Section 3.4.3, the data supplied with TEST is NY, NJ, PA, DE, MD, and VA, which is character data, the same type of data contained in the STATE field: NY, NH, NJ, PA, DE, MD, and VA.

If the data in the field being tested is numerical and non-ASCII, care must be taken to ensure that the data is represented correctly. SORT/MERGE translates character and ASCII data into its machine-language representation, which is necessary for correct processing by the Central Processing Unit (CPU). The utility does not translate other types of data. Therefore, if you wish to sort other types

of data, you must represent it in octal or hexadecimal form, which can be read by the CPU. The octal representation must be preceded by %O and the hexadecimal by %X.

For example, SORT/MERGE does not translate the ASCII representation of the packed decimal number shown in the following /CONDITION statement:

```
/PROCESS=RECORD
/FIELD = (NAME = MYREC, POS = 1, SIZ = 4, PACKED_DECIMAL)
/CONDITION = (NAME = TESTREC, TEST = (MYREC EQ "0100003C"))
/INCLUDE = (CONDITION = TESTREC)
```

The test condition in this /CONDITION statement will result in an error message. The decimal number being represented is 0.00001. To use the PACKED DECIMAL data type with this number, you can represent the number in octal form, as follows:

```
/PROCESS=RECORD
/FIELD = (NAME = MYREC, POS = 1, SIZ = 4, PACKED_DECIMAL)
/CONDITION = (NAME = TESTREC, TEST = (MYREC EQ %O11400000001))
/INCLUDE = (CONDITION = TESTREC)
```

For further information on the use of other types of data, consult the user's guide to your programming language.

3.5 Sorting Files with More than One Record Format

By specifying condition tests and record selection, you can sort records that have their fields formatted differently. Suppose you have two files from two different branches of a real estate agency. The records in the first file start with an "A" in the first position and are formatted as follows (the beginning position of each field is indicated below the format):

```
A  PRICE  TAXES  STYLE  ZIP
1  2      10     14     24
```

The records in the second file start with a "B" in the first position but have the style and zip code fields reversed, as follows:

```
B  PRICE  TAXES  ZIP  STYLE
1  2      10     14  19
```

Suppose you want these two files sorted on the zip code field in the format of record "A." For this sort operation, you indicate the following information in your specification file. (Comments are permitted in the text of a specification file and begin with an exclamation mark, as shown in this example. Do not use comments if you will use the specification file in a program that calls SORT or MERGE.)

```

/FIELD = (NAME=REC_TYPE, POS=1, SIZ=1)      ! Record's type, 1-byte field
/FIELD = (NAME=PRICE, POS=2, SIZ=8)        ! Price field, both files
/FIELD = (NAME=TAXES, POS=10, SIZ=5)       ! Taxes field, both files
/FIELD = (NAME=STYLE_A, POS=14, SIZ=10)    ! Style field, format A file
/FIELD = (NAME=STYLE_B, POS=19, SIZ=10)    ! Style field, format B file
/FIELD = (NAME=ZIP_A, POS=24, SIZ=5)       ! Zip code field, format A file
/FIELD = (NAME=ZIP_B, POS=14, SIZ=5)       ! Zip code field, format B file
/CONDITION = (NAME=FORMAT_A,              ! Condition test, format A file
              TEST=(REC_TYPE EQ "A"))
/CONDITION = (NAME=FORMAT_B,              ! Condition test, format B file
              TEST=(REC_TYPE EQ "B"))
/INCLUDE = (CONDITION=FORMAT_A,           ! Output format, type-A records
            KEY=ZIP_A,
            DATA=PRICE,
            DATA=TAXES,
            DATA=STYLE_A,
            DATA=ZIP_A)
/INCLUDE = (CONDITION=FORMAT_B,           ! Output format, type-B records
            KEY=ZIP_B,
            DATA=PRICE,
            DATA=TAXES,
            DATA=STYLE_B,
            DATA=ZIP_B)

```

Thus, on output, this sort operation changes the format of the records of type B to that of the records of type A.

NOTE

If you specify any key fields or data fields with the `/INCLUDE` qualifier, you must explicitly specify all the key fields and data fields in the operation with the `/INCLUDE` qualifier.

By default, the key fields are not prefixed to the output record. However, you can specify conditional key and data fields, as explained in the previous example, to override this default.

Section 3.10 includes a sample specification file, which shows the use of the `/CONDITION` statement.

3.6 Specifying a Collating Sequence

The default collating sequence for character data is ASCII. You can specify ASCII, EBCDIC, Multinational, or your own collating sequence, as follows:

```

                                (ASCII)
                                (EBCDIC)
/COLLATING_SEQUENCE=(SEQUENCE= (MULTINATIONAL) )
                                (user-defined-sequence)

```

The `MULTINATIONAL` collating sequence is the `DIGITAL Multinational Collating Sequence` listed in Appendix C. The `ASCII Collating Sequence` is listed in Appendix C also.

3.6.1 Defining Your Own Collating Sequence

This section describes how you can modify the ASCII, EBCDIC, and Multinational collating sequences to suit your needs. If none of these collating sequences is suited to your needs, you can define your own. (You can modify your own collating sequence also; for more information on modifying your own sequence, see Section 3.6.3.)

To define your own collating sequence, specify a string of characters (single or double), or ranges of single characters. A double character is any set of two single characters that you want to collate as if they were a single character. Enclose each character in quotation marks, separate characters (or sets or ranges) by commas, and enclose the entire list in parentheses. For example:

```
/COLLATING_SEQUENCE=(SEQUENCE=("A"- "L", "Ll", "M"- "Z"))
```

This sequence signifies that the double character *Ll* collates as a single character between *L* and *M*. If you were to use this collating sequence to sort a field containing the names "Lancaster, Llewellyn, and Lonergan," the names would be sorted in the order of "Lancaster, Lonergan, and Llewellyn" instead of their normal alphabetical order. By default, this collating sequence does not define the lowercase characters *a* through *z*.

When you specify a collating sequence, uppercase characters and lowercase characters are treated separately. By default, uppercase characters are collated before lowercase characters. To collate lowercase characters before uppercase, include the following line in your specification file:

```
/COLLATING_SEQUENCE=(SEQUENCE=("a"- "z", "A"- "Z"))
```

The records are collated first using lowercase *a-z* and then uppercase *A-Z*, as in the following file, VEG.DAT:

```
cucumbers  
eggplant  
turnip  
ASPARAGUS  
MUSHROOMS  
ZUCCHINI
```

When you define a collating sequence, SORT/MERGE creates an ordering table based on the sequence that you define. This table replaces the predefined ASCII, EBCDIC, or MULTINATIONAL tables. When you have finished creating your own collating sequence, the corresponding ordering table should represent a complete specification of all the characters appearing in the character key fields in your sort or merge operation. SORT/MERGE ignores any character to which you have not given a collating value.

The following rules apply to defining a collating sequence:

- Define a character only once.
- Specify the null character with the %X0 rather than "". %X is the hexadecimal radix operator. You can also represent other characters by their corresponding octal, decimal, or hexadecimal values, by using the octal, decimal, and hexadecimal radix operators, %O, %D, and %X respectively. For information on representing non-ASCII data in a specification file, see Section 3.4.3.3.
- Specify quotation marks (") by enclosing them within quotation marks (""""") or by using a radix operator.

3.6.2 FOLD and TIE_BREAK Subqualifiers

To cause uppercase and lowercase characters to be collated together, use the subqualifier FOLD with the /COLLATING_SEQUENCE qualifier. When you use FOLD, SORT/MERGE does not discriminate between uppercase and lowercase characters. Use FOLD as follows to sort VEG.DAT:

```
/COLLATING_SEQUENCE=(SEQUENCE=("a"- "z", "A"- "Z", FOLD))
```


The output is as follows:

```
ASPARAGUS
cucumbers
eggplant
MUSHROOMS
turnip
ZUCCHINI
```

FOLD causes all lowercase characters to be given the collating value of their uppercase equivalents. In effect, FOLD is the same as using the following expression:

```
MODIFICATION = ("a"="A", "b"="B", ... "z"="Z")
```

If you specify FOLD *after* you define a double character that contains no lowercase letters (for example, "CH">"C"), then any lowercase or mixed-case combinations of the defined double character will have a collating value equivalent to the defined double character. (For example, "ch", "Ch", and "CH" all have the same collating value greater than "C".) However, if you specify FOLD *before* you define the "CH">"C" double character, then only the uppercase "CH" collates greater than "C."

Use TIE_BREAK to indicate that you want further processing to be performed after an initial comparison of collating values results in equal values. This tie-breaking process arranges the characters according to a predefined order, as shown in the DIGITAL Multinational Character Set collating sequence table in Appendix C. For ASCII, EBCDIC, and any user-defined collating sequence, the tie-breaking is based on the numeric code values of the characters. You must explicitly specify tie-breaking for these character sets; you should usually use tie-breaking after specifying FOLD or MODIFICATION.

With the Multinational Collating Sequence, tie-breaking is the default unless you explicitly specify NOTIE_BREAK. If you use NOTIE_BREAK with the Multinational Collating Sequence, only an initial comparison of the collating values is made, and some unexpected ordering may result.

3.6.3 Modifying the Collating Sequence

You can modify whatever collating sequence you select by instructing SORT/MERGE to change the order in which certain characters appear in the given sequence. To indicate your instructions for any modifications, use the keyword MODIFICATION with the /COLLATING_SEQUENCE qualifier, as follows:

```
/COLLATING_SEQUENCE=(SEQUENCE=collating-sequence
                      {character > character
                       ,MODIFICATION=( character < character [,...]
                                     character = character}
```

The following rules apply to using the /COLLATING_SEQUENCE qualifier:

- Use a comma between subqualifiers.
- Enclose the characters to be modified in quotation marks or use a radix.
- If you make more than one modification to your collating sequence, separate the modifications with commas.

To modify any of the predefined collating sequences (ASCII, EBCDIC, or MULTINATIONAL), or to use FOLD, IGNORE, or TIE_BREAK, specify the sequence in the /COLLATING_SEQUENCE qualifier as follows: /COLLATING_SEQUENCE = (SEQUENCE = (EBCDIC), MODIFICATION= . . .).

The kinds of modifications permitted with the MODIFICATION subqualifier are as follows:

- Equating a single or double character to a single character. The second character must already have a collating value.

Thus, if you want to modify the previous example of the user-defined collating sequence so that *M* has the same collating value as *N*, specify the following:

```
,MODIFICATION= ("N"="M")
```

- Causing a single or double character to collate after a single character that has already been assigned a collating value.

For example, if you want *M* to collate after *N*, express this modification as follows:

```
,MODIFICATION= ("M">"N")
```

- Causing a single or double character to collate before a single character that has already been assigned a collating value.

If, for example, you want the double character *CH* to collate after *C* and before *D*, you specify this modification in either of the following ways:

```
,MODIFICATION= ("CH"<"D")  
,MODIFICATION= ("CH">"C")
```

- Equating a double character to a previously defined double character.

For example, if you have previously assigned a value to the double character *PH*, you can then equate the double character *GH* to it, as follows:

```
,MODIFICATION= ("GH"="PH")
```

- Equating a single character to a two-character sequence.

Thus, if you want the ligature *Æ* to collate in the same position as the two-character sequence of *AE*, you specify the following:

```
,MODIFICATION= ("Æ"="AE")
```

To request that SORT/MERGE ignore a character or character range within the given collating sequence, use the IGNORE subqualifier, as follows:

```
,IGNORE=(character)
```

For example, the following line:

```
,IGNORE=("-", " ")
```

would cause the following fields to be compared as equal:

```
252-3412
```

```
252 3412
```

```
2523412
```

In the MULTINATIONAL collating sequence, two defaults exist that are not present in the ASCII and EBCDIC collating sequences, FOLD and TIE_BREAK.

To override the default tie-breaking algorithm when using the MULTINATIONAL collating sequence, specify the subqualifier NOTIE_BREAK. To use tie breaking in the other collating sequences, specify the subqualifier TIE_BREAK.

3.6.4 Example of a User-Defined Collating Sequence

The following file, named SEMNAR.DAT, contains a schedule of seminars sorted by title:

```
11 Jan '93   Assertiveness Training
16 NOV 1994  Communication Skills
05 APR 1993  Coping with Alcoholism
12 OCT 1994  Improving Productivity
15 MAR 1993  Living with Your Teenager
08 FEB 1993  Single Parenting
07 Dec '94   Stress: Causes and Cures
14 SEP 1994  Time Management
```

To sort the file by date, assign the year field as the primary key and the month field as the secondary key. Because the month field is not numeric and you want the months ordered chronologically, you must define your own collating sequence. You can do this by sorting on the second two letters of each month in their chronological sequence, thereby giving each month a unique abbreviation.

Specify the specification file text for this sort operation in the following file, SPEC.SRT:

```
/FIELD=(NAME=DAY,PO=1,SIZ=2)
/FIELD=(NAME=MONTH,PO=5,SIZ=2)
/FIELD=(NAME=YEAR,PO=8,SIZ=4)
/KEY=YEAR                                ! Primary key field
/KEY=MONTH                                ! Secondary key field
/KEY=DAY                                  ! Tertiary key field

/COLLATING_SEQUENCE=(SEQUENCE=           ! User-defined sequence
 ("AN", "EB", "AR", "PR", "AY", "UN", "UL", ! that gives each month
 "UG", "EP", "CT", "OV", "EC", "O"- "9"), ! a unique value
MODIFICATION= (" " = "19"),             ! in its chronological order
FOLD)
```

Include this specification file in a SORT command string as follows:

```
DCL> SORT/SPECIFICATION=SPEC.SRT SEMNAR.DAT SCHED.DAT
MCR> SCHED.DAT=SEMNAR.DAT, SPEC.SRT/SF
```

The output from this sort operation appears as follows:

```
11 Jan '93   Assertiveness Training
08 FEB 1993  Single Parenting
15 MAR 1993  Living with Your Teenager
05 APR 1993  Coping with Alcoholism
14 SEP 1994  Time Management
12 OCT 1994  Improving Productivity
16 NOV 1994  Communication Skills
07 Dec '94   Stress: Causes and Cures
```

3.7 Reassigning Work Files

Placing work files on different disk-structured devices, by means of a specification file, maximizes the performance of SORT. Specify the reassignment as follows:

```
/WORK_FILES=(workfile,workfile,...)
```

where *workfile* is in the format *ddnn*: (for example, /WORK_FILES = (db0:,dm0:)). The first work file is placed on the first device listed, the second work file on the second device listed, and so on.

3.8 Specifying a New Pad Character

By default, SORT/MERGE uses a null character to pad records. However, you can specify your own pad character to reformat records or to compare strings of different lengths by using the /PAD qualifier in your specification file. Do not specify a double character as a pad character, even if you equate a double character to a single character elsewhere in the specification file. The format for specifying a pad character is as follows:

/PAD=x

where *x* is one of the following:

%D<decimal-digit>
 %O<octal-digit>
 %X<hex-digit>
 "<character>"

3.9 Format of Qualifiers in a Specification File

This section lists the formats of the qualifiers and subqualifiers that you can use in a specification file. The notation used is as follows: brackets [] indicate that an element is optional; braces { } indicate that you may select only one of the possible choices; parentheses () indicate that parentheses must enclose your choices; and boldface indicates default elements.

[/NO]CHECK_SEQUENCE ! This qualifier is applicable to merge operations only.

/COLLATING_SEQUENCE=

(SEQUENCE= { (ASCII)
 (EBCDIC)
 (MULTINATIONAL)
 (user-defined sequence) } , [MODIFICATION=(char { >
 <
 = } char, ...)
 IGNORE=({ char
 char-range } , ...)
 FOLD
 [/NO]TIE_BREAK])

/CONDITION=(NAME=condition-name, TEST=(field-name1 { EQ
 NE
 GT
 GE
 LT
 LE } { field-name
 constant } ... [{ AND
 OR }]))

field-name1 { [EQ]
 [NE]
 [GT]
 [GE]
 [LT]
 [LE] } { field-name
 constant }]))

/DATA=key-data-clause ! Format of *key-data-clause* is as follows:

$$\left\{ \begin{array}{l} \text{field-name} \\ \left\{ \begin{array}{l} (\text{field-name}) \\ (\text{constant}) \\ (\text{IF condition-name THEN constant}) \\ ([[\text{ELSE}] \text{IF condition-name THEN constant} \dots]) \\ (\text{ELSE constant}) \end{array} \right\} \\ \\ [(, \text{ASCENDING})] \\ [(, \text{DESCENDING})] \end{array} \right\}$$

/FIELD=

(NAME=field-name, POSITION=integer, { SIZE=1–255 lcharacter data
1,2,4,8 lbinary data
DIGITS=1–31 ldecimal data }, [CHARACTER]
[ASCII_FLOATING]
[ASCII_ZONED]
[BINARY]
[SIGNED]
[UNSIGNED]
[DECIMAL]
[SIGNED]
[UNSIGNED]
[TRAILING_SIGN]
[LEADING_SIGN]
[OVERPUNCHED_SIGN]
[SEPARATE_SIGN]
[DIBOL_ZONED]
[D_FLOATING]
[F_FLOATING]
[PACKED_DECIMAL])

/INCLUDE= [(CONDITION=condition-name)
, (KEY=key-data-clause . . .)
, (DATA=key-data-clause . . .)]

/KEY=key-data-clause ! See the /DATA qualifier for the format of *key-data-clause*.

/OMIT=[(CONDITION=condition-name)]

/PAD= { %D<decimal-digit>
%O<octal-digit>
%X<hexadecimal-digit>
"<character>" }

<char> ::= { <single_char>
<double_char> }

```

<single_char> ::= {
    <character>
    %D<decimal-digits>
    %O<octal-digits>
    %X<hex_digits>
    "<character>"
}

<double_char> ::= {
    <single-char><single-char>
    "<character><character>"
}

<char-range> ::= <single-char>—<single-char>

<user-defined-sequence> ::= {
    <char>
    <char-range>, ...
}

<constant> ::= {
    <decimal-digits>
    %O<octal-digits>
    %X<hex-digits>
    "<character>"
}

```

```

/PROCESS= {
    RECORD
    TAG
    ADDRESS
    INDEX
} ! This qualifier is applicable to sort operations only.

```

/WORK_FILES=(workfile,workfile, ...) ! The format of the *workfile* parameter is ddnn:. This qualifier is applicable to sort operations only.

3.10 Sample Specification File

```

! Sort Specification File
!
/FIELD=(NAME=AGENT, POSITION=1, SIZE=15)
/FIELD=(NAME=ZIP, POSITION=16, SIZE=5)
/FIELD=(NAME=STYLE, POSITION=21, SIZE=1)
/FIELD=(NAME=CONDITION, POSITION=22, SIZE=1)
/FIELD=(NAME=PRICE, POSITION=23, SIZE=8)
/FIELD=(NAME=TAXES, POSITION=31, SIZE=4)
!
/CONDITION=(NAME=LOCATION, TEST=(ZIP EQ "01863"))
/CONDITION=(NAME=GAMBREL, TEST=(STYLE EQ "1"))
/CONDITION=(NAME=SPLIT, TEST=(STYLE EQ "2"));
/CONDITION=(NAME=TRILEV, TEST=(STYLE EQ "3"))
/CONDITION=(NAME=RANCH, TEST=(STYLE EQ "4"))

```

```

!
/KEY=(IF LOCATION THEN 1 ELSE 2)
/KEY=ZIP
/DATA=ZIP
/DATA=" "
/DATA=PRICE
/DATA=" "
/DATA=TAXES
/DATA=" "
/DATA=(IF GAMBREL THEN "GAMBREL " ELSE
        IF SPLIT THEN "SPLIT LEVEL" ELSE
        IF TRILEV THEN "TRI-LEVEL " ELSE
        IF RANCH THEN "RANCH " ELSE
        "UNKNOWN ")
/DATA=" "
/DATA=CONDITION
/DATA=" "
/DATA=AGENT

```

NOTE

The exclamation-point comment character (!) is included in this sample specification file. If you plan to include the specification file in a program, do not use any comment characters.

Using SORT and MERGE in a Program

This chapter describes how to access SORT/MERGE from a program at run time, using the subroutines that SORT/MERGE provides. You can use SORT/MERGE to arrange data before or after it is processed by a program. This chapter covers the languages that support SORT/MERGE, the two callable interfaces, and the SORT/MERGE subroutines and their parameters.

Appendix B of this manual contains six sample programs that demonstrate how to use the callable SORT and MERGE subroutines.

4.1 Language Support

The following PDP-11 native-mode languages allow you to invoke SORT/MERGE from a program:

- BASIC-PLUS-2
- COBOL-81
- MACRO-11
- PDP-11 C
- PDP-11 FORTRAN-IV
- PDP-11 FORTRAN-77

Note that COBOL-81 includes the COBOL syntax for many SORT and MERGE functions. You can use the SORT/MERGE subroutines directly for those functions that are not provided by the ANSI COBOL standard syntax.

Individual calls from your program can access a specific SORT or MERGE subroutine and pass parameters to it. You place the syntax to define your sort or merge operation in your calling program.

Calls and the associated parameters conform to the calling standard of PDP-11 FORTRAN. The parameters used in calling SORT/MERGE are passed by reference.

BASIC and COBOL allow you to pass data descriptors for string or character fields. BASIC uses two types of descriptors: string descriptors and array descriptors. The BASIC string descriptor is 2 words, containing the address and the length, in that order. The BASIC array descriptor is also 2 words, but it lists the length first and then the address. The COBOL descriptors used for SORT/MERGE are all 2-word descriptors, containing a length first and then an address. See the BASIC or COBOL documentation for your operating system to learn about using BASIC or COBOL descriptors and the required order for passing information.

If you write your program in BASIC or COBOL and pass information by descriptor, the SORT subroutine names are slightly different from the subroutine names for the other supported languages. If you use BASIC or COBOL and do not pass information by descriptor, use the same SORT subroutines as for all of the other supported languages.

All those parameters for which SORT/MERGE requires only an address, and not a length, are passed by reference.

Because programming languages express parameters differently, this chapter does not give detailed instructions for each language. For further information, see the reference manual or user's guide for the PDP-11 programming language in which you are writing your program.

At installation time, the SORT/MERGE subroutines are placed in the system library directory. When permitted by your programming language, it is good practice to use function references to invoke the subroutines.

4.2 Accessing Callable SORT and MERGE

You can access the SORT or MERGE subroutines through one of two interfaces: the file interface and the record interface. The file interface allows you to submit your records for sorting or merging as complete files. The record interface allows you to submit your records individually. You can use both interfaces within the same SORT or MERGE operation by using one interface for input and the other for output; this is called using a mixed-mode interface.

When your program submits one or more files to SORT or MERGE (resulting in the creation of one sorted or merged output file), you are using the file interface. When your program submits records one at a time and then receives the ordered records one at a time, you are using the record interface. You can combine the file interface with the record interface by having your program perform one of the following steps:

- Submit files as input and receive the output as ordered records
- Submit records as input and have the ordered records written to an output file

The file interface executes faster than the record interface and is easier to incorporate into your program. When you use this interface, you sort (or merge) all records in the files without processing them either before or after sorting. When you use the record interface, you can perform an operation on each record before or after sorting. For example, you would use the record interface if you want to keep a tally of the number of duplicate records that are returned to your program.

If you use the mixed-mode interface with the file interface on input, you can perform an operation on the records after they are sorted. In the mixed-mode interface with the file interface on output, you can perform an operation on the records before they are sorted. The calls that you use in your program differ for the file and record interfaces, as described later in this chapter.

4.3 Specifying Your Own Routines

You can specify your own routines to accomplish special tasks for your sort or merge operation. For example, you can specify your own key-field comparison routine. (All these routines are explained in detail in the descriptions of the subroutine parameters later in this chapter.) However, since a BASIC routine can be called only by another BASIC routine, these user-defined routines cannot be written in BASIC-PLUS-2.

Depending on what your programming language allows, you specify the use of your own routines in one of two ways:

1. If the language you are using permits, specify the address or addresses of your routine or routines as an optional parameter in the first SORT/MERGE subroutine called in your program.
2. If you cannot use this method with your programming language, you can write these routines as separate subprograms. You must use the same global symbols for the entry points as SORT and MERGE use for their default processing: SRTxxx for SORT, and MRGxxx for MERGE. When task building, you must modify the appropriate SORT or MERGE overlay description language file (ODL), as explained later in this chapter. In this way, the addresses of the routines will be resolved with your object module, rather than with the default SORT/MERGE subroutines.

Pass all parameters to these routines by reference. Begin each argument list with a word containing the number of parameters being passed. Use Register 5 (R5) as the linkage register.

4.4 Calling the SORT Subroutines

SORT requires the same type of user input whether you access the utility from a program, from the command level, or from a specification file. Specifically, you provide the following information:

- File specifications (when you use the file or mixed-mode interface)
- Information about key fields (for example, position, size, and data type)
- Instructions about the sorting process

You pass this information to SORT by using subroutine parameters. After being called, each subroutine performs its function and then returns control to the program. One of the parameters to SORT and MERGE subroutine calls is a 4-word error buffer. The routine status is placed in word 1 before control is returned to your program. Words 2, 3, and 4 may contain additional information, depending on the nature of any error that occurs. For example, if an I/O error occurs during a call to SORT, words 2 and 3 of the error buffer will contain the error's status-code-field (STS) and status-value-field (STV) values. (See your operating system's Record Management Services (RMS) documentation for more information on these values.) Your program can test the values of words 1 to 4 to determine success or failure conditions.

SORT subroutine have both both required and optional parameters. Required parameters appear first in the argument list; you can include optional parameters only after you have listed all of the required parameters. Include all parameters in the order in which they are positioned in the argument list, using a comma to separate them. Null parameters are indicated when no value follows the comma in the parameter's position in the argument list. If your programming

language does not permit null parameters, use a 0 or -1 to indicate them in the parameter's position in the argument list. You can end your argument list after you have specified all the required parameters.

Table 4-1 lists the standard calls for the record and file interfaces and briefly describes the function of each. The following sections describe each SORT subroutine in detail, including required and optional parameters.

Table 4-1: SORT Subroutines

Subroutine	Function
File Interface	
SRTINI	Initializes sort operation by passing file names, key information and sort options
SRTSRT	Reads the input file or files, sorts the records, and writes the records to the output file
SRTEND	Performs cleanup functions, such as closing files and releasing memory
Record Interface	
SRTINI	Initializes sort operation by passing key information and sort options
SRTRLS	Passes one input record to SORT; must be called once for each record
SRTRTN	Returns one sorted record to your program; must be called once for each record
SRTEND	Performs cleanup functions, such as closing files and releasing memory

4.4.1 Using the File Interface

For a sort task using the file interface, first call the initialization subroutine SRTINI. Note that if you are using BASIC or COBOL and pass information by descriptor, you use different entry points to initialize a sort operation: the initializing subroutine for BASIC is SRTINB; the initializing subroutine for COBOL is SRTINC. (If you write your program in BASIC or COBOL and you do not pass information by descriptor, use the SRTINI subroutine.)

The first parameters passed to SRTINI define the address of an error buffer and set up work areas. Then you pass the address of a command line buffer, in which you use an MCR command line to specify your input and output file names and your instructions about key fields and sort options. You also indicate whether you want comparisons of key fields to be done by SORT or by your own key-field comparison routine. You may want to provide your own comparison routine to handle special sorting requirements; for example, if you are using a data type not supported by SORT.

The next step is to call SRTSRT (SRTSRB for BASIC, SRTSRC for COBOL) to execute the sort and to direct the sorted records to the output file. Finally, call SRTEND (SRTENB for BASIC, SRTENC for COBOL) to end the sort and to release resources.

A program may call the SRTEND subroutine at any time between calls to the other subroutines to abort a sort operation and to release all resources allocated to the sort or merge process. If a fatal error condition occurs, SORT automatically releases all allocated resources.

4.4.2 Using the Record Interface

When you are using the record interface, first call SRTINI (SRTINB for BASIC, SRTINC for COBOL). As with the file interface, this subroutine sets up work areas and passes parameters that define key fields and sort options.

Next, call SRTRLS (SRTRLB for BASIC, SRTRLC for COBOL) to release a record to the sort process. Your program must call SRTRLS once for each record to be released.

Now, call SRTRTN (SRTRTB for BASIC, SRTRTC for COBOL) to return the sorted records to your program. Your program must call SRTRTN once for each record to be returned. When all the records have been returned, an end-of-file code is returned to the error buffer on the next call to SRTRTN.

After each record has been returned, call the last subroutine, SRTEND (SRTENB for BASIC, SRTENC for COBOL), to complete the sort task and release memory.

4.4.3 Using Mixed-Mode Interface

When you are using a mixed-mode interface, order the SORT subroutine calls to match your output interface. If you use the file interface on output, use the calls SRTINI, SRTSRT, and SRTEND. If you use the record interface on output, use the SRTINI, SRTSRT, and SRTRTN subroutines once for each record, followed by the SRTEND subroutine to end the MERGE operation.

4.4.4 Passing File Names and Initializing the Sort Process

As described in Section 4.4, each interface (file, record, and mixed-mode) begins with a call to the SRTINI subroutine. (For BASIC, the corresponding subroutine is SRTINB; for COBOL, it is SRTINC.) You use this subroutine to pass files, if there are any, and to pass key-field information and key-field options.

When you call the SRTINI initializing subroutine, you must include 6 required parameters, and you may include and 8 optional parameters. The SRTINB (for BASIC) and SRTINC (for COBOL) subroutines each have 4 required parameters and 6 optional parameters. Table 4–2 lists the required and optional parameters for SRTINI and usage information for SRTINB and SRTINC. A discussion of each parameter follows the table. (Unless otherwise noted, pass all parameters by reference.)

Table 4–2: Parameters for SRTINI, SRTINB, and SRTINC

Parameters	BASIC/COBOL Usage Information
1. Error address	
2. Work area address	Pass by descriptor
3. Work area length	Omit
4. Command line buffer	Pass by descriptor
5. Command line length	Omit
6. Longest record length ¹	

¹Required when longest record length is unavailable

(continued on next page)

Table 4–2 (Cont.): Parameters for SRTINI, SRTINB, and SRTINC

Parameters	BASIC/COBOL Usage Information
7. Specification file buffer ²	Pass by descriptor
8. Specification file buffer length ²	Omit
9. Logical unit number (LUN) buffer ²	Pass by descriptor in COBOL; by array descriptor in BASIC
10. LUN buffer length ²	Omit
11. Input file size ²	
12. Warning routine address ²	
13 Comparison routine address ²	
14. Equal-key routine address ²	

²Optional

1. Error address

Specify a four-word buffer for this required parameter that will contain the SORT status code and any other information that can be returned to the calling program, such as the STS and STV codes for errors involving I/O. Your program must check the status code when control is returned. Otherwise, the results of subsequent SORT/MERGE calls may be undefined.

The SORT/MERGE error codes returned in the first word of the error buffer are as follows:

Zero = Success
 Positive number = Exception code or warning
 Negative number = Fatal error

2. Work area address

Specify the work area to be used by your sort operation, including the sort tree and any needed buffers for work files and I/O, in this required parameter.

Only as much memory is used for a sort operation as is necessary. The amount required varies greatly with the parameters of the sort. In general, the more memory provided the faster the sort. Excess memory is used for multiblocking the I/O read and write operations (multiblocking refers to the capability of RMS to to read or write more than one block of a file into the I/O buffer at a time). The following formulas can be used for rough calculations of the minimum memory needed for a particular sort operation. All sizes are in bytes and values in decimal.

Let INP = number of input files
 OUT = number of output files
 WRK = number of work files
 TAP = total number of input and output files on tape
 IDX = total number of indexed keys in all input and output files
 IKS = maximum size of all input and output indexed keys
 TBS = maximum tape block size of all input and output files on tape
 LRL = maximum length of all input records in all input files

The formula for a sort operation is as follows:

$$(100 * INP) + ((800 + LRL) * WRK) + ((800 + LRL) * OUT) + ((2 * IKS + 600) * KEY) + (TBS * TAP) + (4 * LRL) + 1000$$

Some sort operations may require more or less than the amounts given by the above formula. Where work area is at a premium, some fine tuning may be needed. Otherwise, as much work area as possible should be given to increase the performance of the operation.

In BASIC (SRTINB) or COBOL (SRTINC) programs, pass this parameter by descriptor.

3. **Work area length**

For supported languages other than BASIC or COBOL, you must specify a word containing the length of the work area in bytes. In BASIC or COBOL programs, you do not pass this parameter for calls to SRTINB and SRTINC, because you specified this length as part of the work area address descriptor.

4. **Command line buffer**

For this required parameter, specify a word that gives the address of the buffer containing the MCR command line for your sort operation. Chapter 2 describes the use of the MCR switches.

The command line in this buffer differs from the MCR line used at the command level only when you are not passing files either on input or output. For the record interface, specify only the MCR switches that define the sort operation (but not those that define the input or output files). When you use the record interface only for output, include the input file specifications and any switches that describe the input files. When you use the record interface only for input, include the output file specification and any switches that describe the output file. The following example shows a command line with the file interface used on output and the record interface used on input.

```
SRTLIS.DAT=/KE:068.4/FO:V:71/BK:3
```

For SRTINB and SRTINC, pass this parameter by descriptor.

If you are using the record interface on input, and if you do not specify any output file switches in your command line buffer, SORT provides the following output defaults:

- Noncontiguous, sequential file with variable-length records
- Maximum record size equal to the length of the longest input record, as specified in the LRL parameter to SRTINI
- Bucket size of one
- Retrieval window size (RSX-11M/M-PLUS) or cluster size (RSTS/E) of zero

If you are using the record interface on input, you can perform only a record sort process. However, you can specify any one of the four sort processes for the file interface on input in your command line buffer.

5. **Command line length**

For this required parameter, provide a word giving the length of the command line in bytes. For the following example, you would specify a command line length of 33 bytes:

```
SRTLIS.DAT=/KE:068.4/FO:V:71/BK:3
```

Do not use this parameter if you are using BASIC or COBOL and passing parameters by descriptor, since the command line length was included in the command line buffer descriptor.

6. Longest record length (LRL)

This parameter is required in the following instances.

- When you use the record interface on input
- When you have input files not on disk
- In any other instance where the input file LRL is not available

Provide a word giving the size of the longest record that will be released for sorting. If you do not specify the LRL, and an LRL is not available from RMS, SORT returns a fatal error status.

7. Specification file buffer

Use this optional parameter when you want to define specification file text in your program without using an external specification file. Specify a word that gives the address of the buffer containing your specification file text. Chapter 3 discusses how to specify instructions for a sort operation in a specification file. Note that you cannot use any comment characters (!) in the specification file text placed in an internal buffer.

For SRTINB and SRTINC, pass this parameter by descriptor.

As an alternative to using this parameter to pass specification file text to SORT, you can specify the /SF switch and the specification file name in the MCR command line buffer. However, if both methods of passing specification file information to SORT are present in the same call to SRTINI, SORT returns a fatal error status.

8. Specification file buffer length

If you pass the specification file buffer parameter, you must also pass the length of this buffer. Specify a word giving the length of the specification file text in bytes.

Do not use this parameter for SRTINB or SRTINC, since the length is included in the previous parameter passed by descriptor.

9. LUN (Logical Unit Number) buffer

SORT needs a logical unit (often called a channel in RSTS/E documentation) for each work file requested, for each input file, if any, and for the output file, if any. Use this optional parameter if the default LUN assignments are inadequate for your sort operation.

The default LUNs that SORT uses are determined when SORT is installed. Unless otherwise specified at installation time, the default LUNs are as follows:

1. LUN 2—for specification file
2. LUN 3—for output file
3. LUN 4—for 1st input file

After LUN 4, an additional LUN exists for each input file, followed by one LUN for each scratch file. (For example, if you have two input files and two scratch files, LUN 5 is for the second input file, and LUNs 6 and 7 are for the two scratch files.)

Specify a word giving the address of a buffer that contains a word for each LUN that SORT is to use. The LUNs passed need not be consecutive; however, if SORT needs more LUNs than are passed, it will number the additional LUNs consecutively from the last number passed.

For SRTINB, pass this parameter by array descriptor; for SRTINC, pass it by descriptor.

10. LUN buffer length

If you pass the LUN buffer parameter, you must also pass this parameter to specify a word giving the length of the LUN buffer in words. Do not use this parameter with SRTINB or SRTINC, since the information was passed by descriptor with the previous parameter.

11. Input file size

You can use this optional parameter to improve the efficiency of your particular sort operation by overriding the default resources allocated by SORT. By default, SORT estimates work file requirements as follows:

- Input file size for the file interface when the input file is on disk
- 1000 blocks for the file interface when the input file is not on disk
- 1000 blocks for the record interface

To use this parameter, specify a word containing the input file size in blocks.

12. Warning routine address

Use this optional parameter to declare a warning handler and override the default actions for warning situations. To use the parameter, specify the address of a warning condition handling routine that SORT is to call when a warning situation occurs. The warning handler routine should evaluate any warning and return a value to SORT that indicates whether the sort operation terminates or continues.

SORT calls this routine with 2 parameters passed by reference, the error buffer address and a return status code address. The error buffer is the 4-word error buffer that you specified in the first parameter passed to the SRTINI subroutine. The second parameter, a return status code address, is the address of a word in which you will place the return status code value: +1 for continuation and 0 for termination. Any value other than +1 or 0 causes abnormal termination of the sort process.

This routine is called with the global symbol SRTWRN. If your language requires that you write this routine as a separate subprogram, you must use the same global symbol for the entry point. When task building, you must specify the object module for this subprogram. See Section 4.6 for information about task building.

13. Comparison routine address

This optional parameter allows you to use your own comparison routine rather than the key-field comparisons that SORT provides. SORT calls this routine with 5 reference parameters:

- The address of the buffer containing the first record
- The length of the first record
- The address of the buffer containing the second record
- The length of the second record
- The status code return

The routine that you write must pass a parameter back to SORT using the following status code values:

- -1 if the first record collates before the second
- 0 if the records collate as equal
- +1 if the first record collates after the second

Any other value will cause abnormal termination of the sort process.

Do not call this routine if you give key-field specifications in the command line buffer or specification file text.

Use the global symbol SRTCMP to call this routine. If your language requires that you write this routine as a separate subprogram, you must use the same global symbol for the entry point. When task building, you must specify the object module for this subprogram. See Section 4.6 for information about the use of task building.

14. Equal-key-field routine address

For key fields that collate as equal, you can specify the address of an equal-key routine. Using an equal-key-field routine gives you control over record deletion, which you cannot achieve through the use of the /ND (NODUPLICATES) switch. However, you should not use this parameter if you specify the /ST (STABLE) or /ND switch in the command line. Note also that you can pass this parameter only if you are using a record sort process.

SORT calls the equal-key routine with 5 reference parameters:

- The address of the buffer containing the first record
- The length of the first record
- The address of the buffer containing the second record
- The length of the second record
- The status code return

The routine must pass a parameter back to SORT with one of the following status code values:

- 0 = delete both records
- 1 = keep the first record only
- 2 = keep the second record only
- 3 = keep both records

Any other value will cause abnormal termination of the sort process.

You can modify the records passed to this routine before returning the status value. For example, you may want to reformat the records or modify a nonkey field. Suppose you are sorting, by employee name, a file that contains all the pay checks issued for one year. If you need only the total amount paid to each employee, you can add one pay check amount into a second duplicate record and then delete the first record.

Call this routine with the global symbol SRTCLB. If your language requires that you write this routine as a separate subprogram, you must use the same global symbol for the entry point. When task building, you must specify the object module for this subprogram. See Section 4.6 for information about task building.

4.4.5 Passing Records to SORT

When you use either the record interface or a mixed-mode interface with the record interface on input, you must call `SRTRLS` in order to pass records to `SORT`. Call this subroutine once for each record to be sorted. For `BASIC`, use the subroutine `SRTRLB`; for `COBOL`, use `SRTRLC`. You must set up a record buffer in your program's data area that will be used to contain the records.

`SRTRLS` has three required parameters (two for `SRTRLB` and `SRTRLC`), as shown in Table 4-3. An explanation of each parameter follows the table.

Table 4-3: Parameters for `SRTRLS`, `SRTRLB`, and `SRTRLC`

Parameters	BASIC/COBOL Usage Information
1. Error address	
2. Record buffer	Pass by descriptor
3. Record length	Omit

1. Error address

The error address, a required parameter for the `SRTRLS` call, is the same as for the `SRTINI` subroutine, as described earlier in this chapter.

2. Record buffer

For the required record buffer parameter, provide a word giving the address of the buffer that contains the record to be sorted. For `BASIC` (`SRTRLB`) and `COBOL` (`SRTRLC`), you pass this parameter by descriptor.

3. Record length

For this parameter, which is required for all languages other than `BASIC` and `COBOL` (for which the information was passed by descriptor in the previous parameter), you specify a word that gives the length of the record to be sorted.

4.4.6 Returning Records to Your Program

When you use either the record interface or a mixed-mode interface with the record interface on output, you must call `SRTRTN` to return the sorted records to your program. Call this subroutine once for each record that is to be sorted. `SRTRTN` places the record in a record buffer that you set up in your program's data area, returning an end-of-file status (+1) in the first parameter if there are no more records. If your application program is written in `BASIC` and you are passing information by descriptor, use the subroutine call `SRTRTB`; if your program is in `COBOL` and you are passing information by descriptor, use the subroutine call `SRTRTC`.

Table 4-4 shows the parameters for the `SRTRTN` subroutine. An explanation of the individual parameters follows the table.

Table 4–4: Parameters for SRTRTN, SRTRTB, and SRTRTC

Parameters¹	BASIC/COBOL Usage Information
1. Error address	
2. Record buffer	Pass by descriptor
3. Record buffer length	Omit
4. Returned record length	
5. Record location	

¹Note that when you use SRTRTN, you must pass either a record buffer or a record location. When you pass a record buffer, you must also pass the length of the record buffer. If you use BASIC (SRTRTB) or COBOL (SRTRTC), pass the record buffer by descriptor and omit the record buffer length.

1. Error address

The error address is the same as for the SRTINI subroutine, as described earlier in this chapter.

2. Record buffer

For the record buffer parameter, provide a word that gives the address of the buffer that is to contain the returned record. For SRTRTB and SRTRTC, pass this parameter by descriptor.

If you do not pass this parameter, you must pass the record location parameter.

3. Record buffer length

Provide a word giving the length of the record buffer. If you use the SRTRTB (BASIC) or SRTRTC (COBOL) subroutine, do not include this parameter since the information will have been passed by descriptor in the previous parameter.

4. Returned record length

For this parameter, specify the address of a word that is to receive a value representing the length of the returned record.

5. Record location

Use this parameter if you want SORT to return the address of the returned record (in the SORT internal buffer) rather than move the returned record to a buffer in your program. You must specify either a record buffer and length or a record location.

4.4.7 Sorting Records

When you use either the file or mixed-mode interfaces, you must call the SRTSRT subroutine to sort the records. When you use the file interface on input, SRTSRT is the second subroutine that you call; it reads the input file or files and sorts the records. If you use the file interface on output and record interface on input, SRTSRT is the third subroutine that you call; it sorts the records and writes them to the output file. For BASIC, the corresponding subroutine is named SRTSRB; for COBOL, it is SRTSRC.

SRTSRT has one required parameter: error address. This parameter is the same as the error address parameter for SRTINI and each of the other subroutines discussed thus far.

4.4.8 Ending a Sort Operation

Call the SRTEND subroutine to end a sort operation; use SRTEND either at the end of a successful sorting operation or when the program encounters an error during a sorting operation. This subroutine closes files, cleans up sort work areas, and releases memory. For BASIC, the corresponding subroutine is named SRTENB; for COBOL, it is SRTENC.

If an error occurs during the sort operation, SORT automatically closes files, cleans up work areas, and releases memory.

SRTEND has one required parameter: error address. The error address parameter for SRTEND is the same as the error address parameter for each of the other subroutine calls discussed in this chapter.

If you are using SORT from your program more than once, you must use SRTEND once for each time that you use SRTINI. That is, you must issue a call to the SRTEND subroutine to end a sort operation before you issue a subsequent call to SRTINI to begin another sorting operation.

4.5 Calling the MERGE Subroutines

A program calls MERGE in the same way that it calls SORT. For a merge operation at the program level, you must provide MERGE with the number of input files, the file specifications (when using either file or mixed-mode interface), information about key fields, and an input routine (when using either record interface or mixed-mode with record interface on input).

As with SORT, you pass this information to MERGE by using subroutine parameters. After being called, each subroutine performs its function and returns control to your program. You must also pass the address of the first word in a 4-word error buffer to each of the subroutines.

MERGE returns a value to the error buffer to indicate the success or error status for each call that you issue. You can have your program test that value to determine success or failure.

MERGE subroutines have both required and optional parameters. Required parameters appear first in the argument list. Include all parameters in the order in which they are positioned in the argument list, separating them with commas. Null parameters are indicated when no value follows the comma in the parameter's position in the argument list. If your programming language does not permit null parameters, use a 0 or -1 to indicate them in the parameter's position in the argument list. You can end your argument list at any time after you have specified all the required parameters.

Table 4-5 shows the standard calls for record and file interfaces and briefly describes the function of each. Explanations of each of these subroutine calls follow the table.

Table 4–5: MERGE Subroutines

Subroutine	Function
File Interface	
MRGINI	Initializes merge operations by passing file names, key-field information, and merge options
MRGMRG	Reads the input file or files, merges the records, and writes the records to the output file
MRGEND	Performs cleanup functions, such as closing files and releasing memory
Record Interface	
MRGINI	Initializes merge operations by passing key-field information and merge options
MRGRTN	Calls input routine and returns one merged record to your program; must be called once for each record
MRGEND	Performs cleanup functions, such as closing files and releasing memory

4.5.1 File Interface

For a merging task using the file interface, the first step is to call the initialization subroutine MRGINI. (As with SORT, if you are using BASIC or COBOL and pass information by descriptor, you use different entry points to initialize a merge operation. The initializing subroutine for BASIC is MRGINB; for COBOL it is MRGINC.)

The first parameters passed to MRGINI define the address of an error buffer and set up work areas. Then you pass the address of a command line buffer, in which you specify your input and output file names and your instructions about keys and merge options. You can merge up to 10 input files; you always have one and only one output file.

You also indicate whether you want key comparisons to be done by MERGE or by your own key-comparison routine. You may want to provide your own comparison routine to handle special sorting requirements; for example, if you are using a data type not supported by MERGE.

The next step when using the file interface in your program is to call MRGMRG (MRGSRB for BASIC, MRGSRC for COBOL) to execute the sort operation and to direct the sorted records to the output file. Finally, call MRGEND (MRGENB for BASIC, MRGENC for COBOL) to end the sort operation and to release resources.

Your program may call the MRGEND subroutine at any time between calls to the other subroutines to abort a merge operation and to release all resources allocated to the sort or merge process. If a fatal error condition occurs, SORT automatically releases all allocated resources.

4.5.2 Record Interface

When you are using the record interface, first call MRGINI (MRGINB for BASIC, MRGINC for COBOL). As with the file interface, this subroutine sets up work areas and passes parameters that define key fields and merge options. When you use the record interface with MERGE, you must also provide the address of a user-defined input routine when you call MRGINI. This is explained in Section 4.5.4 in the discussion of MRGINI.

Next, call MRGRTN (MRGRTB for BASIC, or MRGRTC for COBOL) to return the merged records to your program. MRGRTN calls the input routine as needed. Unlike SORT, MERGE does not need to hold all the records before it can begin returning them in the desired order. The releasing, merging, and returning of records all take place in this phase of the merge. You must call the MRGRTN subroutine once for each record to be returned and pass a parameter that tells MERGE where to place the merged record.

After all the records have been returned, call the last subroutine, MRGEND (MRGENB for BASIC, MRGENC for COBOL), to release resources.

4.5.3 Mixed-Mode Interface

When you are using a mixed-mode interface, order the MERGE subroutine calls to match your output interface. If you use the file interface on output, use the calls MRGINI, MRGMRG, and MRGEND. If you use the record interface on output, use the MRGINI, MRGMRG, and MRGRTN subroutines once for each record, followed by the MRGEND subroutine to end the MERGE operation.

The following section describes only the parameters passed in the initialization subroutine that are unique to MERGE. Unless otherwise specified in the following sections, the parameters passed for calls to MERGE subroutines are identical to the parameters passed for calls to SORT subroutines.

4.5.4 Initializing the Merge Process

Regardless of the interface that you use (file, record, or mixed-mode), you must first call MRGINI to initialize the merge process. For BASIC, the corresponding subroutine is called MRGINB, and for COBOL it is called MRGINC. This subroutine passes parameters that provide the number of input files, the key specifications, and merge options.

When you call the MRGINI initializing subroutine, there are seven required parameters that you must include and nine optional parameters that you may include, depending on your requirements. The MRGINB (for BASIC) and MRGINC (for COBOL) subroutines each have five required parameters and seven optional parameters. Table 4-6 shows the required and optional parameters for MRGINI and usage information for MRGINB and MRGINC. Following the table is a discussion of those parameters that are different for MERGE than for SORT.

Table 4–6: Parameters for MRGINI, MRGINB, and MRGINC

Parameters	BASIC/COBOL Usage Information
1. Error address	
2. Work area address	Pass by descriptor
3. Work area length	Omit
4. Command line buffer	Pass by descriptor
5. Command line length	Omit
6. Longest record length ¹	
7. Merge order ²	
8. Specification file buffer ³	Pass by descriptor
9. Specification file buffer length ³	Omit
10. LUN buffer ³	Pass by descriptor in COBOL, by array descriptor in BASIC
11. LUN buffer length ³	Omit
12. Input file size ³	
13. Input routine address ^{2,3}	
14. Warning routine address ³	
15. Comparison routine address ³	
16. Equal-key routine address ³	

¹Required when longest record length (LRL) is unavailable
²Required for record interface, or in mixed-mode interface on input
³Optional

2. Work area address

You must specify the size of the work area to be used by your merge operation. To estimate, in bytes, the minimum size of this area, use the following algorithm:

Let INP = number of input files
 OUT = number of output files
 WRK = number of work files
 TAP = total number of input and output files on tape
 IDX = total number of indexed keys in all input and output files
 IKS = maximum size of all input and output indexed keys
 TBS = maximum tape block size of all input and output files on tape
 LRL = maximum length of all input records in all input files

The formula for a merge operation is as follows:

$$((800 + 2 * LRL) * INP) + ((800 + LRL) * OUT) + ((2 * IKS + 600) * KEY) + (TBS * TAP) + 1000$$

A particular merge operation may require more or less than the amounts given by the above formula. In situations where work area is at a premium, some fine tuning may be needed. Otherwise, as much work area as possible should be allowed to enhance the performance of the operation.

For BASIC (MRGINB) and COBOL (MRGINC), pass this parameter by descriptor.

7. Merge order

Specify this required parameter by providing a word that gives the number of input files. You can have up to 10 input files for a merging operation. MERGE ignores this value if you are using the file interface or mixed-mode interface with file interface on input.

13. Input routine address

This parameter is required when you use either the record interface or the mixed-mode interface with the record interface on input. In either of these cases, you must write an input routine that releases a record to the merge operation. Give the address of the routine that you created for this parameter. MRGINI and MRGRTN call this routine until all records have been passed.

Your routine must read (or construct) a record, place it in a record buffer, store its length in an output parameter, and then return control to MERGE. MERGE compares key fields and returns records in merged order until it has processed all records.

The input routine must accept four parameters and return a status value in one of them. Specify the following four-reference parameters, in order, in your input routine:

- The address of the buffer in which the record will be placed
- A word in which to place the length of the record read
- A word containing the file number from which to input a record (the first file is 1, the second 2, and so on)
- The status code return

The routine must pass a parameter back to MERGE with one of the following status code values:

- -1 for a fatal error; end the MERGE
- 0 for a successful read
- +1 when end-of-file status is reached

Any other value will cause abnormal termination of the merge process. When the input routine returns an end-of-file status, it means that there is no valid record in the buffer.

Call this routine with the global symbol MRGINP. If your language requires that you write this routine as a separate subprogram, you must use the same global symbol for the entry point. When task building, you must specify the object module for this subprogram. See Section 4.6 for information for further information about task building.

14. Warning routine address

Call this routine with the global symbol MRGWARN. Otherwise, this parameter is identical to the SORT warning routine address parameter.

15. Comparison routine address

Call this routine with the global symbol MRGCMP. Otherwise, this parameter is identical to the SORT comparison routine address parameter.

16. Equal-key routine address

Call this routine with the global symbol MRGCLB. Otherwise, this parameter is identical to the parameter for the SORT equal-key-field routine address.

4.5.5 Summary of SORT Subroutine Calls

Table 4–7, Table 4–8, Table 4–9, and Table 4–10 summarize the SORT subroutine calls.

Table 4–7: Summary of SORT Subroutine Calls for the File Interface

Subroutine	Parameters	BASIC/COBOL Usage Information
SRTINI	Error address	
SRTINB	Work area address	Pass by descriptor
SRTINC	Work area length	Omit
	Command line buffer	Pass by descriptor
	Command line length	Omit
	Longest record length ¹	
	Specification file buffer ²	Pass by descriptor
	Specification file buffer length ²	Omit
	LUN buffer ²	Pass by descriptor in COBOL; pass by array descriptor in BASIC
	LUN buffer length ²	Omit
	Input file size ²	
	Warning routine address ²	
	Comparison routine address ²	
	Equal-key routine address ²	
SRTSRT	Error address	
SRTSRB		
SRTSRC		
SRTEND	Error address	
SRTENB		
SRTENC		

¹Required if unavailable from the record
²Optional

Table 4–8: Summary of SORT Subroutine Calls for the Record Interface

Subroutine	Parameters	BASIC/COBOL Usage Information
SRTINI	Error address	
SRTINB	Work area address	Pass by descriptor
SRTINC	Work area length	Omit
	Command line buffer	Pass by descriptor
	Command line length	Omit
	Longest record length ¹	
	Specification file buffer ²	Pass by descriptor
	Specification file buffer length ²	Omit
	LUN buffer ²	Pass by descriptor in COBOL; pass by array descriptor in BASIC
	LUN buffer length ²	Omit
	Input file size ²	
	Warning routine address ²	
	Comparison routine address ²	
	Equal-key routine address ²	
SRTRLS	Error address	
SRTRLB	Record buffer	Pass by descriptor
SRTRLC	Record length	Omit
SRTRTN	Error address	
SRTRTB	Record buffer ³	Pass by descriptor
SRTRTC	Record buffer length ³	Omit
	Returned record length	
	Record location ³	
SRTEND	Error address	
SRTENB		
SRTENC		

¹Required if unavailable from record
²Optional
³Use either record buffer and record buffer length or record location

Table 4-9: Summary of SORT Subroutine Calls for File-Interface Input and Record-Interface Output

Subroutine	Parameters	BASIC/COBOL Usage Information
SRTINI	Error address	
SRTINB	Work area address	Pass by descriptor
SRTINC	Work area length	Omit
	Command line buffer	Pass by descriptor
	Command line length	Omit
	Longest record length ¹	
	Specification file buffer ²	Pass by descriptor
	Specification file buffer length ²	Omit
	LUN buffer ²	Pass by descriptor in COBOL; pass by array descriptor in BASIC
	LUN buffer length ²	Omit
	Input file size ²	
	Warning routine address ²	
	Comparison routine address ²	
	Equal-key routine address ²	
SRTSRT	Error address	
SRTSRB		
SRTSRC		
SRTRTN	Error address	
SRTRTB	Record buffer ³	Pass by descriptor
SRTRTC	Record buffer length ³	Omit
	Returned record length	
	Record location ³	
SRTEND	Error address	
SRTENB		
SRTENC		

¹Required if unavailable from record
²Optional
³Use either record buffer and record buffer length OR record location

Table 4–10: Summary of SORT Subroutine Calls for Record-Interface Input and File-Interface Output

Subroutine	Parameters	BASIC/COBOL Usage Information
SRTINI	Error address	
SRTINB	Work area address	Pass by descriptor
SRTINC	Work area length	Omit
	Command line buffer	Pass by descriptor
	Command line length	Omit
	Longest record length ¹	
	Specification file buffer ²	Pass by descriptor
	Specification file buffer length ²	Omit
	LUN buffer ²	Pass by descriptor in COBOL; pass by array descriptor in BASIC
	LUN buffer length ²	Omit
	Input file size ²	
	Warning routine address ²	
	Comparison routine address ²	
	Equal-key routine address ²	
SRTRLS	Error address	
SRTRLB	Record buffer	Pass by descriptor
SRTRLC	Record length	Omit
SRTSRT	Error address	
SRTSRB		
SRTSRC		
SRTEND	Error address	
SRTENB		
SRTENC		

¹Required if unavailable from record
²Optional

4.6 Task Building

Because the callable SORT/MERGE subroutines are overlaid, you must create an overlay descriptor language (ODL) file in order to use the subroutines in your program. This section describes the use of ODL files, logical units, and user-defined routines for task building. For general information about task building and ODL files, consult the documentation for your operating system.

4.6.1 Overlay Descriptor Language Files

SORT/MERGE provides seven overlay descriptor language (ODL) files. When you use SORT/MERGE in a program, you must create your own ODL file that references one of the SORT/MERGE ODL files. Reference a SORT/MERGE ODL file by using the @ symbol followed by the SORT/MERGE ODL file specification in the ODL file that you create. Of the seven ODL files that SORT/MERGE provides, the one that you reference depends on the following factors:

- The operation that you are performing: sort, merge, or a combination of both
- The interface: file, record, or mixed-mode

- Use of the SORT/MERGE resident library: whether or not you are using or not using this library

Table 4-11 lists the SORT/MERGE ODL files, their use, and the approximate amount of memory that they require.

Table 4-11: SORT/MERGE ODL Files

Operation	Interface	File Name	Memory Requirements (in words)
SORT	File or mixed-mode	SRTFIL.ODL	8750
SORT	Record	SRTREC.ODL	7750
MERGE	File or mixed-mode	MGEFIL.ODL	8600
MERGE	Record	MGEREC.ODL	7500
Combined SORT/MERGE (SRTMRG.005)	File or mixed-mode	STMGFL.ODL	10,250
Combined SORT/MERGE (SRTMRG.006)	Record	STMGRC.ODL	9100
Combined SORT/MERGE Resident Library	File, record, or mixed-mode	SMSHR.ODL	8350

Note that the file or mixed-mode interface requires RMS11X.ODL. The record interface requires only sequential I/O and can use RMS11S.ODL.

On RSTS/E systems, the SORT/MERGE ODL files reside in LB:. On RSX-11M and RSX-11M-PLUS systems, the ODL files reside in LB:[1,1]. You reference the SORT/MERGE ODL file from the ODL file that you create.

Whichever ODL file you use, the .ROOT statement in your ODL file must refer to the SORT or MERGE root portion and co-trees. You must concatenate SMROT (code for SORT/MERGE nonoverlaid portion) with your root segments and make SMOVR (code for SORT/MERGE overlaid portion) and STMGIN (code for SORT/MERGE callable interface routines) co-trees with RMS.

The following example shows an ODL file created for a FORTRAN program that uses the file interface to SORT on an RSX system. The program name is YEARLY.FOR; the object file is YEARLY.OBJ. Any line that begins with a semicolon (;) is a comment line.

```

;           ODL File YEARLY.ODL
;
;           reference program YEARLY:
;
YEARLY:    .FCTR  YEARLY.OBJ
;
;           reference FORTRAN OTS:
;
F77OTS:    .FCTR  LB:[1,1]F4POTS/LB
;
;           reference the appropriate SORT ODL:
;
@LB:[1,1]SRTFIL.ODL
;
;           reference the RMS ODL:
;
@LB:[1,1]RMS11X.ODL
;
;           combine all the parts:
;
;           .NAME  TSTF77
M$PROG:    .FCTR  YEARLY-F77OTS
M$ROOT     .FCTR  TSTF77-M$PROG-RMSROT-SMROT
;
;           .ROOT  M$ROOT,RMSALL,SMOVR,STMGIN
;           .END

```

NOTE

If you use the COBOL-81 SORT/MERGE syntax, you automatically create the ODL file when you use the COBOL-81 Build ODL Utility, and no further work with ODL files is required. However, if a COBOL-81 program calls SORT or MERGE directly using the CALL verb (rather than using the embedded SORT/MERGE syntax), the procedures described above for creating an ODL file must be followed.

Use the file SMSHR.ODL when task building with the SORT/MERGE resident library, which is optionally created when SORT/MERGE is installed on your system. (Check with your system manager to ensure that SMSHR.ODL is installed on your system.) The SORT/MERGE resident library is called SMRES, and it is designed to be used in a cluster with RMS and your programming language. In order to cluster SORT/MERGE, SMRES must appear in your Task Builder CLSTR option. For example, in order to cluster SMRES with RMS and the COBOL-81 resident library, you would pass the following option to the Task Builder:

```
CLSTR = C81LIB,RMSRES,SMRES:RO
```

4.6.2 Task Building with User-Defined Routines

SORT/MERGE allows you to use four user-defined routines in your application program:

- Key comparison routine
- Equal key callback routine
- Warning handler routine
- MERGE input routine (not available for SORT)

In most cases, indicate that you want to use one of these routines by passing SORT/MERGE the address of the routine as one of the optional parameters to the SRTINI (SRTINB, SRTINC) or MRGINI (MRGINB, MRGINC) calls. However, if your programming language does not allow you to pass this address, you can still use your own routines by making a copy of the SORT/MERGE ODL file that you will use and then altering the appropriate line in the ODL file.

CAUTION

Before altering the SORT/MERGE ODL file, ensure that the file you intend to change is your local copy and not the ODL file in the system library.

The following chart shows the line of the ODL file that you should edit in order to use your own routine. In each instance, replace only the information that follows '.FCTR' with the name of the OBJ file of your subroutine.

- To use your own SORT or MERGE key-comparison routine, edit the following lines:

```
SCMP:      .FCTR          LB: [1, 1] SRTLIB/LB:$VCCMP
MCMP:      .FCTR          LB: [1, 1] MGELIB/LB:$VCCMP
```

- To use your own SORT or MERGE equal key callback routine, edit the following lines:

```
SCLB:      .FCTR          LB: [1, 1] SRTLIB/LB:$VCCLB:CALLBK
MCLB:      .FCTR          LB: [1, 1] MGELIB/LB:$VCCLB:CALLBK
```

- To use your own SORT or MERGE warning routine, edit the following lines:

```
SWRN:      .FCTR          LB: [1, 1] SRTLIB/LB:$VCWRN:SRTWRN
MWRN:      .FCTR          LB: [1, 1] MGELIB/LB:$VCWRN:SRTWRN
```

- To use your own MERGE input routine, edit the following line:

```
MINP:      .FCTR          LB: [1, 1] MGESHR/LB:$VCINP:MRGINP
```

Note that you must edit this line to reference your own merge input routine if you are using the record interface to MERGE (that is, using either MGEREC.ODL or STMGRC.ODL)

NOTE

If you use the SORT/MERGE resident library and task build against the file SMSHR.ODL, then the ODL file will read SRTSHR or MGESHR rather than SRTLIB or MGELIB. For example, the line for a SORT key-comparison routine using the resident library is as follows:

```
SCMP:      .FCTR LB: [1, 1] SRTSHR/LB:$VCCMP
```

For example, suppose you write a program in COBOL that calls SORT directly (using the record interface), you do not use the ANSI COBOL syntax, and you want to use your own equal-key-field callback routine. The name of the object file for your equal-key-field callback routine is EQUAL.OBJ.

Make a copy of SRTREC.ODL, and then modify the following line:

```
SCLB:      .FCTR LB: [1, 1] SRTLIB/LB:$VCCLB:CALLBK
```

to read as follows:

```
SCLB:      .FCTR EQUAL.OBJ
```


If you use both SORT and MERGE in the same task (therefore using either STMGFL.ODL or STMGRC.ODL) and want to use your own routine in both SORT and MERGE, you must change both the SORT and MERGE lines in the ODL file, as follows:

```
SCLB:      .FCTR          EQUAL.OBJ
MCLB:      .FCTR          EQUAL.OBJ
```

The symbols MINP, SCLB, MCLB, SWRN, MWRN, SCMP, and MCMP are referenced later in the ODL files, so be sure that you do not change any of these when modifying the ODL file.

4.6.3 Usage of Logical Unit Numbers

SORT requires the following number of logical unit numbers (LUNs):

- One LUN for a specification file (if used)
- One LUN for the output file
- One LUN for each input file
- One LUN for each work file

MERGE requires the same number of LUNs as SORT, except that MERGE does not use work file LUNs.

Customizing SORT

PDP-11 SORT/MERGE is designed for an environment of random-access disks, fairly large files, and medium-size records. PDP-11 SORT/MERGE automatically provides an efficient sort or merge operation for the data types listed in Chapter 2.

Rather than use the defaults provided by PDP-11 SORT/MERGE, you can design your ordering routine to work at maximum efficiency in the environment in which it is likely to be used most frequently. The environment includes such elements as input/output (I/O) devices, key-field data types, file sizes, and key-field and record sizes.

This chapter is intended to help you understand the factors to consider when fine-tuning your ordering operations. It provides a brief description of how SORT operates internally; explains the meaning and use of SORT/MERGE statistics; and suggests procedures for tailoring ordering operations to your environment.

5.1 SORT and MERGE Internal Operation

This section summarizes SORT's internal operation. It emphasizes the internal sorting data structure and work files, because you can modify these for greater efficiency.

You can use SORT in either of two ways:

- As a utility program that you invoke with a DCL or MCR command
- As a package of subroutines that you call from a program

The utility uses the subroutines to perform a sort operation. Almost no difference exists in the way that the utility and a user-written program operate. The following sections describe the phases of a sort operation.

5.1.1 Initialization Phase

During the initialization phase, SORT performs the following tasks:

1. Interprets the command line
2. Interprets the specification file if one is in use
3. Opens all the input files to determine longest record length, total input allocation, and file format and organization
4. Opens the output file
5. Divides the memory between SORT or MERGE data structures and I/O work area

6. Initializes the SORT or MERGE data structures
7. Creates and opens the specified number of work files

SORT uses a replacement selection algorithm to create ordered strings (or runs), which are then merged using a polyphase merge algorithm. The replacement selection algorithm uses a tree structure consisting of nodes of information about the records being sorted. The internal node size varies depending upon the type of sort operation being performed. Node size is discussed further in Section 5.2.

First, certain data structures containing information about key fields, record formats, collating sequences, and so on, are set up in the data structure portion of the work area; then the remaining area is given to the tree. The number of nodes in the tree is determined by dividing this area by the internal node size. In most cases, the more nodes in the tree, the faster the sort operation.

The I/O area is set up with all the required RMS data structures, as well as some necessary SORT I/O-related data structures. Then the remaining I/O area is divided into blocks and dynamically assigned as buffers for multiblock read and write operations.

Before the polyphase merge phase begins, a second initialization phase occurs in which the work area is redistributed. Since the merge normally involves significantly less work area for the data structure (there are fewer nodes), a greater proportion of the work area is allocated for I/O use. This allows larger multiblock counts during the merge phase.

The MERGE Utility (not to be confused with the merge phase of SORT) uses a straight n -way merging algorithm to merge the n input files. This requires n nodes plus one or two extra nodes for certain kinds of processing during the merge operation. Since fewer merge nodes exist than sort nodes, less space is needed for the merge data structures. Therefore, the proportion of memory given for I/O area is larger, and more multiblocking (RMS read or writes more than one block of a file into the I/O buffer at a time) occurs for I/O read and write operations.

5.1.2 Sort Phase

After the initialization phase, SORT reads the input records (or has the records released to it), converts them to the internal format, and places them in the sort data structure, calling the key-field-comparison and equal-key-field routines as needed. This continues until the sort data structure is full or all records have been read. If all records fit into the sort data structure, they are sorted in memory, and the work files that were created are not used.

If the sort data structure becomes full, SORT selects the record with the smallest value for a given key field from the sort data structure (or the largest value if you specified descending sort order), and writes it to a work file. This frees space for the next record to be read. SORT then reads another record into the sort data structure. Again, it selects the record with the smallest value for a given key field (but not smaller than that of the record just written to the work file), and writes it to the work file.

This process continues, producing a string of records that are in sequence (called a run), either until all the records have been read or there is no record in the data structure with a key field that is larger than the previous record written to the work file. If there is such a record, SORT begins building a second run, again first selecting the record with the smallest value for a given key field from those in the sort data structure, and continues reading records and writing them to the work files.

The runs thus produced are distributed among $n-1$ of the n work files in such a way that the number of runs in the work files approximates a generalized Fibonacci number. (A Fibonacci number is an integer in the infinite sequence 1,1,2,3,5,8,13, . . . of which the first two terms are 1 and 1 and each succeeding term is the sum of the two immediately preceding.) Since the number of runs produced depends on the data, there may be some instances when not all the work files are used even though there are many input records, and there may be some instances when all of the work files are used with a relatively small number of records. (For a discussion of generalized Fibonacci numbers, as well as the replacement selection and polyphase merge algorithms, see Donald Knuth's *Sorting and Searching*, Volume 3 in his multivolume set *The Art of Computer Programming*.) Dummy runs (containing no records) are assumed to exist in work files as needed so that the number of runs in the work files exactly equals a generalized Fibonacci number. The distribution is carried out in such a way as to minimize the number of dummy runs used. After all the records have been read, the sort data structure is emptied to the work files as one or two final runs. In general, the higher the Fibonacci level reached, the more time the polyphase merge phase will require. The more work files available, the lower the resulting Fibonacci level.

Therefore, it would appear to be best to use the maximum number of work files. A tradeoff exists, however, since the more work files there are in use, the less I/O multiblocking space is available for each work file, thus increasing the time required for work-file read and write operations.

The default number of work files that SORT/MERGE provides is 5, which produce the best performance for most operations. You can change this default when you install SORT/MERGE, but Digital recommends that you use caution when changing the default. For any SORT operation, you can change the number of work files, by using the /WORK_FILES qualifier (/FI:n switch), to a number between 3 and 10. For more information on work files, see Section 5.3.1.

5.1.3 Merge Phase

If the Fibonacci level reached is greater than 1 (that is, at least one work file contains more than one run), then SORT merges runs from $n-1$ work files to an empty work file. Whenever a work file becomes depleted of runs (including dummy runs), the next lower Fibonacci level is reached. That work file then receives the merged output from the other ($n-1$) work files. When Fibonacci level 1 is reached, there is one run in each work file. At that point, SORT either merges the $n-1$ work files with runs in them to the output file or returns them to the calling program.

5.1.4 Cleanup Phase

After the last record is written, SORT closes the input and output files, and then closes and deletes the work files. If you used the /STATISTICS qualifier (/SS switch), the utility displays the SORT statistics after the last record has been written and all files have been closed.

5.2 Understanding and Using SORT/MERGE Statistics

Using the /STATISTICS qualifier (/SS switch) causes SORT/MERGE to display statistics on your output device. You can also have statistics returned from Callable SORT or MERGE.

The following is the statistics display that the SORT/MERGE Utility returns:

```
PDP-11 SORT/MERGE V3.1
Elapsed time ( hh:mm:ss.ss ): 00:00:07.86
Process: record sort
Collating sequence: ASCII
Input files: 1                      Total input file allocation: 2
Work files: 0                      Total work file allocation: 0
Number of records output: 10       Final output file allocation: 2
Number of records input: 10       Longest input record found: 70
Number of records omitted: 0      Node size: 83
Number of keys: 1                 Number of nodes: 226
Total key size: 4                 Initial I/O area size: 15390
Number of initial runs: 0         Fibonacci level: 0
```

You can use these statistics to evaluate the efficiency of your ordering operation and to determine adjustments that could improve its performance. The statistics also include information about the sort or merge operation to help you determine if it proceeded as you had intended. The statistics include the following information:

- **Identification** shows the version and maintenance release numbers of SORT/MERGE. For example, V3.1 is version 3, maintenance release 1.
- **Elapsed time** is the clock time from the beginning of the initialization phase to just before the statistics are output. Note that this is not CPU time.
- **Process** is the process type (record, tag, index, address, or merge), whether it was stable or nonstable, whether or not duplicate records were allowed, and whether the sort was external (that is, required work files) or internal.
- **Collating sequence** is the collating sequence used: ASCII, EBCDIC, multinational, or user-defined. This statistic also indicates whether or not the basic collating sequence was modified (using a specification file).
- **Input files** shows the number of input files that were sorted or merged.
- **Work files (SORT only)** is the number of work files that were used. If a work file was opened but data was not written to it, the work file is still counted. All work files are opened at the same time.
- **Number of records output** is the number of records written to the output file or returned to the calling program.
- **Number of records input** is the number of records read from the input files or passed to the callable subroutines.
- **Number of records omitted** is the number of records omitted from the sort or merge operations. Records are omitted because of an /OMIT qualifier in a specification file, a /NODUPLICATES qualifier (or /ND switch), or because an equal-key-field callback routine returned a request that deleted a record.

- **Number of keys** is the number of key fields used in the sort or merge operation.
- **Total key size** is the maximum length of the key fields of the record formats in the sort or merge operation.
- **Total input allocation** is the total allocation of space (in blocks) for all input files.
- **Total work allocation (SORT only)** is the total allocation of space (in blocks) for all work files in a single operation. This is the final space allocation after all necessary extensions have been made during the sort.
- **Total output allocation** is the total space allocation (in blocks) for the output file. This is the final allocation after all necessary extensions have been made during the sort operation.
- **Longest input record found** is the length of the longest record in the input files, including omitted records. The longest record length (LRL) information kept by RMS on a file may not be accurate if records have been deleted from the file.
- **Node size** varies depending upon the type of sort operation being performed. For a record sort operation or a merge operation, the node size is approximately the sum of the following numbers:

The maximum longest record length for the input files
 Approximately 6 to 10 bytes of information about the record
 Approximately 8 bytes of pointers for the replacement selection and polyphase merge or merge algorithm

For a tag, index, or address sort, the node size is approximately the sum of the following numbers:

The total key-field size for the sort operation
 Approximately 12 to 16 bytes of information about the record
 Approximately 8 bytes of pointers for the replacement selection or polyphase merge algorithm

For large record size and small key-field size, the tag sort process will have a smaller internal node size than a record sort process. It will therefore have more tree nodes, fewer runs, and a lower Fibonacci level. The tag sort operation may still run slower, however, since it requires reaccessing the input file randomly to retrieve the output data from the input records.

- **Number of nodes** is the number of nodes in the replacement selection tree for a sort operation, or the number of nodes in the merge list for a merge operation. In a sort operation, if the number of nodes initially allocated to the tree is larger than the number of records being sorted, SORT uses only the smallest subtree necessary. The statistics nevertheless reflect the total number of nodes initially allocated to the tree.
- **Initial I/O area size** is the size (in bytes) of the area provided for I/O data structures and buffers. For a sort operation, this applies only to the sort distribution phase. Generally, more area is given for I/O during the polyphase merge phase.
- **Number of initial runs (SORT only)** is the number of ordered strings (runs) written to work files during the sort distribution phase.
- **Fibonacci level (SORT only)** is the Fibonacci level, as discussed in Section 5.1.2. If the sort operation was internal (that is, it required no work files), the Fibonacci level is zero.

- **Merge order (MERGE only)** is the number of input files.

5.2.1 Using Statistics with Callable SORT/MERGE

If you generate statistics when using SORT or MERGE from an application program, the statistical information is placed at the beginning of the work buffer that you specify. The following list shows the order in which the statistics are listed in the work buffer; the numbers in parentheses indicate the space (in words) that is allocated for each statistic.

1. SORT or MERGE version number
 - Binary number specifying major release number (1 word)
 - Binary number specifying update number (1 word)

(In Version 3.1, for example, the two words would contain 000011 and 000001)
2. Process Type (1 word)
 - 0 = Record sort
 - 1 = Tag sort
 - 2 = Address sort
 - 3 = Index sort
 - 4 = Merge
3. Collating sequence (1 word)
 - 0 = Unmodified ASCII
 - 1 = Unmodified EBCDIC
 - 2 = Unmodified Multinational
 - 3 = User-defined
 - 4 = Modified ASCII
 - 5 = Modified EBCDIC
 - 6 = Modified Multinational
4. Stable/nostable (1 word)
 - 0 = Nostable
 - 1 = Stable
5. Duplicates/noduplicates (1 word)
 - 0 = Noduplicates
 - 1 = Duplicates
6. Number of input files (1 word)
7. Total input file allocation (2 words)
8. Number of work files (1 word)
9. Total work file allocation (2 words)
10. Final output file allocation (2 words)
11. Size of sort tree node (1 word)
12. Number of nodes in SORT/MERGE data structure (1 word)
13. Size of I/O buffer area (1 word)
14. Number of input records (2 words)
15. Number of bytes in longest input record (1 word)
16. Number of records sorted or merged (2 words)

17. Number of records omitted during record selection (2 words)
18. Number of records output (2 words)
19. Number of key fields (1 word)
20. Total composite key-field size (1 word)
21. Number of initial runs produced by SORT (1 word)
22. Fibonacci level for sort or merge order for MERGE (1 word)
23. Elapsed clock time in hrs, mins, secs, 1/100 secs (4 words)

5.3 Modifications the User Can Make

After evaluating the variables in the environment for your sort or merge operation, consider the following possibilities for increasing SORT efficiency:

- Sorting fewer records: you can use a specification file to include only those records that you need to output.
- Sorting shorter records: you can also use a specification file to reformat records to eliminate fields that you do not need to output.

Additional strategies for increasing efficiency are as follows:

- Changing the number or assignment of work files
- Specifying input file allocation
- Adjusting output file allocation
- Changing the sort process
- Using the /TREE_SPACE qualifier or /PT switch

The following sections discuss these options in detail.

5.3.1 Work Files

Unless you specifically request that no work files be created, (using the /WORK_FILES=NUMBER:0 qualifier or the /FI:0 switch), SORT creates work files during the initialization phase to ensure that there will be sufficient disk space to perform the sort operation. By default, five work files are created, and this number provides the best performance for typical sort operations. If your available disks are too small or too full for SORT work files, you can increase the number of work files to make each work file smaller, and you can also assign the work files to different devices. However, the more work files SORT uses, the less I/O area each work file receives for multiblocking; thus the operation requires more time.

You can improve performance by using fewer work files. For example, if you know that the input file is almost in the desired order to begin with and will therefore produce only one or two long initial runs, the use of fewer work files is likely to improve the performance.

In addition to specifying the number of work files, you can also improve SORT efficiency by assigning the location of your work files to alternate random-access, mass-storage devices, such as disks. You can place work files on the fastest device available, the device having the least activity, or the least full device available. Use the /WORK_FILES=DEVICE qualifier or /DE switch (as indicated in Chapter 2 to select a different device for the work files.

When you use the `/WORK_FILES` qualifier (`/DE` switch) in a `SORT` command line, the work files are assigned to a single alternate device for the entire sort operation. If you use a specification file for `SORT`, you can assign individual work files to separate devices.

You can further increase `SORT` efficiency by specifying contiguous allocation for your work files with the `ALLOCATION` and `CONTIGUOUS` subqualifiers to the `/WORK_FILES` qualifier. The initial block allocation for each work file is derived as follows:

1. Multiply the estimated number of input records by the node size (provided in the statistics display).
2. Divide the product by 1 less than the number of work files.
3. Divide the resultant figure by 512 and round the result up to the next integer. This is the initial allocation, in blocks.

You can change this initial allocation of space to the work files, and you can request that this allocation be contiguous, if possible. However, you will receive no indication if it is necessary to extend a work file during the sort operation, making the space allocation noncontiguous.

You can also specify the `RSTS/E` file cluster size or the `RSX-11M/M-PLUS` retrieval window size for your work files using the `SIZE` subqualifier to the `/WORK_FILES` qualifier or the `/SI` input file switch. See your operating system documentation for more information on file cluster size and retrieval window size.

Chapter 2 shows the syntax for the subqualifiers to the `/WORK_FILES` qualifier and for the corresponding file switches `/DE`, `/AL`, `/CO`, and `/SI`. Note that the subqualifiers must be enclosed in parentheses and separated by commas.

5.3.2 Input File Allocation

`SORT` uses input file size information to determine the size of the work files. Usually, `RMS` determines the file size. However, if you are sorting files not residing on disk or standard ANSI magnetic tape and you do not provide the file size, `SORT` sets a default file size of 1000 blocks. The default for the record interface on input is also 1000 blocks.

If this space allocation is too large, `SORT` overestimates its memory and work file requirements; therefore, your sort operation is more efficient if you specify a smaller input file size. If the default of 1000 blocks is insufficient space, `SORT` underestimates its memory requirements and, conversely, your sort operation is more efficient if you specify a larger input file size.

Chapter 2 describes the use of the `FILE_SIZE` subqualifier to the `/FORMAT` qualifier and `/BK` switch to specify file size.

5.3.3 Output File Preallocation

`SORT/MERGE` preallocates space for your output file based on the total amount of space allocated to input files. This avoids the overhead of extending the file each time additional blocks are written to it.

However, if you know that your output file allocation will differ substantially from the total input file allocation (for example, because you are reformatting data or omitting records), you can specify the number of blocks to be preallocated for the output file using the `/ALLOCATION` output-file qualifier or the `/AL` output-file switch. See Chapter 2 for syntax information on this qualifier and switch.

By default, SORT/MERGE does not allocate the output file in contiguous blocks. You can request that the output file be stored in contiguous disk blocks, thereby decreasing access time, by using the /CONTIGUOUS qualifier or the /CO output file switch. However, if the preallocated space is too small, RMS may be unable to extend the file contiguously.

Two other output file options are available for fine-tuning your ordering operations: specifying the fill factor (/LOAD_FILL qualifier or /LO switch) and the bucket size (/BUCKET_SIZE=*n* qualifier or /BU:*n* switch).

You can specify the fill factor only for indexed-sequential files. RMS loads the buckets according to the fill size established when the file was created, minimizing bucket splitting if many records are added later.

If you use relative or indexed-sequential output, you can specify the bucket size to indicate RMS bucket size (that is, the number of 512-byte blocks per bucket). If the output file organization is the same as for the input files, the default value is the same as for the input file bucket size. If output file organization is different, the default value is 1. The maximum number of blocks per bucket is 32 for RSX-11M and RSX-11M/M-PLUS and 15 for RSTS/E.

Chapter 2 provides syntax information for the fill factor and bucket size options.

By default, SORT/MERGE divides available work area between tree-related data structures and input/output-related data structures in such a way as to ensure the best performance for a typical sort operation. However, in some instances the input/output (I/O) requirements of your job may require more space than the default provides. The /TREE_SPACE qualifier (in DCL) or /PT switch (in MCR) allows you to override this default and choose the distribution of available work area between SORT/MERGE data tree structures and I/O data structures.

For SORT, the default division is 55 percent to the tree and 45 percent to I/O. For MERGE, the default division is 30 percent to the merge list and 70 percent to I/O. If you use a consistently large number of input files, or if the majority of the files you are sorting (for example, an INDEXED file with many key fields) require a large number of I/O data structures, you may want to alter the ratio so that there will be enough room for the I/O requirements. For example, if you are sorting several indexed files, each having many key fields, it may be desirable to allow SORT a smaller tree, thereby allocating more room for RMS-required structures.

The /TREE_SPACE qualifier (/PT switch) allows you to override the default division of work-area space and to choose the distribution of available work area between SORT/MERGE data structures and I/O data structures.

In DCL, to allocate the work area, use the input file qualifier /TREE_SPACE=*n*, where *n* is the percentage of work space allocated to data tree structures.

In MCR use the input file switch /PT:*n*, where *n* is the percentage of work space allocated to data tree structures. Chapter 2 shows the syntax for the /TREE_SPACE qualifier and /PT switch.

5.3.4 Process

Although you usually select a sort process for reasons other than performance, there are differences in speed among the four sort processes. See Chapter 2 for a description of these differences. In any operation in which the sorted records are to be retrieved in order, record sort is usually the fastest sort process. If limited work space is available, or your records are very large relative to the total size of the key fields, consider using tag sort, which sorts only key fields and reaccesses

the input file to create the output file. Tag sort therefore requires less space than record sort.

5.4 Modifications the System Manager Can Make

The following are modifications the system manager can make for maximum SORT/MERGE performance:

- Designate one batch queue for sorting jobs and provide this queue with characteristics that improve system performance or SORT performance. In addition, job-process parameters can be adjusted for greatest SORT efficiency.
- Modify the ODL file used to compile the SORT/MERGE Utility to improve performance with regard to the type of data that you routinely sort or merge.
- Modify the default installation parameters. Although the default SORT/MERGE installation yields optimum performance for most applications, your special needs may require that different defaults be installed. See the *PDP-11 SORT/MERGE Installation Guide* for your operating system to install SORT/MERGE with different default parameters.

Error Messages

This appendix lists the error messages generated by the SORT and MERGE utilities. If you use SORT or MERGE from a DCL or MCR command line, error messages are displayed on your output device as an error code (for example, %SORT_F_EXTSRT) followed by a brief explanation of the error (for example, SORT requires work files).

The error messages listed are all for SORT. MERGE error messages are identical to SORT messages, except that they begin with %MERGE rather than %SORT.

If you use the callable SORT or MERGE subroutines from an application program, a numeric code for the error messages is placed in the first word of your error buffer. In this case, a positive number at the beginning of the word indicates a nonfatal exception or warning message, and a negative number indicates a fatal error message.

Table A-1 lists the error messages in the order of their error code. The numeric code shown with the error message is the number that is returned to the error message buffer.

Following the table are all of the SORT/MERGE error messages listed in alphabetical order.

Table A-1: SORT/MERGE Utility Error Messages

No. Code	Message Code	No. Code	Message Code	No. Code	Message Code
0	%SORT_W_SUCCESS	32	%SORT_F_XSLUNS	65	%SORT_F_MISPRM
1	%SORT_W_EOFEXC	33	%SORT_F_NUFRAB	66	%SORT_F_BADVAL
2	%SORT_W_BUFOVR	34	%SORT_F_NUFBUF	67	%SORT_F_INVSWH
3	%SORT_W_MRGORD	35	%SORT_F_EXTSRT	68	%SORT_F_MAXINP
4	%SORT_W_LCKBKT	36	%SORT_F_CRSF00	69	%SORT_F_MAXOUT
5	%SORT_W_WRTSHR	37	%SORT_F_CNSF00	70	%SORT_F_MISLRL
6	%SORT_W_SPCIVC	38	%SORT_F_WRSF00	71	%SORT_F_NOTMRG
7	%SORT_W_SPCIVD	39	%SORT_F_RDSF00	72	%SORT_F_NOTSRT
8	%SORT_W_SPCIVF	40	%SORT_F_DCSF00	73	%SORT_F_BADSEQ
9	%SORT_W_SPCIVI	41	%SORT_F_RWSF00	74	%SORT_F_ZMGORD
10	%SORT_W_SPCIVK	42	%SORT_F_CLSF00	75	%SORT_F_CHNPRS
11	%SORT_W_SPCIVP	43	%SORT_F_OPIF00	76	%SORT_F_CHNERR

(continued on next page)

Table A-1 (Cont.): SORT/MERGE Utility Error Messages

No. Code	Message Code	No. Code	Message Code	No. Code	Message Code
12	%SORT_W_SPCIVS	44	%SORT_F_CNIF00	77	%SORT_F_CHNFIL
13	%SORT_W_SPCIVX	45	%SORT_F_RDIF00	78	%SORT_F_ODADTR
14	%SORT_W_SPCMIS	46	%SORT_F_DCIF00	79	%SORT_F_MEMPRO
15	%SORT_W_SPCOVR	47	%SORT_F_CLIF00	80	%SORT_F_BPTBIT
16	%SORT_W_SPC SIS	48	%SORT_F_CROF00	81	%SORT_F_IOTTRP
17	%SORT_W_TRNREC	49	%SORT_F_OPOF00	82	%SORT_F_ILOPTR
18	%SORT_W_NUMTRN	50	%SORT_F_CN OF00	83	%SORT_F_EM TTRP
19	%SORT_W_LSTWRN	51	%SORT_F_WROF00	84	%SORT_F_TRPTRP
20	%SORT_F_BADFLD	52	%SORT_F_DCOF00	85	%SORT_F_FPTRAP
21	%SORT_F_MIXKEY	53	%SORT_F_CLOF00	90	%SORT_F_SPCADJ
22	%SORT_F_GCMBAD	54	%SORT_F_OPSP00	91	%SORT_F_SPCPLX
23	%SORT_F_MULSPC	55	%SORT_F_CN SP00	92	%SORT_F_SPCCHR
24	%SORT_F_NOIORM	56	%SORT_F_RDSP00	93	%SORT_F_SPCPAD
25	%SORT_F_ILCALL	57	%SORT_F_DCSP00	94	%SORT_F_SPCTHR
26	%SORT_F_WKAREA	58	%SORT_F_CLSP00	95	%SORT_F_INCNOKEY
27	%SORT_F_RLFAIL	60	%SORT_F_INTERR	96	%SORT_F_INCNODATA
28	%SORT_F_RSFAIL	61	%SORT_F_BADCMP	97	%SORT_F_WRTI00
29	%SORT_F_NSFRAB	62	%SORT_F_BADCLB	133	%SORT_F_BADORG
30	%SORT_F_NSFBUF	63	%SORT_F_BADINP	139	%SORT_F_LSTMSG
31	%SORT_F_NOSCBF	64	%SORT_F_NOMSG		

%SORT_F_BADCLB, Bad return from equal key callback routine

Callable returned error buffer: first word = -62.

Explanation: User-supplied equal key callback routine has returned illegal status.

%SORT_F_BADCMP, Bad return from comparison routine

Callable returned error buffer: first word = -61.

Explanation: User-supplied comparison routine has returned illegal status.

%SORT_F_BADFLD, Bad field in record number:

Callable returned error buffer:

first word = -20
second word = low word of record number
third word = high word of record number.

Explanation: Given record contains an invalid field. Record number displayed assumes all input files are concatenated.

%SORT_F_BADINP, Bad return from merge input routine

Callable returned error buffer: first word = -63.

Explanation: A user-supplied merge input routine has returned invalid status.

`%SORTFBADORG`, Organization of existing file incorrectly specified

Callable returned error buffer: first word = 133.

Explanation: You specified a file organization in the command line that did not match the actual organization of an existing input or output file. For output files, this can occur only if you are attempting to overlay an existing output file. For input files, this can occur if you specify as indexed-sequential a file that is not index-sequential, or if you fail to specify an index-sequential file as index sequential.

`%SORT_F_BADSEQ`, Bad sequence in input file

Callable returned error buffer:

first word = -73
second word = input file number

Explanation: Given input file is out of sequence for merge operation.

`%SORT_F_BADVAL`, Bad switch value

Callable returned error buffer:

first word = -66
second word = offset into command line of bad switch

Explanation: Command line has an invalid switch value.

`%SORT_F_BPTBIT`, Breakpoint or T-bit exception

Callable returned error buffer: No return to callable.

Explanation: Register dump will follow. Please submit it with an SPR.

`%SORT_W_BUFOVR`,

Callable returned error buffer:

first word = 2
second word = length of output record
third word = buffer size

Explanation: A user-supplied buffer was not big enough to hold a returned record. Normally just a warning. The record is truncated. If this message occurs in utility SORT or MERGE, please submit an SPR.

`%SORT_F_CHNERR`, Failed to chain: .RUN return status = (RSTS/E only)

Callable returned error buffer:

first word = -76
second word = .RUN directive return status

Explanation: SORT or MERGE was unable to chain to requested task.

`%SORT_F_CHNFIL`, Bad chain file specification: XRB flag2/flag1: (RSTS/E only)

Callable returned error buffer:

first word = -77
second word = XRB flag 2
third word = XRB flag 1

Explanation: You specified an invalid chain file.

%SORT_F_CHNPRS, Error parsing chain file: .FSS return status = (RSTS/E only)

Callable returned error buffer:

first word = -75
second word = .FSS directive return status

Explanation: Unable to parse chain file specification.

%SORT_F_CLIF00, Error closing input file: RMS codes

Callable returned error buffer:

first word = -47
second word = RMS STS code
third word = RMS STV code
fourth word = input file number

Explanation: RMS could not close the given input file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status value field (STV) codes.

%SORT_F_CLOF00, Error closing output file: RMS codes

Callable returned error buffer:

first word = -53
second word = RMS STS code
third word = RMS STV code

Explanation: RMS could not close the output file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

%SORT_F_CLSF00, Error closing work file: RMS codes

Callable returned error buffer:

first word = -42
second word = RMS STS code
third word = RMS STV code
fourth word = work file number

Explanation: RMS could not close the given work file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status value field (STV) codes.

%SORT_F_CLSP00, Error closing specification file: RMS codes

Callable returned error buffer:

first word = -58
second word = RMS STS code
third word = RMS STV code

Explanation: RMS could not close the specification file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

`%SORT_F_CNIF00`, Error connecting to input file: RMS codes

Callable returned error buffer:

first word = -44
second word = RMS STS code
third word = RMS STV code
fourth word = input file number

Explanation: RMS could not connect to the given input file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

`%SORT_F_CNOF00`, Error connecting to output file: RMS codes

Callable returned error buffer:

first word = -50
second word = RMS STS code
third word = RMS STV code

Explanation: RMS could not connect to the output file. See the *RMS-11 MACRO Programmer's Manual* for information on STS and STV codes.

`%SORT_F_CNFS00`, Error connecting to work file: RMS codes

Callable returned error buffer:

first word = -37
second word = RMS STS code
third word = RMS STV code
fourth word = work file number

Explanation: RMS could not connect to the given work file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

`%SORT_F_CNPS00`, Error connecting to specification file: RMS codes

Callable returned error buffer:

first word = -55
second word = RMS STS code
third word = RMS STV code

Explanation: RMS could not connect to the given specification file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

`%SORT_F_CROF00`, Error creating output file: RMS codes

Callable returned error buffer:

first word = -48
second word = RMS STS code
third word = RMS STV code

Explanation: RMS could not create the output file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

%SORT_F_CRSF00, Error creating work file: RMS codes

Callable returned error buffer:

first word = -36
second word = RMS STS code
third word = RMS STV code
fourth word = work file number

Explanation: RMS could not create the given work file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

%SORT_F_DCIF00, Error disconnecting from input file: RMS codes

Callable returned error buffer:

first word = -46
second word = RMS STS code
third word = RMS STV code
fourth word = input file number

Explanation: RMS could not disconnect from the given input file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

%SORT_F_DCOF00, Error disconnecting from output file: RMS codes

Callable returned error buffer:

first word = -52
second word = RMS STS code
third word = RMS STV code

Explanation: RMS could not disconnect from the given output file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

%SORT_F_DCSF00, Error disconnecting from work file: RMS codes

Callable returned error buffer:

first word = -40
second word = RMS STS code
third word = RMS STV code
fourth word = work file number

Explanation: RMS could not disconnect from the given work file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

%SORT_F_DCSP00, Error disconnecting from specification file: RMS codes

Callable returned error buffer:

first word = -57
second word = RMS STS code
third word = RMS STV code

Explanation: RMS could not disconnect from the given specification file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

%SORT_F_EMTRP, Non-RSX EMT trap

Callable returned error buffer: No return to callable.

Explanation: Register dump will follow. Please submit with an SPR.

%SORT_W_EOFEXC,

Callable returned error buffer: first word = 1.

Explanation: End of file returned from Callable interface. If this message occurs in utility SORT or MERGE, please submit an SPR.

%SORT_F_EXTSRT, SORT requires work files: RMS codes

Callable returned error buffer:

first word = -35
second word = RMS STS code
third word = RMS STV code
fourth word = input file number

Explanation: You did not indicate work files, but the sort operation could not be done internally. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

%SORT_F_FPTRAP, Floating-point exception

Callable returned error buffer: No return to callable.

Explanation: Register dump will follow. Please submit with an SPR.

%SORT_F_GCMBAD, Cannot get command line: GCML error

Callable returned error buffer:

first word = -22
second word = GCML error code

Explanation: RMS was unable to get a command line. The GCML error codes are as follows:

- 1 I/O error occurred during command line input.
- 2 Unable to open command file; make sure that command file name is correct and exists
- 3 Syntax error in command file name
- 4 Command file nesting level exceeded
- 5 Command line in file is too long. (Use hyphen continuation character (-) to divide line into smaller units.)
- 40 Command line input buffer too small for total command line; shorten your command line

%SORT_F_ILCALL, Illegal calling sequence: state

Callable returned error buffer:

first word = -25
second word = SORT or MERGE internal state code

Explanation: You called sort or merge subroutines in incorrect order. If this message occurs in utility SORT or MERGE, please submit an SPR.

%SORT_F_ILOPTR, Illegal instruction trap

Callable returned error buffer: No return to callable.

Explanation: Register dump will follow. Please submit with an SPR.

%SORT_F_INCNODATA, INCLUDE specification references no data

Callable returned error buffer:

first word = -96

second word = specification file line number

Explanation: In specification file, INCLUDE specification has to contain a DATA clause.

%SORT_F_INCNOKEY, INCLUDE specification references no keys

Callable returned error buffer:

first word = -95

second word = specification line number

Explanation: In specification file, /INCLUDE specification has to contain a /KEY specification.

%SORT_F_INTERR, Internal SORT/MERGE error

Callable returned error buffer: first word = -60

Explanation: Please submit an SPR.

%SORT_F_INVSWH, Invalid or redundant switch

Callable returned error buffer:

first word = -67

second word = location (offset into command line) of bad switch

Explanation: Command line has invalid switch or two switches that should not be used together.

%SORT_F_IOTTRP, IOT trap

Callable returned error buffer: No return to callable.

Explanation: Register dump will follow. Please submit with an SPR.

%SORT_W_LCKBKT, Locked bucket in input file

Callable returned error buffer:

first word = 4

second word = input file number

Explanation: RMS attempted to read a bucket that was locked in the input file. SORT/MERGE will retry reading bucket the number of times specified at installation. If the retry fails, a read error is issued.

%SORT_F_LSTMSG,

Callable returned error buffer:

first word = -139

second word = message number

Explanation: Message code is too large; this is probably an internal error. Please submit an SPR.

%SORT_W_LSTWRN,

Callable returned error buffer:

first word = 19

second word = message number

Explanation: Warning message code is too large; this is probably an internal error. Please submit an SPR.

%SORT_F_MAXINP, Too many input files

Callable returned error buffer: first word = -68.

Explanation: Too many input files are specified in the command line.

%SORT_F_MAXOUT, Too many output files

Callable returned error buffer: first word = -69.

Explanation: Too many output files are specified in the command line.

%SORT_F_MEMPRO, Memory protect error

Callable returned error buffer: No return to callable.

Explanation: Register dump will follow. Please submit with an SPR.

%SORT_F_MISLRL, No LRL found for file

Callable returned error buffer:

first word = -70

second word = input file number

Explanation: The longest record length for each input file must be made known to SORT or MERGE either through RMS or by a /FORMAT qualifier (/FO switch)

%SORT_F_MISPRM, Missing required parameter

Callable returned error buffer: first word = -65.

Explanation: You did not pass a required parameter to callable subroutine.

%SORT_F_MIXKEY, Incompatible key comparison

Callable returned error buffer: first word = -21.

Explanation: SORT or MERGE attempted to compare two key fields that were not compatible. This should only occur when you use a specification file to specify multiple record formats.

%SORT_W_MRGORD,

Callable returned error buffer: first word = 3.

Explanation: You passed a merge order to callable SORT/MERGE in which the number of files specified did not equal the number of files specified in the passed MCR command line. The passed merge order will be ignored. If you are not using callable file or mixed file-to-record interface, please submit an SPR.

%SORT_F_MULSPC, Multiply defined specification file

Callable returned error buffer: first word = -23.

Explanation: You passed a specification file buffer as well as a command line containing a file specification for a specification file. If this message occurs in the utility SORT or MERGE, please submit an SPR.

%SORT_F_NOIORM, No room for I/O pool space

Callable returned error buffer: first word = -24.

Explanation: The I/O area provided for the SORT or MERGE was not big enough for the current operation. For callable SORT, pass a larger work area, or use the /PT switch to allocate more of the given work area for I/O use.

%SORT_F_NOMSG, Message number

Callable returned error buffer:

first word = -64
second word = bad message number

Explanation: This is an internal error; please submit an SPR.

%SORT_F_NOSCBF, Out of work file I/O buffer space

Callable returned error buffer: first word = -31.

Explanation: This is probably an internal error; please submit an SPR.

%SORT_F_NOTMRG, Non-MERGE switch

Callable returned error buffer:

first word = -71
second word = location of switch (offset into MCR command line)

Explanation: You specified a MERGE command line switch that is valid only for SORT.

%SORT_F_NOTSRT, Non-SORT switch

Callable returned error buffer:

first word = -72
second word = offset into MCR command line.

Explanation: You specified a SORT command line switch that is valid only for MERGE.

%SORT_F_NSFBUF, Out of work file buffer space

Callable returned error buffer: first word = -30.

Explanation: This is probably an internal error; please submit an SPR.

`%SORT_F_NSFRAB`, Out of work file RAB space

Callable returned error buffer: first word = -29.

Explanation: This is probably an internal error; please submit an SPR.

`%SORT_F_NUFBUF`, Out of user file buffer space

Callable returned error buffer: first word = -34.

Explanation: This is probably an internal error; please submit an SPR.

`%SORT_F_NUFRAB`, Out of user file RAB space

Callable returned error buffer: first word = -33.

Explanation: This is probably an internal error; please submit an SPR.

`%SORT_W_NUMTRN`, Number of records truncated:

Callable returned error buffer:

first word = 18

second word = low word of number of records truncated

third word = high word of number of records truncated

Explanation: See warning message `%SORT_W_TRNREC`.

`%SORT_F_ODADTR`, Odd address trap

Callable returned error buffer: No return to callable.

Explanation: Register dump will follow. Please submit with an SPR.

`%SORT_F_OPIF00`, Error opening input file: RMS codes

Callable returned error buffer:

first word = -43

second word = RMS STS code

third word = RMS STV code

fourth word = input file number

Explanation: RMS could not open the given input file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

`%SORT_F_OPOF00`, Error opening output file: RMS codes

Callable returned error buffer:

first word = -49

second word = RMS STS code

third word = RMS STV code

Explanation: RMS could not open the given output file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

%SORT_F_OPSP00, Error opening specification file: RMS codes

Callable returned error buffer:

first word = -54
second word = RMS STS code
third word = RMS STV code

Explanation: RMS could not open the given specification file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

%SORT_F_RDIF00, Error reading from input file: RMS codes

Callable returned error buffer:

first word = -45
second word = RMS STS code
third word = RMS STV code
fourth word = input file number

Explanation: RMS failed while trying to read the given input file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

%SORT_F_RDSF00, Error reading from work file: RMS codes

Callable returned error buffer:

first word = -39
second word = RMS STS code
third word = RMS STV code
fourth word = work file number

Explanation: RMS failed while trying to read the given work file.

%SORT_F_RDSP00, Error reading from specification file: RMS codes

Callable returned error buffer:

first word = -56
second word = RMS STS code
third word = RMS STV code

Explanation: RMS failed while trying to read the specification file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

%SORT_F_RLFAIL, Failure to release allocated pool block

Callable returned error buffer: first word = -27.

Explanation: This is probably an internal error; please submit an SPR.

%SORT_F_RSFAIL, Failure to allocate requested pool block

Callable returned error buffer: first word = -28.

Explanation: This is probably an internal error; please submit an SPR.

%SORT_F_RWSF00, Error rewinding work file: RMS codes

Callable returned error buffer:

first word = -41
second word = RMS STS code
third word = RMS STV code
fourth word = work file number

Explanation: RMS could not rewind the given work file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

%SORT_F_SPCADJ, Invalid collating sequence definition

Callable returned error buffer: first word = -90.

Explanation: Collating sequence in the specification file is not valid.

%SORT_F_SPCCHR, Invalid character definition

Callable returned error buffer: first word = -92.

Explanation: Character definition in the specification file is not valid.

%SORT_W_SPCIVC, Invalid collating sequence, on line

Callable returned error buffer:

first word = 6
second word = specification file line number

Explanation: Collating sequence in the specification file is not valid.

%SORT_W_SPCIVD, Invalid data type, on line

Callable returned error buffer:

first word = 7
second word = specification file line number

Explanation: Data type found in the specification file is not valid.

%SORT_W_SPCIVF, Invalid field, on line

Callable returned error buffer:

first word = 8
second word = specification file line number

Explanation: Field definition in the specification file is not valid.

%SORT_W_SPCIVI, Invalid include or omit, on line

Callable returned error buffer:

first word = 9
second word = specification file line number

Explanation: Include or omit definition in the specification file is not valid.

%SORT_W_SPCIVK, Invalid key or data, on line

Callable returned error buffer:

first word = 10
second word = specification file line number

Explanation: Key field definition in the specification file is not valid.

%SORT_W_SPCIVP, Invalid sort process, on line

Callable returned error buffer:

first word = 11
second word = specification file line number

Explanation: Sort process found in the specification file is not valid.

%SORT_W_SPCIVS, Invalid specification line

Callable returned error buffer:

first word = 12
second word = specification file line number

Explanation: The given line in the specification file contains an error.

%SORT_W_SPCIVX, Invalid condition, on line

Callable returned error buffer:

first word = 13
second word = specification file line number

Explanation: Condition definition in the specification file is not valid.

%SORT_W_SPCMIS, Invalid merge specification, on line

Callable returned error buffer:

first word = 14
second word = specification file line number

Explanation: Specification given in a MERGE specification file that is valid only for SORT.

%SORT_W_SPCOVR, Specification overridden, on line

Callable returned error buffer:

first word = 15
second word = specification file line number

Explanation: Specification has been overridden by command line or callable parameter.

%SORT_F_SPCPAD, Invalid pad character

Callable returned error buffer: first word = -93.

Explanation: Pad character definition in specification file is not valid.

%SORT_F_SPCPLX, Collating sequence too complex

Callable returned error buffer: first word = -91.

Explanation: The collating sequence in the specification file has too many collating values.

%SORT_W_SPC SIS, Invalid sort specification, on line

Callable returned error buffer:

first word = 16

second word = specification file line number

Explanation: You have specified a qualifier or switch in the SORT specification file that is valid only for MERGE.

%SORT_F_SPCTHR, Cannot define three-byte collating value

Callable returned error buffer: first word = -94.

Explanation: You attempted to define 3-byte collating value in a specification file.

%SORT_W_SUCCESS,

Callable returned error buffer: first word = 0.

Explanation: Success returned from callable interface. If this message occurs in utility SORT, please submit an SPR.

%SORT_W_TRNREC, Truncating records longer than specified LRL of

Callable returned error buffer:

first word = 17

second word = specified LRL

Explanation: At least one record in an input file was longer than the longest record length (LRL) you specified. All such records are truncated.

%SORT_F_TRPTRP, TRAP instruction execution

Callable returned error buffer: No return to callable.

Explanation: Register dump will follow. Please submit with an SPR.

%SORT_F_WKAREA, Insufficient work area (bytes):

Callable returned error buffer:

first word = -26

second word = number of bytes of work area supplied

Explanation: Work area supplied is insufficient for the SORT or MERGE operation.

%SORT_F_WROF00, Error writing to output file: RMS codes

Callable returned error buffer:

first word = -51
second word = RMS STS code
third word = RMS STV code

Explanation: RMS failed while trying to write to the given output file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

%SORT_F_WRSF00, Error writing to work file

Callable returned error buffer:

first word = -38
second word = RMS STS code
third word = RMS STV code
fourth word = work file number

Explanation: RMS failed while trying to write to the given work file. See the *RMS-11 MACRO Programmer's Manual* for information on status-code-field (STS) and status-value-field (STV) codes.

%SORT_F_WRTI00, Error writing to terminal device

Callable returned error buffer: first word = -97.

Explanation: RMS failed while trying to write to the terminal device.

%SORT_W_WRTSHR, Input file opened allowing writes—file number

Callable returned error buffer:

first word = 5
second word = input file number

Explanation: Warning that the input file could be modified during the SORT or MERGE operation.

%SORT_F_XSLUNS, Too many LUNs required (required/max allowed):

Callable returned error buffer:

first word = -32
second word = number of LUNs required
third word = maximum LUNs allowed

Explanation: The total number of logical unit numbers (LUNs) required (because of the number of input, output, work, and specification files) exceeds the maximum allowed on the system.

%SORT_F_ZMGORD, Invalid merge order

Callable returned error buffer: first word = -74.

Explanation: You passed a zero merge order to a callable MERGE record operation.

Appendix B

Sample Programs

This appendix includes sample application programs that demonstrate the use of the callable SORT and MERGE subroutines. The purpose of the sample programs is to show the subroutines in program source code; the programs have not necessarily been designed to demonstrate common applications, performance optimization, or programming practices.

This appendix includes six sample programs, as follows:

- A BASIC-PLUS-2 program using the MERGE file interface
- A BASIC-PLUS-2 program using both SORT and MERGE mixed-mode interfaces
- A COBOL-81 program using the MERGE record interface
- A COBOL-81 program using the SORT record interface
- A FORTRAN program using the MERGE file interface
- A FORTRAN program using the SORT file interface

The BASIC-PLUS-2 and COBOL programs pass arguments by descriptor, so they use the special subroutine names for BASIC and COBOL (for example, SRTINB and SRTINC). The FORTRAN programs use the standard subroutine names (for example, SRTINI).

B.1 BASIC-PLUS-2 Program Using the MERGE File Interface

```
1  EXTEND
10 DECLARE                                &
    INTEGER                               &
        lun_buf(0%),                      &
        err_buf(4%)
20 DECLARE                                &
    STRING                                 &
        command_line                       &
!
40 MAP( WORK )                            &
    STRING wrk_area = 15000 &
!
1910 command_line = "output=bmgfil/fo:v:80,bmgfll/fo:v:80/ke:col.1/ss" &
!
1920 lun_buf(0) = 6%                       &
!
1990 PRINT "calling MRGINB"
2000 CALL MRGINB (err_buf() BY REF,        &
                 wrk_area BY DESC,        &
                 command_line BY DESC,    &
                 0% BY VALUE,             &
                 0% BY VALUE,             &
                 0% BY VALUE,             &
                 lun_buf() BY DESC,       &
                 0% BY VALUE,             &
                 0% BY VALUE,             &
                 0% BY VALUE,             &
                 0% BY VALUE,             &
                 0% BY VALUE )            &
!
2100 IF err_buf(0) = 0 THEN GOTO 2300      &
!
2110 PRINT "error in MRGINB:  err_buf(0) = ",err_buf(0)
2120 PRINT "                  err_buf(1) = ",err_buf(1)
2130 PRINT "                  err_buf(2) = ",err_buf(2)
2140 PRINT "                  err_buf(3) = ",err_buf(3)
2200 GOTO 30000                            &
!
2300 PRINT "calling MRGMRB"
3000 CALL MRGMRB (err_buf() BY REF)        &
3100 IF err_buf(0) = 0 THEN GOTO 4990      &
!
3110 PRINT "error in MRGMRB:  err_buf(0) = ",err_buf(0)
3120 PRINT "                  err_buf(1) = ",err_buf(1)
3130 PRINT "                  err_buf(2) = ",err_buf(2)
3140 PRINT "                  err_buf(3) = ",err_buf(3)
3150 GOTO 30000                            &
!
4990 PRINT "calling MRGENB"
5000 CALL MRGENB (err_buf() BY REF)        &
!
5105 IF err_buf(0) = 0 THEN GOTO 30000     &
!
5110 PRINT "error in MRGENB:  err_buf(0) = ",err_buf(0)
5120 PRINT "                  err_buf(1) = ",err_buf(1)
5130 PRINT "                  err_buf(2) = ",err_buf(2)
5140 PRINT "                  err_buf(3) = ",err_buf(3) &
!
30000 END
```

B.2 BASIC-PLUS-2 Program Using Both SORT and MERGE Mixed-Mode Interfaces

```

1  EXTEND
10 DECLARE                                &
    INTEGER                                &
    lun_buf(0%),                          &
    err_buff(4%),                          &
    inp_lrl
20 DECLARE                                &
    STRING                                  &
    command_line                            &
!
25 MAP( DISK )                             &
    STRING rec_buf = 80                    &
!
40 MAP( WORK )                             &
    STRING wrk_area = 10000                &
!
1900 ON ERROR GOTO 30000                    &
    ! DEFINE THE ENVIRONMENT
1910 command_line = "temp1/al:6=/ke:col.1/pt:30/fi:3" &
!
1905 inp_lrl = 80%
1920 lun_buf(0) = 6%                        &
!
1990 PRINT "calling SRTINB"
2000 CALL SRTINB (err_buff() BY REF,        &
    wrk_area BY DESC,                      &
    command_line BY DESC,                  &
    inp_lrl BY REF,                        &
    0% BY VALUE,                           &
    lun_buf() BY DESC,                     &
    0% BY VALUE,                           &
    0% BY VALUE,                           &
    0% BY VALUE,                           &
    0% BY VALUE )                          &
!
2100 IF err_buff(0) = 0 THEN GOTO 2300      &
!
2110 PRINT "error in SRTINB:  err_buff(0) = ",err_buff(0)
2120 PRINT "                  err_buff(1) = ",err_buff(1)
2130 PRINT "                  err_buff(2) = ",err_buff(2)
2140 PRINT "                  err_buff(3) = ",err_buff(3)
2200 GOTO 30900                             &
!
2250 PRINT "open the input file"
2300 OPEN "BSMMIX.DAT" FOR INPUT AS FILE 1%, &
    ORGANIZATION SEQUENTIAL FIXED,        &
    MAP DISK,                              &
    ACCESS READ,                            &
    ALLOW NONE                              &
!
2800 GET #1                                  &
!
2990 PRINT "calling SRTRLB"
3000 CALL SRTRLB (err_buff() BY REF,        &
    rec_buf BY DESC)                       &
3100 IF err_buff(0) = 0 THEN GOTO 2800     &

```

```

!
3110 PRINT "error in SRTRLB:  err_buff(0) = ",err_buff(0)
3120 PRINT "                  err_buff(1) = ",err_buff(1)
3130 PRINT "                  err_buff(2) = ",err_buff(2)
3140 PRINT "                  err_buff(3) = ",err_buff(3)
3150 GOTO 30900 &
!
3300 PRINT "calling SRTSRB"
4000 CALL SRTSRB (err_buff() BY REF)
4100 IF err_buff(0) = 0 THEN GOTO 4990 &
!
4110 PRINT "error in SRTSRB:  err_buff(0) = ",err_buff(0)
4120 PRINT "                  err_buff(1) = ",err_buff(1)
4130 PRINT "                  err_buff(2) = ",err_buff(2)
4140 PRINT "                  err_buff(3) = ",err_buff(3)
4150 GOTO 30900 &
!
4990 PRINT "calling SRTENB"
5000 CALL SRTENB (err_buff() BY REF) &
!
5105 IF err_buff(0) = 0 THEN GOTO 5150 &
!
5110 PRINT "error in SRTENB:  err_buff(0) = ",err_buff(0)
5120 PRINT "                  err_buff(1) = ",err_buff(1)
5130 PRINT "                  err_buff(2) = ",err_buff(2)
5140 PRINT "                  err_buff(3) = ",err_buff(3) &
!
5150 command_line = "=BSMMIX/FO:V:80/ke:col.1/pt:30/fi:3" &
!
5155 CLOSE #1
5160 PRINT "calling SRTINB"
6000 CALL SRTINB (err_buff() BY REF, &
                wrk_area BY DESC, &
                command_line BY DESC, &
                inp_lrl BY REF, &
                0% BY VALUE, &
                lun_buf() BY DESC, &
                0% BY VALUE, &
                0% BY VALUE, &
                0% BY VALUE, &
                0% BY VALUE ) &
!
6100 IF err_buff(0) = 0 THEN GOTO 6300 &
!
6110 PRINT "error in SRTINB:  err_buff(0) = ",err_buff(0)
6120 PRINT "                  err_buff(1) = ",err_buff(1)
6130 PRINT "                  err_buff(2) = ",err_buff(2)
6140 PRINT "                  err_buff(3) = ",err_buff(3)
6200 GOTO 30900 &
!
6300 OPEN "TEMP2.DAT" FOR OUTPUT AS FILE 1%, &
        ORGANIZATION SEQUENTIAL FIXED, &
        MAP DISK, &
        ACCESS WRITE, &
        ALLOW NONE &

```



```

!
6400 PRINT "calling SRTSRB"
7000 CALL SRTSRB (err_buff() BY REF)
7100 IF err_buff(0) = 0 THEN GOTO 7200 &
!
7110 PRINT "error in SRTSRB:  err_buff(0) = ",err_buff(0)
7120 PRINT "                    err_buff(1) = ",err_buff(1)
7130 PRINT "                    err_buff(2) = ",err_buff(2)
7140 PRINT "                    err_buff(3) = ",err_buff(3)
7150 GOTO 30900 &
!
7200 CALL SRTRTB (err_buff() BY REF, &
                rec_buf BY DESC, &
                rtn_len BY REF) &
!
7300 IF err_buff(0) = 0 THEN GOTO 7400 &
!
7305 IF err_buff(0) = 1 THEN GOTO 7500 &
!
7310 PRINT "error in SRTRTB:  err_buff(0) = ",err_buff(0)
7320 PRINT "                    err_buff(1) = ",err_buff(1)
7330 PRINT "                    err_buff(2) = ",err_buff(2)
7340 PRINT "                    err_buff(3) = ",err_buff(3)
7350 GOTO 30900 &
!
7400 PUT #1 &
!
7410 GOTO 7200  !loop back for next output record &
!
7500 PRINT "calling SRTENB"
7510 CALL SRTENB (err_buff() BY REF) &
!
7515 IF err_buff(0) = 0 THEN GOTO 7700 &
!
7600 PRINT "error in SRTENB:  err_buff(0) = ",err_buff(0)
7610 PRINT "                    err_buff(1) = ",err_buff(1)
7620 PRINT "                    err_buff(2) = ",err_buff(2)
7630 PRINT "                    err_buff(3) = ",err_buff(3) &
!
7700 CLOSE #1 &
!
! now for a merge mixed file to record &
!
7800 command_line = "=temp1/FO:V:80,temp2/fo:v:80/ke:col.1" &
!
7990 PRINT "calling MRGINB"
8000 CALL MRGINB (err_buff() BY REF, &
                wrk_area BY DESC, &
                command_line BY DESC, &
                inp_lrl BY REF, &
                0% BY VALUE, &
                0% BY VALUE, &
                lun_buf() BY DESC, &
                0% BY VALUE, &
                0% BY VALUE, &
                0% BY VALUE, &
                0% BY VALUE, &
                0% BY VALUE, &
                0% BY VALUE ) &

```

```

!
8100 IF err_buff(0) = 0 THEN GOTO 8275          &
!
8110 PRINT "error in MRGINB:  err_buff(0) = ",err_buff(0)
8120 PRINT "                  err_buff(1) = ",err_buff(1)
8130 PRINT "                  err_buff(2) = ",err_buff(2)
8140 PRINT "                  err_buff(3) = ",err_buff(3)
8200 GOTO 30900                                &
!
8275 PRINT "calling MRGMRB"
8300 CALL MRGMRB (err_buff() BY REF)
8310 IF err_buff(0) = 0 THEN GOTO 8500          &
!
8320 PRINT "error in MRGMRB:  err_buff(0) = ",err_buff(0)
8340 PRINT "                  err_buff(1) = ",err_buff(1)
8350 PRINT "                  err_buff(2) = ",err_buff(2)
8360 PRINT "                  err_buff(3) = ",err_buff(3)
8370 GOTO 30900                                &
!
8500 OPEN "OUTPUT.DAT" FOR OUTPUT AS FILE 1%,   &
      ORGANIZATION SEQUENTIAL FIXED,          &
      MAP DISK,                               &
      ACCESS WRITE,                           &
      ALLOW NONE                               &
!
8600 PRINT "calling MRGRTB"
8620 CALL MRGRTB (err_buff() BY REF,          &
      rec_buf BY DESC,                        &
      rtn_len BY REF)                        &
!
8640 IF err_buff(0) = 0 THEN GOTO 8900          &
!
8650 IF err_buff(0) = 1 THEN GOTO 9000          &
!
8660 PRINT "error in MRGRTB:  err_buff(0) = ",err_buff(0)
8670 PRINT "                  err_buff(1) = ",err_buff(1)
8675 PRINT "                  err_buff(2) = ",err_buff(2)
8680 PRINT "                  err_buff(3) = ",err_buff(3)
8690 GOTO 30900                                &
!
8900 PUT #1                                    &
!
8910 GOTO 8620      !loop back for next output record &
!
9000 PRINT "calling MRGENB"
9010 CALL MRGENB (err_buff() BY REF)          &
!
9020 IF err_buff(0) = 0 THEN GOTO 30900        &
!
9030 PRINT "error in MRGENB:  err_buff(0) = ",err_buff(0)
9040 PRINT "                  err_buff(1) = ",err_buff(1)
9050 PRINT "                  err_buff(2) = ",err_buff(2)
9060 PRINT "                  err_buff(3) = ",err_buff(3) &
!
9070 GOTO 30900      !it's all over            &
!
30000      !                                     &
          !      ERROR HANDLER                 &
          !
30100 IF ERR = 11% AND                          &
      ERL = 2800                                &
      THEN RESUME 3300      !it was end of input file &
!
30300 PRINT "falling through error handler"
30900 ON ERROR GOTO 0
32000 END

```

B.3 COBOL-81 Program Using the MERGE Record Interface

```
IDENTIFICATION DIVISION.
PROGRAM-ID. C81TST.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. PDP-11.
OBJECT-COMPUTER. PDP-11.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT OUTFIL ASSIGN TO "OUTPUT.DAT".
DATA DIVISION.
* PIC X IS ALPHA-NUMERIC
* PIC 9 IS NUMERIC (S MEANS SIGNED)
* COMP IS BINARY S9(4) COMP IS BINARY STORED IN ONE-WORD
FILE SECTION.
FD  OUTFIL
    LABEL RECORD STANDARD.
01  OUTREC PIC X(100).
WORKING-STORAGE SECTION.
01  SRT-WRK-AREA PIC X(9000).
01  SRT-WRK-LEN PIC S9(4) COMP VALUE 9000.
01  ERR-BUF.
    03  SRT-CODE PIC S9(4) COMP.
    03  RMSSTS PIC S9(4) COMP.
    03  RMSSTV PIC S9(4) COMP.
    03  SRT-EXTRA PIC S9(4) COMP.
01  CMD-BUF PIC X(1) VALUE "=" .
01  CMD-BUF-LEN PIC S9(4) COMP VALUE 1.
01  INP-LRL PIC S9(4) COMP VALUE 80.
01  MRG-ORD PIC S9(4) COMP VALUE 3.
01  SPEC-BUF PIC S9(4) COMP VALUE 0.
01  LB.
    03  LBX PIC S9(4) COMP VALUE 6.
    03  LUN-BUF REDEFINES LBX PIC XX.
01  LUN-BUF-LEN PIC S9(4) COMP VALUE 1.
01  REC-LEN PIC S9(4) COMP VALUE 100.
01  RTN-LEN PIC S9(4) COMP VALUE 666.
01  MISC-AREA.
    03  INCTR PIC 9(4) COMP VALUE 0.
    03  RELCTR PIC 9(4) COMP VALUE 0.
    03  RTNCTR PIC 9(4) COMP VALUE 0.
    03  OUTCTR PIC 9(4) COMP VALUE 0.
```

```

01 DISP-AREA.
03 LINE-1.
05 FILLER PIC X(17) VALUE "SORT TEST DISPLAY".
03 LINE-2.
05 FILLER PIC X(17) VALUE "RECORDS READ      :".
05 D-INCTR PIC 9(5) VALUE 0.
03 LINE-3.
05 FILLER PIC X(17) VALUE "RECORDS RELEASED:".
05 D-RELCTR PIC 9(5) VALUE 0.
03 LINE-4.
05 FILLER PIC X(17) VALUE "RECORDS RETURNED:".
05 D-RTNCTR PIC 9(5) VALUE 0.
03 LINE-5.
05 FILLER PIC X(17) VALUE "RECORDS WRITTEN :".
05 D-OUTCTR PIC 9(5) VALUE 0.
03 LINE-5A.
05 FILLER PIC X(17) VALUE "SORT ERROR REPORT".
03 LINE-6.
05 FILLER PIC X(17) VALUE "SORT ERROR CODE :".
05 D-SRT-CODE PIC S9(6) VALUE 0.
03 LINE-7.
05 FILLER PIC X(17) VALUE "RMS STS VALUE      :".
05 D-RMSSTS PIC S9(6) VALUE 0.
03 LINE-8.
05 FILLER PIC X(17) VALUE "RMS STV VALUE      :".
05 D-RMSSTV PIC S9(6) VALUE 0.
03 LINE-8A.
05 FILLER PIC X(17) VALUE "EXTRA SORT INFO :".
05 D-SRT-EXTRA PIC S9(6) VALUE 0.
03 LINE-9.
05 FILLER PIC X(17) VALUE "EXCEPTION REPORT ".
03 LINE-10.
05 FILLER PIC X(17) VALUE "SRT RTN LENGTH  :".
05 D-RTN-LEN PIC S9(6) VALUE 0.

PROCEDURE DIVISION.
START-UP.
OPEN OUTPUT OUTFIL.
DISPLAY "CALLING MRGINC".
CALL "MRGINC" USING BY REFERENCE ERR-BUF,
                  BY DESCRIPTOR SRT-WRK-AREA,
                  BY DESCRIPTOR CMD-BUF,
                  BY REFERENCE INP-LRL,
                  BY REFERENCE MRG-ORD,
                  BY REFERENCE SPEC-BUF,
                  BY DESCRIPTOR LUN-BUF.

IF SRT-CODE NOT = 0
    PERFORM ERROR-RTN
    GO TO THE-END.
DISPLAY "CALLING MRGRTC".

RTN-LOOP.
CALL "MRGRTC" USING BY REFERENCE ERR-BUF,
                  BY DESCRIPTOR OUTREC,
                  BY REFERENCE RTN-LEN.

IF SRT-CODE NOT = 0
    PERFORM ERROR-RTN
    GO TO THE-END.
ADD 1 TO RTNCTR.
WRITE OUTREC.
ADD 1 TO OUTCTR.
GO TO RTN-LOOP.

THE-END.

```

```

        DISPLAY "CALLING MRGENC".
        CALL "MRGENC" USING BY REFERENCE ERR-BUF.
*       MOVE INCTR TO D-INCTR.
*       MOVE RELCTR TO D-RELCTR.
*       MOVE RTNCTR TO D-RTNCTR.
*       MOVE OUTCTR TO D-OUTCTR.
*       DISPLAY LINE-1.
*       DISPLAY LINE-2.
*       DISPLAY LINE-3.
*       DISPLAY LINE-4.
*       DISPLAY LINE-5.
        CLOSE OUTFIL.
        STOP RUN.
ERROR-RTN.
        MOVE SRT-CODE TO D-SRT-CODE.
        MOVE RMSSTS TO D-RMSSTS.
        MOVE RMSSTV TO D-RMSSTV.
        MOVE SRT-EXTRA TO D-SRT-EXTRA.
        DISPLAY LINE-5A.
        DISPLAY LINE-6.
        DISPLAY LINE-7.
        DISPLAY LINE-8.
        DISPLAY LINE-8A.

```

B.4 COBOL-81 Program Using the SORT Record Interface

```

IDENTIFICATION DIVISION.
PROGRAM-ID. C81TES.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. PDP-11.
OBJECT-COMPUTER. PDP-11.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT INFILE ASSIGN TO "C81REC.DAT".
        SELECT OUTFIL ASSIGN TO "OUTPUT.DAT".
DATA DIVISION.
* PIC X IS ALPHA-NUMERIC
* PIC 9 IS NUMERIC (S MEANS SIGNED)
* COMP IS BINARY S9(4) COMP IS BINARY STORED IN ONE-WORD
FILE SECTION.
FD INFILE
        LABEL RECORD STANDARD.
01 INREC PIC X(100).
FD OUTFIL
        LABEL RECORD STANDARD.
01 OUTREC PIC X(100).
WORKING-STORAGE SECTION.
01 SRT-WRK-AREA PIC X(9000).
01 SRT-WRK-LEN PIC S9(4) COMP VALUE 9000.
01 ERR-BUF.
        03 SRT-CODE PIC S9(4) COMP.
        03 RMSSTS PIC S9(4) COMP.
        03 RMSSTV PIC S9(4) COMP.
        03 SRT-EXTRA PIC S9(4) COMP.
01 CMD-BUF PIC X(14) VALUE "/KE:CN1.2/FI:3".
01 CMD-BUF-LEN PIC S9(4) COMP VALUE 14.
01 INP-LRL PIC S9(4) COMP VALUE 100.
01 SPEC-BUF PIC S9(4) COMP VALUE 0.
01 LB.
        03 LBX PIC S9(4) COMP VALUE 6.
        03 LUN-BUF REDEFINES LBX PIC XX.
01 LUN-BUF-LEN PIC S9(4) COMP VALUE 1.
01 REC-LEN PIC S9(4) COMP VALUE 100.
01 RTN-LEN PIC S9(4) COMP VALUE 666.
01 MISC-AREA.

```

```

03 INCTR PIC 9(4) COMP VALUE 0.
03 RELCTR PIC 9(4) COMP VALUE 0.
03 RTNCTR PIC 9(4) COMP VALUE 0.
03 OUTCTR PIC 9(4) COMP VALUE 0.
01 DISP-AREA.
03 LINE-1.
05 FILLER PIC X(17) VALUE "SORT TEST DISPLAY".
03 LINE-2.
05 FILLER PIC X(17) VALUE "RECORDS READ      :".
05 D-INCTR PIC 9(5) VALUE 0.
03 LINE-3.
05 FILLER PIC X(17) VALUE "RECORDS RELEASED:".
05 D-RELCTR PIC 9(5) VALUE 0.
03 LINE-4.
05 FILLER PIC X(17) VALUE "RECORDS RETURNED:".
05 D-RTNCTR PIC 9(5) VALUE 0.
03 LINE-5.
05 FILLER PIC X(17) VALUE "RECORDS WRITTEN :".
05 D-OUTCTR PIC 9(5) VALUE 0.
03 LINE-5A.
05 FILLER PIC X(17) VALUE "SORT ERROR REPORT".
03 LINE-6.
05 FILLER PIC X(17) VALUE "SORT ERROR CODE :".
05 D-SRT-CODE PIC S9(6) VALUE 0.
03 LINE-7.
05 FILLER PIC X(17) VALUE "RMS STS VALUE      :".
05 D-RMSSTS PIC S9(6) VALUE 0.
03 LINE-8.
05 FILLER PIC X(17) VALUE "RMS STV VALUE      :".
05 D-RMSSTV PIC S9(6) VALUE 0.
03 LINE-8A.
05 FILLER PIC X(17) VALUE "EXTRA SORT INFO :".
05 D-SRT-EXTRA PIC S9(6) VALUE 0.
03 LINE-9.
05 FILLER PIC X(17) VALUE "EXCEPTION REPORT ".
03 LINE-10.
05 FILLER PIC X(17) VALUE "SRT RTN LENGTH :".
05 D-RTN-LEN PIC S9(6) VALUE 0.
PROCEDURE DIVISION.
START-UP.
OPEN INPUT INFILE.
OPEN OUTPUT OUTFIL.
DISPLAY "CALLING SRTINC".
CALL "SRTINC" USING BY REFERENCE ERR-BUF,
                  BY DESCRIPTOR SRT-WRK-AREA,
                  BY DESCRIPTOR CMD-BUF,
                  BY REFERENCE INP-LRL,
                  BY REFERENCE SPEC-BUF,
                  BY DESCRIPTOR LUN-BUF.
IF SRT-CODE NOT = 0
    PERFORM ERROR-RTN
    GO TO THE-END.
DISPLAY "ENTERING SRTRLC LOOP".

```

```

READ-LOOP.
  READ INFILE
  AT END
    DISPLAY "ENTERING SRTRTC LOOP"
    GO TO RTN-LOOP.
  ADD 1 TO INCTR.
  CALL "SRTRLC" USING BY REFERENCE ERR-BUF,
    BY DESCRIPTOR INREC.
  IF SRT-CODE NOT = 0
    PERFORM ERROR-RTN
    GO TO THE-END.
  ADD 1 TO RELCTR.
  GO TO READ-LOOP.
RTN-LOOP.
  CALL "SRTRTC" USING BY REFERENCE ERR-BUF,
    BY DESCRIPTOR OUTREC,
    BY REFERENCE RTN-LEN.
  IF SRT-CODE NOT = 0
    PERFORM ERROR-RTN
    GO TO THE-END.
  ADD 1 TO RTNCTR.
  IF RTN-LEN NOT = 100
    THEN MOVE RTN-LEN TO D-RTN-LEN
    DISPLAY LINE-9
    DISPLAY LINE-10
    PERFORM ERROR-RTN
    GO TO THE-END.
  WRITE OUTREC.
  ADD 1 TO OUTCTR.
  GO TO RTN-LOOP.
THE-END.
  DISPLAY "CALLING SRTENC".
  CALL "SRTENC" USING BY REFERENCE ERR-BUF.
  MOVE INCTR TO D-INCTR.
  MOVE RELCTR TO D-RELCTR.
  MOVE RTNCTR TO D-RTNCTR.
  MOVE OUTCTR TO D-OUTCTR.
  DISPLAY LINE-1.
  DISPLAY LINE-2.
  DISPLAY LINE-3.
  DISPLAY LINE-4.
  DISPLAY LINE-5.
  CLOSE INFILE.
  CLOSE OUTFIL.
  STOP RUN.
ERROR-RTN.
  MOVE SRT-CODE TO D-SRT-CODE.
  MOVE RMSSTS TO D-RMSSTS.
  MOVE RMSSTV TO D-RMSSTV.
  MOVE SRT-EXTRA TO D-SRT-EXTRA.
  DISPLAY LINE-5A.
  DISPLAY LINE-6.
  DISPLAY LINE-7.
  DISPLAY LINE-8.
  DISPLAY LINE-8A.

```

B.5 FORTRAN Program Using the MERGE File Interface

```
PROGRAM FMGFIL
C
C      THIS PROGRAM TESTS THE MERGE FILE INTERFACE.
C
C      INTEGER*2 IERROR(4), I1LUN, IWKSIZ, IWORK(8000)
C          RETURN      FIRST  SCR.   WORK
C          STATUS      MERGE  AREA   AREA
C          (4 WORDS)  LUN     SIZE
C
C      INTEGER*2 ILUNLN
C          LUN BUFFER LENGTH
C
C      INTEGER*2 MAXREC, ICOMLN, MRGEOF
C          MAXIMUM  COMMAND MERGE
C          INPUT   LINE    ERROR
C          RECORD  LENGTH  CODE (END OF FILE ERROR)
C          SIZE
C
C      INTEGER*2 INPSIZ, LENGTH, INRECS, OUTRCS
C          TOTAL   INPUT   INPUT   OUTPUT
C          INPUT   RECORD  RECORD  RECORD
C          FILES   LENGTH  COUNT   COUNT
C          SIZE
C
C      INTEGER*2 ENDFIL
C          END OF FILE FLAG
C
C
C      CHARACTER ROUTINE*8, COMAND*55, A(80)
C          ROUTINE  MERGE  MISC.
C          RETURNING COMMAND  STRING
C          THE ERROR LINE
C
C          EXTERNAL MRGINI, MRGMRG, MRGEND
C
C          MERGE-11 SUBROUTINES
C
C      DATA I1LUN, ILUNLN, IWKSIZ/6,1,16000/
C      DATA MAXREC, IERROR/20,0,0,0,0/
C      COMAND = 'OUT=FMGFIL/FO:F:20,FMGFL1/FO:F:20/KE:COL.20'
C      ICOMLN = 43
C
C          INITIALIZE MERGE PARAMETERS
C
C      ROUTINE=' MRGINI '
C      TYPE 902, ROUTINE
C      CALL MRGINI(IERROR, IWORK, IWKSIZ, COMAND, ICOMLN, MAXREC, 0, 0, 0,
X          I1LUN, ILUNLN)
```



```

C
C
C
CALL THE MERGE-11 INITIALIZE ROUTINE
C
C
C
IF (IERROR(1) .NE. 0) THEN
    GOTO 900
ENDIF
C
C
C
EXIT AND TYPE AN ERROR MESSAGE IF MRGINI WAS UNSUCCESSFUL.
C
C
C
INRECS = 0
OUTRCS = 0
ENDFIL = 0
IFILE = 1
C
C
C
RUTINE=' MRGMRG '
TYPE 902, RUTINE
150 CALL MRGMRG(IERROR)
C
C
C
START THE MERGING PROCEDURE
C
C
C
IF (IERROR(1) .NE. 0) THEN
    GOTO 900
ENDIF
C
C
C
EXIT AND TYPE AN ERROR MESSAGE IF MRGRLS WAS UNSUCCESSFUL.
C
C
C
RUTINE=' MRGEND '
TYPE 902, RUTINE
CALL MRGEND(IERROR)
C
C
C
CALL THE MERGE-11 CLEAN-UP ROUTINES.
C
C
C
IF (IERROR(1) .NE. 0) THEN
    GOTO 900
ENDIF
C
C
C
EXIT AND TYPE AN ERROR MESSAGE IF UNSUCCESSFUL.
C
C
C
STOP 'SUCCESSFUL FORTRAN MERGE TEST.'
C
C
C
C
C
C
900 TYPE 901, RUTINE
901 FORMAT(/' ERROR OCCURRED IN ',A8//)
902 FORMAT(/' CALLING ',A8//)
C
C
C
TYPE ERROR MESSAGE GIVING FAILING ROUTINE.
C
C
C
TYPE 903, IERROR
903 FORMAT(' ERROR STATUS = ',I6,///' STS= ',I6,5X,' STV= 'I6)
C
C
C
TYPE RETURNED STATUS VALUES.
C
C
C
C
C
C
STOP 'AN ERROR OCCURRED CALLING MERGE FROM FORTRAN.'
END

```

B.6 FORTRAN Program Using the SORT File Interface

```
PROGRAM FORFIL
C
C      THIS PROGRAM TESTS THE SORT FILE INTERFACE.
C
C      INTEGER*2 IERROR(4), I1LUN, IWKSIZ, IWORK(8000)
C      RETURN      FIRST SCR. WORK
C      STATUS      SORT AREA AREA
C      (4 WORDS) LUN SIZE
C
C      INTEGER*2 ILUNLN
C      LUN BUFFER LENGTH
C
C      INTEGER*2 MAXREC, ICOMLN, SRTEOF
C      MAXIMUM COMMAND SORT
C      INPUT LINE ERROR
C      RECORD LENGTH CODE (END OF FILE ERROR)
C      SIZE
C
C      INTEGER*2 INPSIZ, LENGTH, INRECS, OUTRCS
C      TOTAL INPUT INPUT OUTPUT
C      INPUT RECORD RECORD RECORD
C      FILES LENGTH COUNT COUNT
C      SIZE
C
C      INTEGER*2 ENDFIL
C      END OF FILE FLAG
C
C
C      CHARACTER ROUTINE*8, COMAND*45, A(80)
C      ROUTINE SORT MISC.
C      RETURNING COMMAND STRING
C      THE ERROR LINE
C
C      EXTERNAL SRTINI, SRTSRT, SRTEND
C
C      SORT-11 SUBROUTINES
C
C      DATA I1LUN, ILUNLN, IWKSIZ/6,1,16000/
C      DATA MAXREC, IERROR/20,0,0,0,0/
C      COMAND = 'OUT=FORFIL/FO:F:20/KE:CN1.20/FI:3/PT:30'
C      ICOMLN = 39
C
C      INITIALIZE SORT PARAMETERS
C
C      ROUTINE=' SRTINI '
C      TYPE 902, ROUTINE
C      CALL SRTINI(IERROR, IWORK, IWKSIZ, COMAND, ICOMLN, MAXREC, 0, 0,
X      I1LUN, ILUNLN)
C
C      CALL THE SORT-11 INITIALIZE ROUTINE
C
C      IF (IERROR(1) .NE. 0) THEN
C      GOTO 900
C      ENDIF
C
C      EXIT AND TYPE AN ERROR MESSAGE IF SRTINI WAS UNSUCCESSFUL.
C
C
C      INRECS = 0
C      OUTRCS = 0
C      ENDFIL = 0
C      IFILE = 1
C
C
C
```

```

C
      ROUTINE=' SRTCLS '
      TYPE 902, ROUTINE
150      CALL SRTSRT(IERROR)
C
C          START THE SORTING PROCEDURE
C
      IF (IERROR(1) .NE. 0) THEN
          GOTO 900
      ENDIF
C
      EXIT AND TYPE AN ERROR MESSAGE IF SRTCLS WAS UNSUCCESSFUL.
C
      ROUTINE=' SRTEND '
      TYPE 902, ROUTINE
      CALL SRTEND(IERROR)
C
C          CALL THE SORT-11 CLEAN-UP ROUTINES.
C
      IF (IERROR(1) .NE. 0) THEN
          GOTO 900
      ENDIF
C
      EXIT AND TYPE AN ERROR MESSAGE IF UNSUCCESSFUL.
C
      STOP ' SUCCESSFUL FORTRAN/SORT TEST.'
C
C
C
C
900      TYPE 901, ROUTINE
901      FORMAT('/' ERROR OCCURRED IN ',A8//)
902      FORMAT('/' CALLING ',A8//)
C
C          TYPE ERROR MESSAGE GIVING FAILING ROUTINE.
C
      TYPE 903, IERROR
903      FORMAT(' ERROR STATUS = ',I6, '/' STS= ',I6,5X,' STV= 'I6)
C
C          TYPE RETURNED STATUS VALUES.
C
C
C
      STOP 'AN ERROR OCCURRED CALLING SORT FROM FORTRAN.'
      END

```


DIGITAL Multinational and ASCII Collating Sequences

Table C-1 contains the DIGITAL Multinational Collating Sequence, and Table C-2 contains the ASCII Collating Sequence.

Table C-1: DIGITAL Multinational Collating Sequence

HEX Code	Octal Code	Decimal Code	Char or Abbrev.	Description
00	000	000	NUL	Null character
01	001	001	SOH	Start of heading
02	002	002	STX	Start of text
03	003	003	ETX	End of text
04	004	004	EOT	End of transmission
05	005	005	ENQ	Enquiry
06	006	006	ACK	Acknowledge
07	007	007	BEL	Bell
08	010	008	BS	Backspace
09	011	009	HT	Horizontal tabulation
0A	012	010	LF	Line feed
0B	013	011	VT	Vertical tabulation
0C	014	012	FF	Form feed
0D	015	013	CR	Carriage return
0E	016	014	SO	Shift out
0F	017	015	SI	Shift in
10	020	016	DLE	Data link escape
11	021	017	DC1	Device control 1
12	022	018	DC2	Device control 2
13	023	019	DC3	Device control 3
14	024	020	DC4	Device control 4
15	025	021	NAK	Negative acknowledge
16	026	022	SYN	Synchronous idle
17	027	023	ETB	End of transmission block

(continued on next page)

Table C-1 (Cont.): DIGITAL Multinational Collating Sequence

HEX Code	Octal Code	Decimal Code	Char or Abbrev.	Description
18	030	024	CAN	Cancel
19	031	025	EM	End of medium
1A	032	026	SUB	Substitute
1B	033	027	ESC	Escape
1C	034	028	FS	File separator
1D	035	029	GS	Group separator
1E	036	030	RS	Record separator
1F	037	031	US	Unit separator
20	040	032	SP	Space
21	041	033	!	Exclamation point
22	042	034	"	Quotation marks (double quote)
23	043	035	#	Number sign
24	044	036	\$	Dollar sign
25	045	037	%	Percent sign
26	046	038	&	Ampersand
27	047	039	'	Apostrophe (single quote)
28	050	040	(Opening parenthesis
29	051	041)	Closing parenthesis
2A	052	042	*	Asterisk
2B	053	043	+	Plus
2C	054	044	,	Comma
2D	055	045	-	Hyphen or minus
2E	056	046	..	Period or decimal point
2F	057	047	/	Slash
30	060	048	0	Zero
31	061	049	1	One
32	062	050	2	Two
33	063	051	3	Three
34	064	052	4	Four
35	065	053	5	Five
36	066	054	6	Six
37	067	055	7	Seven
38	070	056	8	Eight
39	071	057	9	Nine
3A	072	058	:	Colon
3B	073	059	;	Semicolon
3C	074	060	<	Less than
3D	075	061	=	Equals

(continued on next page)

Table C-1 (Cont.): DIGITAL Multinational Collating Sequence

HEX Code	Octal Code	Decimal Code	Char or Abbrev.	Description
3E	076	062	>	Greater than
3F	077	063	?	Question mark
40	100	064	@	Commercial at
61	141	097	a	Lowercase a
41	101	065	A	Uppercase A
E0	340	224	à	Lowercase a with grave accent
C0	300	192	À	Uppercase A with grave accent
E1	341	225	á	Lowercase a with acute acent
C1	301	193	Á	Uppercase A with acute accent
E2	342	226	â	Lowercase a with circumflex
C2	302	194	Â	Uppercase A with circumflex
E3	343	227	ã	Lowercase a with tilde
C3	303	195	Ã	Uppercase A with tilde
E4	344	228	ä	Lowercase a with umlaut, (diaeresis)
C4	304	196	Ä	Uppercase A with umlaut, (diaeresis)
62	142	098	b	Lowercase b
42	102	066	B	Uppercase B
63	143	099	c	Lowercase c
43	103	067	C	Uppercase C
E7	347	231	ç	Lowercase c with cedilla
C7	307	199	Ç	Uppercase C with cedilla
64	144	100	d	Lowercase d
44	104	068	D	Uppercase D
65	145	101	e	Lowercase e
45	105	069	E	Uppercase E
E8	350	232	è	Lowercase e with grave accent
C8	310	200	È	Uppercase E with grave accent
E9	351	233	é	Lowercase e with acute accent
C9	311	201	É	Uppercase E with acute accent
EA	352	234	ê	Lowercase e with circumflex
CA	312	202	Ê	Uppercase E with circumflex
EB	353	235	ë	Lowercase e with umlaut, (diaeresis)
CB	313	203	Ë	Uppercase E with umlaut, (diaeresis)
66	146	102	f	Lowercase f
46	106	070	F	Uppercase F
67	147	103	g	Lowercase g
47	107	071	G	Uppercase G
68	150	104	h	Lowercase h

(continued on next page)

Table C-1 (Cont.): DIGITAL Multinational Collating Sequence

HEX Code	Octal Code	Decimal Code	Char or Abbrev.	Description
48	110	072	H	Uppercase H
69	151	105	i	Lowercase i
49	111	073	I	Uppercase I
EC	354	236	ì	Lowercase i with grave accent
CC	314	204	Ì	Uppercase I with grave accent
ED	355	237	í	Lowercase i with acute accent
CD	315	205	Í	Uppercase I with acute accent
EE	356	238	î	Lowercase i with circumflex
CE	316	206	Î	Uppercase I with circumflex
EF	357	239	ï	Lowercase i with umlaut, (diaeresis)
CF	317	207	Ï	Uppercase I with umlaut, (diaeresis)
6A	152	106	j	Lowercase j
4A	112	074	J	Uppercase J
6B	153	107	k	Lowercase k
4B	113	075	K	Uppercase K
6C	154	108	l	Lowercase l
4C	114	076	L	Uppercase L
6D	155	109	m	Lowercase m
4D	115	077	M	Uppercase M
6E	156	110	n	Lowercase n
4E	116	078	N	Uppercase N
F1	361	241	ñ	Lowercase n with tilde
D1	321	209	Ñ	Uppercase N with tilde
6F	157	111	o	Lowercase o
4F	117	079	O	Uppercase O
F2	362	242	ò	Lowercase o with grave acent
D2	322	210	Ò	Uppercase O with grave accent
F3	363	243	ó	Lowercase o with acute accent
D3	323	211	Ó	Uppercase O with acute accent
F4	364	244	ô	Lowercase o with circumflex
D4	324	212	Ô	Uppercase O with circumflex
F5	365	245	õ	Lowercase o with tilde
D5	325	213	Õ	Uppercase O with tilde
F6	366	246	ö	Lowercase o with umlaut, (diaeresis)
D6	326	214	Ö	Uppercase O with umlaut, (diaeresis)
F7	367	247	œ	Lowercase oe ligature
D7	327	215	Œ	Uppercase OE ligature
70	160	112	p	Lowercase p

(continued on next page)

Table C-1 (Cont.): DIGITAL Multinational Collating Sequence

HEX Code	Octal Code	Decimal Code	Char or Abbrev.	Description
50	120	080	P	Uppercase P
71	161	113	q	Lowercase q
51	121	081	Q	Uppercase Q
72	162	114	r	Lowercase r
52	122	082	R	Uppercase R
73	163	115	s	Lowercase s
53	123	083	S	Uppercase S
DF	337	223	ß	German lowercase sharp s
74	164	116	t	Lowercase t
54	124	084	T	Uppercase T
75	165	117	u	Lowercase u
55	125	085	U	Uppercase U
F9	371	249	ù	Lowercase u with grave accent
D9	331	217	Û	Uppercase U with grave accent
FA	372	250	ú	Lowercase u with acute accent
DA	332	218	Ú	Uppercase U with acute accent
FB	373	251	û	Lowercase u with circumflex
DB	333	219	Û	Uppercase U with circumflex
FC	374	252	ü	Lowercase u with umlaut, (diaeresis)
DC	334	220	Û	Uppercase U with umlaut, (diaeresis)
76	166	118	v	Lowercase v
56	126	086	V	Uppercase V
77	167	119	w	Lowercase w
57	127	087	W	Uppercase W
78	170	120	x	Lowercase x
58	130	088	X	Uppercase X
79	171	121	y	Lowercase y
59	131	089	Y	Uppercase Y
FD	375	253	ÿ	Lowercase y with umlaut, (diaeresis)
DD	335	221	ÿ	Uppercase Y with umlaut, (diaeresis)
7A	172	122	z	Lowercase z
5A	132	090	Z	Uppercase Z
E6	346	230	æ	Lowercase ae diphthong
C6	306	198	Æ	Uppercase AE with diphthong
F8	370	248	ø	Lowercase o with slash
D8	330	216	Ø	Uppercase O with slash
E5	345	229	å	Lowercase a with ring
C5	305	197	Å	Uppercase A with ring

(continued on next page)

Table C-1 (Cont.): DIGITAL Multinational Collating Sequence

HEX Code	Octal Code	Decimal Code	Char or Abbrev.	Description
5B	133	091	[Opening bracket
5C	134	092	\	Backslash
5D	135	093]	Closing bracket
5E	136	094	^	Circumflex
5F	137	095	_	Underline (underscore)
60	140	096	‘	Grave accent
7B	173	123	{	Opening brace
7C	174	124		Vertical line
7D	175	125	}	Closing brace
7E	176	126	~	Tilde
7F	177	127	DEL	Delete, rubout
84	204	132	IND	Index
85	205	133	NEL	Next line
86	206	134	SSA	Start of selected area
87	207	135	ESA	End of selected area
88	210	136	HTS	Horizontal tab set
89	211	137	HTJ	Horizontal tab set with justification
8A	212	138	VTS	Vertical tab set
8B	213	139	PLD	Partial line down
8C	214	140	PLU	Partial line up
8D	215	141	RI	Reverse index
8E	216	142	SS2	Single shift 2
8F	217	143	SS3	Single shift 3
90	220	144	DCS	Device control string
91	221	145	PU1	Private use 1
92	222	146	PU2	Private use 2
93	223	147	STS	Set transmit state
94	224	148	CCH	Cancel character
95	225	149	MW	Message waiting
96	226	150	SPA	Start of protected area
97	227	151	EPA	End of protected area
9B	233	155	CSI	Control sequence introducer
9C	234	156	ST	String terminator
9D	235	157	OSC	Operating system command
9E	236	158	PM	Privacy message
9F	237	159	APC	Application
A1	241	161	¡	Inverted exclamation mark
A2	242	162	¢	Cent sign

(continued on next page)

Table C-1 (Cont.): DIGITAL Multinational Collating Sequence

HEX Code	Octal Code	Decimal Code	Char or Abbrev.	Description
A3	243	163	£	Pound sign
A5	245	165	¥	Yen sign
A7	247	167	§	Section sign
A8	250	168	¤	General currency sign
A9	251	169	©	Copyright sign
AA	252	170	♀	Feminine ordinal indicator
AB	253	171	«	Angle quotation mark left
B0	260	176	°	Degree sign
B1	261	177	±	Plus/minus sign
B2	262	178	²	Superscript 2
B3	263	179	³	Superscript 3
B5	265	181	µ	Micro sign
B6	266	182	¶	Paragraph sign, pilcrow
B7	267	183	.	Middle dot
B9	271	185	¹	Superscript 1
BA	272	186	♂	Masculine ordinal indicator
BB	273	187	»	Angle quotation mark right
BC	274	188	¼	Fraction one quarter
BD	275	189	½	Fraction one half
BF	277	191	¿	Inverted question mark

Table C-2: ASCII Collating Sequence

ASCII	Hex	Octal	Decimal
NUL	00	000	0
SOH	01	001	1
STX	02	002	2
ETX	03	003	3
EOT	04	004	4
ENQ	05	005	5
ACK	06	006	6
BEL	07	007	7
BS	08	010	8
HT	09	011	9
LF	0A	012	10
VT	0B	013	11
FF	0C	014	12
CR	0D	015	13

(continued on next page)

Table C-2 (Cont.): ASCII Collating Sequence

ASCII	Hex	Octal	Decimal
SO	0E	016	14
SI	0F	017	15
DLE	10	020	16
DC1	11	021	17
DC2	12	022	18
DC3	13	023	19
DC4	14	024	20
NAK	15	025	21
SYN	16	026	22
ETB	17	027	23
CAN	18	030	24
EM	19	031	25
SUB	1A	032	26
ESC	1B	033	27
FS	1C	034	28
GS	1D	035	29
RS	1E	036	30
US	1F	037	31
SP	20	040	32
!	21	041	33
"	22	042	34
#	23	043	35
\$	24	044	36
%	25	045	37
&	26	046	38
'	27	047	39
(28	050	40
)	29	051	41
*	2A	052	42
+	2B	053	43
,	2C	054	44
-	2D	055	45
.#	2E	056	46
/	2F	057	47
0	30	060	48
1	31	061	49
2	32	062	50
3	33	063	51
4	34	064	52

(continued on next page)

Table C-2 (Cont.): ASCII Collating Sequence

ASCII	Hex	Octal	Decimal
5	35	065	53
6	36	066	54
7	37	067	55
8	38	070	56
9	39	071	57
:	3A	072	58
;	3B	073	59
<	3C	074	60
=	3D	075	61
>	3E	076	62
?	3F	077	63
@	40	100	64
A	41	101	65
B	42	102	66
C	43	103	67
D	44	104	68
E	45	105	69
F	46	106	70
G	47	107	71
H	48	110	72
I	49	111	73
J	4A	112	74
K	4B	113	75
L	4C	114	76
M	4D	115	77
N	4E	116	78
O	4F	117	79
P	50	120	80
Q	51	121	81
R	52	122	82
S	53	123	83
T	54	124	84
U	55	125	85
V	56	126	86
W	57	127	87
X	58	130	88
Y	59	131	89
Z	5A	132	90
[5B	133	91

(continued on next page)

Table C-2 (Cont.): ASCII Collating Sequence

ASCII	Hex	Octal	Decimal
\	5C	134	92
]	5D	135	93
^	5E	136	94
-	5F	137	95
'	60	140	96
a	61	141	97
b	62	142	98
c	63	143	99
d	64	144	100
e	65	145	101
f	66	146	102
g	67	147	103
h	68	150	104
i	69	151	105
j	6A	152	106
k	6B	153	107
l	6C	154	108
m	6D	155	109
n	6E	156	110
o	6F	157	111
p	70	160	112
q	71	161	113
r	72	162	114
s	73	163	115
t	74	164	116
u	75	165	117
v	76	166	118
w	77	167	119
x	78	170	120
y	79	171	121
z	7A	172	122
{	7B	173	123
	7C	174	124
}	7D	175	125
~	7E	176	126
DEL	7F	177	127

Index

A

- Address sort, 5–5
 - reasons for selecting, 2–20
- Algorithms
 - merge, 5–2
 - polyphase merge, 5–2
 - replacement selection, 5–2
- ASCII Collating Sequence, C–7
- ASCII data type, 2–17

B

- BASIC–PLUS–2 sample program
 - using both SORT and MERGE mixed-mode interfaces, B–3
 - using the MERGE file interface, B–2
- Batch processing, 1–8
 - using to maximize SORT/MERGE performance, 5–10
- BINARY data type, 2–16, 2–17
- /BK switch, 2–23
- Bucket size
 - default and maximum sizes, 5–9
- /BUCKET_SIZE=n qualifier, 5–9
- /BU switch, 5–9

C

- Callable SORT/MERGE, 4–2, 5–6
- Chaining (RSTS/E only), 2–25
- CHARACTER data type, 2–16, 2–17
- /CHECK_SEQUENCE qualifier, 2–26
- /CH switch, 2–26
- COBOL–81 sample program
 - using the MERGE record interface, B–7
 - using the SORT record interface, B–9
- Collating sequence, 5–6
 - ASCII, 2–17
 - EBCDIC, 2–17
 - in a specification file, 3–11
 - MULTINATIONAL, 2–17
 - rules for defining, 3–12
 - rules for modifying, 3–14
 - user-defined, 3–11, 3–15
- /COLLATING_SEQUENCE qualifier, 2–17
 - rules for using, 3–13
- Command line
 - continuing to a second line, 1–8
- Comment character (!)
 - restriction, 3–2
- Continuation character, 1–8

/CS switch, 2–17

D

- Data types, 2–17, 3–9
 - and determining size of key fields, 2–17
 - DCL, 2–15
 - default, 2–15
 - MCR, 2–16
- DECIMAL data type, 2–16, 2–17
- /DE switch, 5–8
- DIGITAL Multinational Collating Sequence, C–1
 - and sequence checking, 2–18
 - ordering procedures, 2–17
- Diphthong
 - collating, 3–14
- Double character
 - defining as single, 3–13
- D_FLOATING data type
 - implicit size, 1–4

E

- Equal key fields, 4–10
- Error messages, A–1
 - codes, A–1t

F

- Fibonacci number
 - defined, 5–3
- Fields
 - identifying in a specification file, 3–4
- File interface
 - summary of SORT subroutine calls, 4–18, 4–19, 4–20
- File-name extensions
 - default, 1–7
- File organization
 - indexed-sequential, 2–22, 2–24
 - relative, 2–22, 2–24
 - sequential, 2–22, 2–24
- Files
 - See also Input file, Output file, and Work files
 - having different formats, 3–10
 - ODL, 4–3, 4–21, 4–22t
 - sorting multiple, 1–1
- File size
 - default for input file, 2–23
 - default for output file, 2–25
 - determining, 2–23

File size (Cont.)

- maximum for input file, 2-23
- /FI switch, 5-3
- FOLD subqualifier, 3-12 to 3-13
 - and collating sequences, 3-14
- /FORMAT qualifier, 2-23, 2-25
- FORTTRAN sample program
 - using the MERGE file interface, B-12
 - using the SORT file interface, B-14
- /FO switch, 2-23, 2-25
- F_FLOATING data type
 - implicit size, 1-4

I

- IGNORE subqualifier, 3-14
- /INCLUDE qualifier, 3-9
- /INDEXED_SEQUENTIAL qualifier, 2-22, 2-24
- Index sort, 5-5
 - reasons for selecting, 2-21
- Input file, 1-2, 5-1, 5-8
 - attributes, 2-22
- /IN switch, 2-22, 2-24

K

- /KE switch, 1-4, 2-14
- Key field, 1-2
 - determining size of, 2-17
 - equal key fields, 2-19
 - maximum size according to data type, 2-16
 - multiple, 2-18
 - primary, 1-4
 - secondary, 1-4
 - specifying in MCR, 2-18
- /KEY qualifier, 1-4, 2-14
 - example, 1-4
- Knuth, Donald, 5-3

L

- Languages
 - supported by PDP-11 SORT/MERGE, 4-1
- Ligature
 - collating, 3-14
- /LOAD_FILL qualifier, 5-9
- Logical unit number
 - See LUN
- Longest record length
 - See LRL
- /LO switch, 5-9
- LRL
 - maximum by file organization, 2-22
- LUN usage, 4-25

M

- MCR
 - command line, 1-7
 - default values, 1-8
 - using MERGE with, 1-7
 - using SORT with, 1-7
- Memory
 - minimum amount needed for sort operation, 4-6
- MERGE command
 - format, 1-6

MERGE command (Cont.)

- using qualifiers, 1-7
 - using qualifiers in a specification file, 3-16
 - using switches, 1-7
- MERGE MCR command line, 1-7
- Merge process
 - initializing, 4-15
- Merge subroutines, 4-13t
 - functions, 4-13
- MODIFICATION subqualifier, 3-13, 3-14
- MRGCLB global symbol, 4-18
- MRGCMP global symbol, 4-17
- MRGENB subroutine
 - using with file interface, 4-14
- MRGENC subroutine
 - using with file interface, 4-14
- MRGEND subroutine
 - using with file interface, 4-14
 - using with mixed-mode interface, 4-15
- MRGINB subroutine, 4-15
 - using with file interface, 4-14
 - using with record interface, 4-15
- MRGINC subroutine, 4-15
 - using with file interface, 4-14
 - using with record interface, 4-15
- MRGINI subroutine
 - required parameters, 4-15
 - using with file interface, 4-14
 - using with mixed-mode interface, 4-15
 - using with record interface, 4-15
- MRGINP global symbol, 4-17
- MRGMRG subroutine
 - using with file interface, 4-14
 - using with mixed-mode interface, 4-15
- MRGRTB subroutine
 - using with record interface, 4-15
- MRGRTC subroutine
 - using with record interface, 4-15
- MRGRTN subroutine
 - using with record interface, 4-15
- MRGSRB subroutine
 - using with file interface, 4-14
- MRGSRC subroutine
 - using with file interface, 4-14
- MRGW RN global symbol, 4-17

N

- /ND switch, 2-19, 4-10
- Node size, 5-2, 5-5
- /NODUPLICATES qualifier, 2-19
- Null character
 - specifying, 3-12
- Null parameter, 4-3
 - options when not permitted by language, 4-13

O

- ODL file, 4-21
 - location on RSTS/E, 4-22
 - location on RSX-11M and RSX-11M-PLUS, 4-22
 - sample, 4-22
- /OMIT qualifier, 3-9
- Output file, 1-2, 5-1
 - attributes, 2-24
 - empty, 2-24
 - how default size is determined, 5-8

Output file (Cont.)
 how to change default size, 5–8
/OVERLAY qualifier, 2–24
/OV switch, 2–24

P

PACKED-DECIMAL data type, 2–17
Pad character
 specifying, 3–16
Parameters
 passing by descriptor, 4–1
 passing by reference, 4–1 to 4–2
POSITION subqualifier, 1–4
 example, 1–4
 identifying, 1–4
/PROCESS qualifier, 2–21
/PR switch, 2–21
/PT switch, 2–27, 5–7, 5–9

Q

Qualifiers
 MERGE command, 2–8
 SORT command, 2–2
Qualifiers
 negative form, 1–3
 syntax, 1–3
 used in a specification file, 3–3

R

Record format
 controlled, 2–23, 2–24, 2–25
 default for output file, 2–24
 differing between input and output files, 2–24
 fixed, 2–23, 2–24
 RMS stream, 2–23, 2–24
 stream, 2–23, 2–24
 unknown, 2–23, 2–24
 variable, 2–23, 2–24
Record interface
 summary of SORT subroutine calls, 4–18, 4–19, 4–20
Record Management Services
 See RMS
Record sort
 reasons for selecting, 2–20
/RELATIVE qualifier, 2–24
/RE switch, 2–24
RMS
 files accepted by SORT/MERGE, 2–22
Routines
 equal-key-field, 5–2
 user-defined, 4–23

S

Sequence checking, 2–26
SEQUENCE subqualifier, 3–12
/SEQUENTIAL qualifier, 2–24
/SE switch, 2–24
Shareable file, 2–23
SIZE subqualifier, 1–4
 example, 1–4

SORT command
 format, 1–2
 using qualifiers, 1–3
 using qualifiers in a specification file, 3–16
 using switches, 1–3
SORT MCR command line, 1–7
SORT/MERGE subroutines
 See also Sort subroutines and Merge subroutines
 location, 4–2
Sort operation
 first initialization phase, 5–1
 merge phase, 5–3
 second initialization phase, 5–2
 sort phase, 5–2
Sort processes, 2–19
 and node size, 5–5
 choosing, 5–9
 defined, 2–20t
 differences in speed, 5–9
 initializing, 4–5
 passing file names to, 4–5
Sort subroutines, 4–2, 4–4t
 accessing, 4–2
 BASIC, 4–2
 COBOL, 4–2
 file interface, 4–2
 functions, 4–4
 mixed-mode interface, 4–2
 names according to language, 4–2
 parameters, 4–3
 record interface, 4–2
 using with file interface, 4–4
 using with mixed-mode interface, 4–5
 using with record interface, 4–5
Specification file
 format, 3–16
SRTCLB global symbol, 4–10
SRTCMP global symbol, 4–10
SRTENB subroutine
 using to end a sort operation, 4–13
 using with file interface, 4–4
 using with record interface, 4–5
SRTENC subroutine
 using to end a sort operation, 4–13
 using with file interface, 4–4
 using with record interface, 4–5
SRTEND subroutine
 using to end a sort operation, 4–13
 using with file interface, 4–4
 using with mixed-mode interface, 4–5
 using with record interface, 4–5
SRTINB
 required parameters, 4–5
SRTINB subroutine
 using with file interface, 4–4
 using with record interface, 4–5
SRTINC
 required parameters, 4–5
SRTINC subroutine
 using with file interface, 4–4
 using with record interface, 4–5
SRTINI
 required parameters, 4–5
SRTINI subroutine
 using with file interface, 4–4
 using with mixed-mode interface, 4–5
 using with record interface, 4–5

- SRTRLB subroutine, 4-11
 - using with record interface, 4-5
- SRTRLC subroutine, 4-11
 - using with record interface, 4-5
- SRTRLS subroutine, 4-11
 - using with record interface, 4-5
- SRTRTB subroutine, 4-11
 - using with record interface, 4-5
- SRTRTC subroutine, 4-11
 - using with record interface, 4-5
- SRTRTN subroutine, 4-11
 - using with record interface, 4-5
- SRTSRB subroutine
 - using with file interface, 4-4
 - using with file or mixed-mode interfaces, 4-12
- SRTSRC subroutine
 - using with file interface, 4-4
 - using with file or mixed-mode interfaces, 4-12
- SRTSRT subroutine
 - using with file interface, 4-4
 - using with file or mixed-mode interfaces, 4-12
 - using with mixed-mode interface, 4-5
- /SS switch, 5-3
- /STABLE qualifier, 2-19, 4-10
- Statistics
 - defined, 5-4
 - sample display, 5-4
- /STATISTICS qualifier, 5-3, 5-6
- /ST switch, 2-19
- Subqualifiers
 - syntax, 1-3
- Subswitches
 - syntax, 1-3

Switches

- negative form, 1-3
- syntax, 1-3

T

Tag sort, 5-5

- reasons for selecting, 2-20

Task building, 4-3, 4-21

TIE_BREAK subqualifier, 3-12 to 3-13

- and collating sequences, 3-14

/TREE_SPACE qualifier, 2-27, 5-7, 5-9

U

User-defined routines, 4-3

- and task building, 4-23

W

Work area

- data structure, 5-2
- default size, 2-27, 5-9
- I/O requirements, 2-27
- minimum amount needed for merge operation, 4-16
- optimizing, 2-27, 5-9
- tree, 5-2

Work files, 5-2, 5-3, 5-7

- default sizes, 4-9
- how default size is determined, 5-8
- reassigning, 3-15

/WORK_FILES qualifier, 5-3, 5-8

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	_____	Local Digital subsidiary or approved distributor
Internal ¹	_____	USASSB Order Processing - WMO/E15 <i>or</i> U.S. Area Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

¹For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

PDP-11 SORT/MERGE
User's Guide
AA-C167B-TC

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Completeness (enough information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Clarity (easy to understand)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Organization (structure of subject matter)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Figures (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Examples (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Index (ability to find topic)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Page layout (easy to find information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

I would like to see more/less _____

What I like best about this manual is _____

What I like least about this manual is _____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

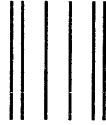
Additional comments or suggestions to improve this manual:

I am using **Version** _____ of the software this manual describes.

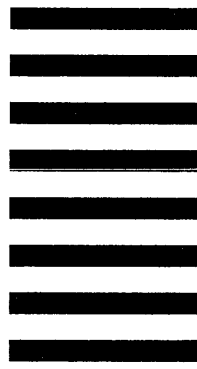
Name/Title _____ Dept. _____
Company _____ Date _____
Mailing Address _____
_____ Phone _____

Do Not Tear - Fold Here and Tape

digital



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

**DIGITAL EQUIPMENT CORPORATION
CORPORATE USER PUBLICATIONS
PKO3-1/D30
129 PARKER STREET
MAYNARD, MA 01754-9975**



Do Not Tear - Fold Here and Tape

Reader's Comments

PDP-11 SORT/MERGE
User's Guide
AA-CI67B-TC

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Completeness (enough information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Clarity (easy to understand)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Organization (structure of subject matter)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Figures (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Examples (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Index (ability to find topic)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Page layout (easy to find information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

I would like to see more/less _____

What I like best about this manual is _____

What I like least about this manual is _____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

I am using **Version** _____ of the software this manual describes.

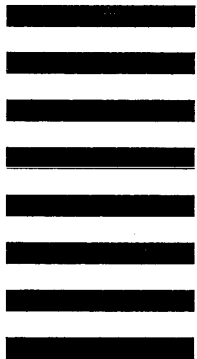
Name/Title _____ Dept. _____
Company _____ Date _____
Mailing Address _____
_____ Phone _____

Do Not Tear - Fold Here and Tape

digital



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

**DIGITAL EQUIPMENT CORPORATION
CORPORATE USER PUBLICATIONS
PKO3-1/D30
129 PARKER STREET
MAYNARD, MA 01754-9975**



Do Not Tear - Fold Here and Tape