PDP–11 SORT/MERGE
User's Guide

Order No. AA-CI67A-TC

April 1984

This manual describes how to use the PDP-11 SORT/MERGE utility. The manual is intended for all RSTS/E, RSX-11M/M-PLUS, and Micro/RSX users of SORT/MERGE who need to reorder or combine data files.

OPERATING SYSTEMS AND VERSION:	RSTS/E V8.0 RSX-11M V4.1 RSX-11M-PLUS V2.1 Micro/RSX V1.0
SOFTWARE VERSION:	PDP-11 SORT/MERGE V3.0

digital equipment corporation · maynard, massachusetts

First Printing, April 1984

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.

The 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 on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

Printed in U.S.A.

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	RSTS
DEC/CMS	EduSystem	RSX
DEC/MMS	IAS	TOPS-20
DECnet	MASSBUS	UNIBUS
DECsystem-10	Micro/PDP-11	VAX
DECSYSTEM-20	Micro/RSX	VMS
DECUS	PDP	VT
DECwriter	PDT	digital

ZK2576

HOW TO ORDER ADDITIONAL DOCUMENTATION

In Continental USA and Puerto Rico call 800-258-1710

In New Hampshire, Alaska, and Hawaii call 603-884-6660

In Canada call 613-234-7726 (Ottawa-Hull)
800-267-6146 (all other Canadian)

DIRECT MAIL ORDERS (USA & PUERTO RICO)*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire 03061

*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575)

DIRECT MAIL ORDERS (CANADA)

Digital Equipment of Canada Ltd.
940 Belfast Road
Ottawa, Ontario K1G 4C2
Attn: A&SG Business Manager

DIRECT MAIL ORDERS (INTERNATIONAL)

Digital Equipment Corporation
A&SG Business Manager
c/o Digital's local subsidiary or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

Contents

Page

Preface vii

Chapter 1 Getting Started

1.1 Sorting Records 1-1
1.1.1 The SORT Command 1-2
1.1.2 Using SORT Qualifiers 1-3
1.1.3 Input and Output File Parameters 1-5
1.2 Merging Records 1-5
1.2.1 The MERGE Command 1-6
1.2.2 MERGE Qualifiers 1-7
1.2.3 Input and Output File Parameters 1-7
1.3 Running BATCH SORT and MERGE 1-7

Chapter 2 Invoking SORT and MERGE at the Command Level

2.1 Using SORT/MERGE from the MCR/CCL Interface 2-1
2.2 Defining Your SORT/MERGE Operations 2-2
2.2.1 Default File Types for DCL 2-3
2.2.2 Default File Names for MCR/CCL 2-3
2.3 Specifying Key Field Information 2-4
2.3.1 Identifying the Position, Size, and Order of a Key 2-4
2.3.2 Multiple Keys 2-5
2.3.3 Data Type and Size 2-6
2.3.4 Specifying Key Information in MCR/CCL 2-7
2.3.5 Collating Sequences 2-8
2.3.6 Options for Equal Keys 2-9
2.4 Defining the Sorting Process 2-10
2.5 Specifying File Attributes 2-12
2.5.1 Specifying Input File Format and Organization 2-12
2.5.2 Optimizing the SORT/MERGE Work Area 2-13
2.5.3 Specifying Shareable Files 2-14
2.5.4 Specifying Output File Organization and Format 2-14
2.5.5 Chaining to a SORT or MERGE Image (RSTS/E only) 2-16
2.6 Merging Files 2-17
2.7 Summary of DCL Qualifiers and MCR/CCL Switches 2-17

Chapter 3 Using a Specification File

3.1 Creating a Specification File 3-1
3.1.1 Entering Text in a Specification File 3-3
3.1.2 Using Qualifiers in the Specification File 3-3
3.2 Specifying Record Fields 3-4
3.2.1 Specifying Keys 3-5
3.2.2 Formatting Data for the Output File 3-6
3.2.3 Defining and Using Conditions 3-7
3.2.4 Specifying Record Selection 3-8
3.3 Sorting Files with More Than One Record Format 3-9
3.4 Special Considerations 3-10
3.5 Specifying a Collating Sequence 3-11
3.5.1 Defining Your Own Collating Sequence 3-11
3.5.2 FOLD and TIE_BREAK 3-12
3.5.3 Modifying the Collating Sequence 3-13
3.5.4 Example of a User-Defined Collating Sequence 3-14
3.6 Reassigning Work Files 3-15
3.7 Specifying a New Pad Character 3-16
3.8 Specification File Summary 3-16

Chapter 4 Using SORT and MERGE in Application Programs

4.1 Language Support 4-1
4.2 The Two Callable SORT and MERGE Interfaces 4-2
4.3 Specifying Your Own Routines 4-3
4.4 Calling the SORT Subroutines 4-3
4.4.1 Passing File Names and Initializing Your Sort 4-5
4.4.2 Passing Records to SORT 4-12
4.4.3 Returning Records to Your Program 4-12
4.4.4 Sorting the Records 4-13
4.4.5 Ending Your Sort 4-14
4.5 Calling the MERGE Subroutines 4-14
4.5.1 Initializing Your Merge 4-16
4.5.2 Summary of SORT Subroutine Calls 4-19
4.6 Task Building 4-21
4.6.1 Overlay Descriptor Language (ODL) files 4-21
4.6.2 Task Building with User-Defined Routines 4-23
4.6.3 LUN Usage 4-25

Chapter 5 Improving SORT Efficiency

5.1 How SORT Operates 5-1
5.1.1 Initialization Phase 5-2
5.1.2 Sort Phase 5-2
5.1.3 Merge Phase of the SORT 5-4
5.1.4 Cleanup Phase 5-4
5.2 Understanding and Using SORT/MERGE Statistics 5-4
5.2.1 Using Statistics with Callable SORT/MERGE 5-6

5.3 What the User Can Do 5–7
5.3.1 Work Files 5–8
5.3.2 Input File Allocation 5–9
5.3.3 Output File Preallocation 5–9
5.3.4 Process 5–10
5.4 What the System Manager Can Do 5–10

Appendix A Error Messages

Appendix B Sample Programs

BASIC-PLUS-2 Program Using the MERGE File Interface B–2
BASIC-PLUS-2 Program Using Both SORT and MERGE Mixed-Mode Interfaces B–3
COBOL-81 Program Using the MERGE Record Interface B–6
COBOL-81 Program Using the SORT Record Interface B–8
FORTRAN Program Using the MERGE File Interface B–11
FORTRAN Program Using the SORT File Interface B–13

Appendix C Specification File Translator

C.1 Converting Version 2 Specification Files C–1
C.2 Translator Error Messages C–1

Appendix D Multinational Collating Sequence

Appendix E Compatibility with PDP–11 SORT/MERGE Version 2

Index

Tables

2–1 SORT/MERGE — DCL Qualifiers, MCR/CCL Switches 2–2
2–2 SORT Processes 2–10
2–3 SORT Qualifiers in a DCL Command Line 2–18
2–4 SORT Switches in an MCR/CCL Command Line 2–20
2–5 MERGE Qualifiers in a DCL Command Line 2–22
2–6 MERGE Switches in an MCR/CCL Command Line 2–24
3–1 Specification File Keywords 3–2
3–2 Specification File Summary 3–16
4–1 SORT Subroutines 4–4
4–2 SRTINI Parameters 4–6
4–3 SRTRLS Parameters 4–12
4–4 SRTRTN Parameters 4–13
4–5 MERGE Subroutines 4–15
4–6 MRGINI Parameters 4–17
4–7 Summary of SORT Subroutine Calls for File Interface 4–19
4–8 Summary of SORT Subroutine Calls for Record Interface 4–19
4–9 Summary of SORT Subroutine Calls for File Interface Input, Record Interface Output 4–20
4–10 Summary of SORT Subroutine Calls for Record Interface Input, File Interface Output 4–21
4–11 SORT/MERGE ODL Files 4–22

Preface

How to Use This Manual

The *PDP-11 SORT/MERGE User's Guide* describes how to use the SORT and MERGE utilities for RSTS/E, RSX-11M, RSX-11M-PLUS, and Micro/RSX operating systems. You should use this manual if you want to sort the data in files, or if you want to combine two or more data files. This manual describes simple and complex applications of SORT and MERGE.

The manual has five chapters and five appendixes:

- Chapter 1 is an introduction to SORT and MERGE; it shows some elementary examples of using SORT/MERGE at DCL level.
- Chapter 2 is a discussion of using SORT and MERGE in DCL and MCR/CCL command lines, and it describes all of the qualifiers that you can use to customize your sort or merge operation.
- Chapter 3 discusses the use of specification files with SORT or MERGE. A specification file allows you to define parameters for a sort or merge operation, and it also allows you to perform some functions that are unavailable in a simple command line SORT or MERGE.
- Chapter 4 shows how to use SORT and MERGE in an application program, discussing the callable subroutines that are included in the SORT/MERGE Utility.
- Chapter 5 discusses optimization techniques, describing how you can improve system performance depending on the sort or merge operations that you use.
- Appendix A lists the SORT and MERGE error messages.
- Appendix B lists six sample application programs that use the callable SORT and MERGE subroutines.
- Appendix C discusses the translator utility, which you use to convert SORT/MERGE Version 2 specification files to files that are compatible with Version 3.
- Appendix D lists the MULTINATIONAL collating sequence.
- Appendix E lists changes between this version of PDP-11 SORT/MERGE and the previous version.

Conventions Used in This Manual

The following conventions are used throughout this manual to designate required and optional syntax:

[ ]	Optional; you may select none, one, or all of the choices
{ }	Select one and only one of the choices
( )	Enclose the choices that you select in parentheses
...	A horizontal ellipsis in syntax formats means that you can repeat the preceding element
⋮	A vertical ellipsis in syntax formats means that there can be additional syntax statements between the listed elements
red	Syntax shown in red can be entered interactively

Chapter 1
Getting Started

PDP-11 SORT/MERGE provides you with a fast and flexible means of reordering and combining data files, which are important functions in many data processing applications. For example, suppose you maintain your employee records online; depending on the application, you may want to sort the data alphabetically by employee name, numerically by department number or by a sales performance measurement. You may also want to combine, or merge, two or more of your data files into a single file. PDP-11 SORT/MERGE provides you with a fast and easy way to sort the individual files and/or combine them into a single ordered file.

This chapter introduces the PDP-11 SORT and MERGE utilities as invoked at DCL command level, shows the DCL syntax required for simple sort and merge operations, and describes how to run SORT and MERGE in batch mode (for RSTS/E and RSX-11M-PLUS only).

1.1 Sorting Records

PDP-11 SORT reads records from as many as ten input files, sorts them according to the field(s) you specify, and generates one reordered output file. For example, assume that you have a file named MAGSUB.DAT that contains the data records of a magazine subscription list. There is one record for each subscriber; each record has five fields:

- Subscriber's name
- Street address
- City
- State
- Expiration date of the subscription

```
                                                        Exp Date
Name                Street                City          State   (Yr/Mo/Da)

Yellen Mark         90 Lynwood Lane       Westfield     MA      841231
Germont Alfredo     15 Town House Dr      Waltham       MA      850501
Thompson Lynda      395 N Main St         Easton        MA      831130
Fallon Curtis       56 Juniper Lane       Lenox         MA      841101
```

Name	Street	City	State	Exp Date (Yr/Mo/Da)
Tosca Floria	108 Winfield Dr	Rome	NY	850630
Weaver Stephen	72 Newton Ave	Hyde Park	NY	840509
Marsh Beverly	305 Cambridge St	Pittsfield	MA	851015
Barber Robert	81 River St	Belmont	NY	841031

Suppose that you want to create a new file with the subscription records ordered alphabetically by subscriber's name and that you want the new file to be called MAGSUB.LIS. The field containing the subscriber's name is your sort field, or *key*. Use SORT to collate this file in the order that you want with the DCL level *SORT command line*. The SORT command line specifies information specific to the type of sorting operation that you want to perform, the name of the file that you want to sort (the *input* file), and the name of the file to which the sorted data is to be written (the *output* file).

For example, in the simple list of magazine subscribers shown above, where the input file is MAGSUB.DAT, the output file is to be called MAGSUB.LIS, and you want to accept all of the default parameters that SORT supplies, the DCL command line is

```
$ SORT MAGSUB.DAT MAGSUB.LIS
```

The command line for this sort operation does not give any information about the key field because the key field (subscriber's name) begins in the first position of the record. When you sort the file using the default of entire record in ascending alphabetical order (A to Z), the new file, MAGSUB.LIS, now looks like this:

Name	Street	City	State	Exp Date (Yr/Mo/Da)
Barber Robert	81 River St	Belmont	NY	841031
Fallon Curtis	56 Juniper Lane	Lenox	MA	841101
Germont Alfredo	15 Town House Dr	Waltham	MA	850501
Marsh Beverly	305 Cambridge St	Pittsfield	MA	851015
Thompson Lynda	395 N Main St	Easton	MA	831130
Tosca Floria	108 Winfield Dr	Rome	NY	850630
Weaver Stephen	72 Newton Ave	Hyde Park	NY	840509
Yellen Mark	90 Lynwood Lane	Westfield	MA	841231

1.1.1 The SORT Command

The SORT command line at the DCL interface consists of three parts, separated by spaces.

```
$ SORT[/qualifiers] input-file[/qualifiers] output-file[/qualifiers]
```

SORT is the DCL command that invokes the PDP-11 SORT utility. The SORT command accepts qualifiers that describe the key(s), specify sort options, or describe other aspects of the sort operation or the data that you are using. The SORT command requires two parameters that identify the input file(s) and the output file.

The input file parameter identifies the file or files that you want to sort. SORT allows you to sort up to 10 input files (in a single operation) into one output file. If you have more than 10 files to sort, you can break them into

smaller groups, sort each of the groups, and then merge the output files into a single output file. If you do not supply an input file, SORT prompts you for the input file name.

The output file parameter identifies the name of the file that SORT creates; this file will contain the sorted data from the input file or files. You identify various characteristics for the output file by using qualifiers, as described later in this chapter. You can specify only one output file; if you do not supply an output file, SORT prompts you for the output file name.

You use the qualifiers with one, two, or all of the parts of the SORT command line when your particular application requires that you override the defaults that are supplied with SORT. The default values are established when SORT is installed on your system.

PDP-11 SORT supports all RMS-11 files.

1.1.2 Using SORT Qualifiers

The first element of the command string is the command SORT. The SORT command is followed by the qualifier(s) appropriate to your sorting task. Qualifiers always begin with a slash (/) and are placed after the SORT command. These qualifiers let you identify your key fields, describe the data in these fields, and specify various sort options.

Subqualifiers provide additional information about a qualifier. Use an equal sign (=) between the qualifier and its subqualifiers. If you use more than one subqualifier, enclose the subqualifiers in parentheses and separate them by commas. When a subqualifier takes a value, separate the subqualifier from its value with a colon. You can use an equal sign in place 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.

In the sorting operation on the magazine subscription list, no qualifiers were used. Suppose, however, that you want to sort the magazine subscription data based on the expiration date. Since the field that identifies the expiration date is other than the default field, you must use the /KEY qualifier as well as two of its subqualifiers, POSITION and SIZE. In addition to the required POSITION and SIZE subqualifiers, the /KEY qualifier accepts several optional subqualifiers, which are used to define your key fields for the SORT utility.

Look at the fields in the following file, which is named MAGSUB.DAT. The numbers in parentheses indicate the position of each field in the record. For example, the street field begins at position 20 and occupies positions 20 through 39. The field representing the expiration date begins at position 60 and is 6 characters (bytes) long:

```
Name              Street              City         State   Exp Date
(1-19)            (20-39)             (40-51)      (52-59) (60-65)

Yellen Mark       90 Lynwood Lane     Westfield    MA      841231
Germont Alfredo   15 Town House Dr    Waltham      MA      850501
Thompson Lynda    395 N Main St       Easton       MA      831130
Fallon Curtis     56 Juniper Lane     Lenox        MA      841101
```

```
Name                Street              City         State   Exp Date
(1-19)              (20-39)             (40-51)      (52-59) (60-65)

Tosca Floria        108 Winfield Dr     Rome         NY      850630
Weaver Stephen      2 Newton Ave        Hyde Park    NY      840509
Marsh Beverly       305 Cambridge St    Pittsfield   MA      851015
Barber Robert       81 River St         Belmont      NY      841031
```

To sort MAGSUB.DAT based on expiration date, use the command line

```
$ SORT/KEY=(POSITION:60,SIZE:6) MAGSUB.DAT MAGEXP.LIS
```

The POSITION subqualifier (POSITION:60) identifies the position of the first character in the key field, and the SIZE subqualifier (SIZE:6) gives the length of this key field. When you use more than one subqualifier, separate them with commas. Note that the first byte in a record is considered position 1.

As shown below, the records in the output file created from this sort operation are listed in order of their expiration dates.

```
Name                Street              City         State   Exp Date

Thompson Lynda      395 N Main St       Easton       MA      831130
Weaver Stephen      72 Newton Ave       Hyde Park    NY      840509
Barber Robert       81 River St         Belmont      NY      841031
Fallon Curtis       56 Juniper Lane     Lenox        MA      841101
Yellen Mark         90 Lynwood Lane     Westfield    MA      841231
Germont Alfredo     15 Town House Dr    Waltham      MA      850501
Tosca Floria        108 Winfield Dr     Rome         NY      850630
Marsh Beverly       305 Cambridge St    Pittsfield   MA      851015
```

Notice that the expiration dates are now in ascending order.

A *primary key* is the first field on which your records are sorted. That is, SORT reorders your records based on the information in the field that you choose as the primary key. However, if some of your records have the same data for the primary key, you may want to specify a *secondary key*. The secondary key is used to sort those records with identical primary keys. For example, assume that the primary key for MAGSUB.DAT is the state field. There are five records with the data "MA" and three records with the data "NY". To sort the records alphabetically by state, and then alphabetically by name (for subscribers within the same state), you specify the name field as the secondary key.

```
$ SORT/KEY=(POSITION:52,SIZE:8)/KEY=(POSITION:1,SIZE:19) -
_$ MAGSUB.DAT MAGEXP.LIS
```

As shown below, the records are listed alphabetically for Massachusetts subscribers, then alphabetically for New York subscribers.

```
Name                Street              City         State   Exp Date

Fallon Curtis       56 Juniper Lane     Lenox        MA      841101
Germont Alfredo     15 Town House Dr    Waltham      MA      850501
Marsh Beverly       305 Cambridge St    Pittsfield   MA      851015
Thompson Lynda      395 N Main St       Easton       MA      831130
Yellen Mark         90 Lynwood Lane     Westfield    MA      841231
Barber Robert       81 River St         Belmont      NY      841031
Tosca Floria        108 Winfield Dr     Rome         NY      850630
Weaver Stephen      72 Newton Ave       Hyde Park    NY      840509
```

In Chapter 2, you will learn more about the SORT command qualifiers and how they are used to specify information to the SORT utility about your sort operation. Chapter 2 also explains how to specify multiple keys and describes the defaults for qualifiers and subqualifiers. Chapter 5 describes the command qualifiers that you can use for optimization.

1.1.3 Input and Output File Parameters

You can sort up to 10 input files using PDP–11 SORT; all sort operations generate a single output file. Use the input file parameter and the output file parameter to identify the input file(s) that you want to use and the output file that you want to create. Use qualifiers to specify file attributes for the input and output files: list the input file qualifier(s) immediately after the input file specification(s) and the output file qualifier(s) after the output file specification.

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 file named SALES.DAT (assuming that you accept all of the default qualifiers), use the following command line:

```
$ SORT SALES1.DAT,SALES2.DAT,SALES3.DAT SALES.DAT
```

Chapter 2 describes the qualifiers that you can use with the input and the output file parameters to define your file attributes. Chapter 5 describes the output file qualifiers used in optimizing sort operations.

1.2 Merging Records

You can use the MERGE utility to combine files that have already been sorted on the same key or keys. The MERGE utility accepts up to 10 input files and combines them according to the key field(s). As with SORT, MERGE always generates a single output file. All of the input files to MERGE should previously be sorted according to the same key(s) that you specify for the merging operation. You can also use MERGE to determine whether or not a file has already been sorted according to a specified set of keys.

For example, the following lists represent two files named EREPRT.LIS and WREPRT.LIS, containing reports from two plants in a manufacturing company:

```
DATE      PLANT     PART      QTY       QTY       PCT
          CODE      NUM       MANUF     REJECTED  USABLE

821109    E         0275      1000      37        96.3
821109    E         7820      1200      28        97.6
821109    E         2064       800      12        98.5
821109    E         4016       950      11        98.8
821109    E         3198      1500      11        99.3

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

Each of the records in these files contains six fields (DATE, PLANT CODE, PART NUMBER, QUANTITY MANUFACTURED, QUANTITY REJECTED, AND PERCENT USABLE). Each of the two files has been sorted on the field containing the percent usable figure; this field begins in position 40 and is 4 characters long.

Suppose you want to merge these files based on the "Percent Usable" field. The command to merge the two files is:

```
$ MERGE/KEY=(POSITION:40,SIZE:4) EREPRT.LIS,WREPRT.LIS MRGRPT.LIS
```

The command line format for MERGE is identical to that for SORT, except that the command keyword is MERGE. In this example:

- The /KEY qualifier uses the two required subqualifiers (*POSITION:40* to denote the location of the key field and *SIZE:4* to denote the length of the key field)
- The two input files (which, as with SORT, are separated by commas) are EREPRT.LIS and WREPRT.LIS
- The name of the output file that MERGE creates is MRGRPT.LIS

The merged file created with this command string, MRGRPT.LIS, now looks like this:

```
DATE      PLANT     PART      QTY       QTY       PCT
          CODE      NUM       MANUF     REJECTED  USABLE

821109    E         0275      1000      37        96.3
821109    W         3198      1300      33        97.4
821109    E         7820      1200      28        97.6
821109    W         2064       700      13        98.1
821109    E         2064       800      12        98.5
821109    W         7820      1800      25        98.6
821109    W         0275      1650      21        98.7
821109    E         4016       950      11        98.8
821109    W         4016      1400      14        99.0
821109    E         3198      1500      11        99.3
```

1.2.1 The MERGE Command

The MERGE command line at the DCL interface consists of three parts, separated by spaces.

```
$ MERGE[/qualifiers] input-file[/qualifiers] output-file[/qualifiers]
```

MERGE is the DCL command that invokes the PDP-11 MERGE utility. The MERGE command accepts qualifiers that describe the key(s), specify options, or describe other aspects of the merge operation or the data that you are using. The MERGE command requires two parameters that identify the input file(s) and the output file.

The input file parameter identifies the file or files that you want to merge. MERGE allows you to merge up to 10 input files (in a single operation) into one output file. The keys must be the same in each of the input files. If you do not supply an input file, MERGE prompts you for the input file name.

The output file parameter identifies the name of the file that MERGE creates; this file will contain the data from the input file or files. You identify various characteristics for the output file by using qualifiers, as described later in this chapter. You can specify only one output file; if you do not supply an output file, MERGE prompts you for the output file name.

You use the qualifiers with one, two, or all of the parts of the MERGE command line when your particular application requires that you override the defaults that are supplied with MERGE. The default values are established when MERGE is installed on your system.

PDP-11 MERGE supports all RMS-11 files.

1.2.2 MERGE Qualifiers

As with SORT, MERGE provides default values for qualifiers; you need specify qualifiers only when you want to override the defaults. In the merge task on the two quality control reports, the /KEY qualifier is required because the key field is other than the entire record. The POSITION and SIZE subqualifiers are required whenever you use the /KEY qualifier.

Chapter 2 describes how the MERGE command qualifiers and subqualifiers are used to specify information to MERGE about your merge operation.

1.2.3 Input and Output File Parameters

MERGE accepts up to 10 presorted input files, each ordered using the same key(s). You must use a comma to separate individual input file specifications, as shown in the command string for the merging operation for the quality control reports:

```
$ MERGE/KEY=(POSITION:40,SIZE:4) EREPRT.LIS,WREPRT.LIS MRGRPT.LIS
```

You can name only one output file in your command string.

Chapter 2 explains the qualifiers that you use with both the input and output file parameters to define your file attributes.

1.3 Running Batch SORT and MERGE

You can run a sort or merge operation as a batch job (on RSTS/E and RSX-11M-PLUS only). Batch processing frees your terminal for other work and is particularly useful when you are running frequent or lengthy sort or merge operations. See the documentation for the operating system that you are using to learn about creating and submitting batch jobs.

Chapter 2
Invoking SORT and MERGE at the Command Level

This chapter describes how to define your reordering operations when you use the SORT or MERGE command in the DCL and MCR/CCL interfaces. The first part of the chapter shows you how to sort data (using SORT); you can follow the same procedures to merge your data files (using MERGE). Later, you learn how to use sequence checking, the option that is unique to MERGE. In addition, this chapter describes the following:

- How to specify key information, including key position and size, the order of your sort, and the way key field data is stored in your records
- How to define the way in which SORT or MERGE processes your keys internally according to what is most appropriate for your application
- How to specify the file attributes for the input and output files

A summary of all DCL qualifiers and MCR/CCL switches for SORT and MERGE is found at the end of this chapter.

When you issue the SORT or MERGE command interactively, you can also use a *specification file* (that you create with a text editor). Using a specification file allows you to use additional SORT features, such as user-defined collating sequences. You learn about using a specification file in Chapter 3. You can also invoke SORT or MERGE from an application program; Chapter 4 describes this procedure.

2.1 Using SORT/MERGE from the MCR/CCL Interface

Chapter 1 showed examples using SORT from the DCL interface. From DCL, you issue the SORT (or MERGE) command, followed by input file information, then output file information. You can also use SORT and MERGE using standard MCR/CCL commands; when you invoke these utilities from the MCR/CCL interface, the system displays the SRT> or MGE> prompts, respectively.

The format for using SORT and MERGE at MCR/CCL level is as follows:

```
SRT> output-file/switches = input-file(s)/switches
```

Note that when you use the MCR/CCL interface, the output file information precedes the input file information and is separated from the input file information by an equal sign. (With the DCL interface, input file information is listed first and is separated from output file information by a space.)

All switches in the MCR/CCL interface begin with a slash and are exactly two letters in length. There are also some negative switches that include a minus sign between the slash and the two-letter mnemonic (for example, /-ST).

2.2 Defining Your SORT/MERGE Operations

PDP-11 SORT/MERGE requires information from you about the type of sort or merge operation that you want to do (for example, ascending or descending order) and the type of data with which you are working (for example, file organization). You provide this information in the command string that you issue when you invoke SORT or MERGE. The SORT/MERGE utility assumes some information by default; however, you must provide other parts of this information explicitly. This chapter describes how to issue SORT and MERGE command strings and the optional and required qualifiers (for DCL) or switches (for MCR/CCL).

Table 2-1 summarizes the DCL qualifiers and the MCR/CCL switches used with SORT or MERGE; the remainder of this chapter describes each of these qualifiers and switches.

Table 2-1: SORT/MERGE — DCL Qualifiers, MCR/CCL Switches

DCL	MCR	Function
/KEY	/KE	Describes key field(s), including position, size, order, and data type
/COLLATING_SEQUENCE	/CS	Specifies one of three predefined collating orders for character key field(s)
/STABLE /NOSTABLE	/ST /-ST	Defines options for equal keys
/NODUPLICATES	/ND	
/PROCESS	/PR	Defines the internal sorting process (SORT only)
/SPECIFICATION	/SF	Identifies a specification file, described in Chapter 3
/STATISTICS	/SS	Displays a statistical summary, primarily for help with optimization, described in Chapter 5
/WORK_FILES	/FI /DE	Used for optimization, described in Chapter 5 (SORT only)
/CHECK_SEQUENCE /NOCHECK_SEQUENCE	/CH /-CH	Verifies that the records in the input files are in order (MERGE only)
/FORMAT	/FO /BK /BL /SI	Required under certain conditions for defining input file characteristics; also used to define output file format and, for optimization, window or clustersize, described in Chapter 5
/SHAREABLE	/SH	Specifies that the input file is to be opened in a write-shareable mode

Table 2-1 (Cont.): SORT/MERGE — DCL Qualifiers, MCR/CCL Switches

DCL	MCR	Function
/INDEXED_SEQUENTIAL	/IN	Defines input or output file organization
/SEQUENTIAL /RELATIVE	/SE /RE	Define output file organization
/OVERLAY	/OV	Specifies that the output file is to be overlaid on, or written to, an existing empty file
/ALLOCATION /CONTIGUOUS /BUCKET_SIZE /LOAD_FILL /TREE_SPACE	/AL /CO /BU /LO /PT	Used for optimization, described in Chapter 5
(Not available in DCL)	/CN	Identifies an image to chain to (RSTS/E only)

2.2.1 Default File Types for DCL

Default file types are file types that SORT or MERGE provides when you do not specify a file type in your command line. For the DCL interface, the default file type for SORT and MERGE input files is .DAT. The default file type for the output file is .DAT, regardless of the file type of the input file(s). For example, suppose you use the command line

```
$ SORT INPUT OUTPUT
```

SORT looks in your default directory for the file INPUT.DAT and writes the sorted output file to a file named OUTPUT.DAT.

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

```
INPUT FILE?
```

If you do not provide an output file specification, SORT prompts you with

```
OUTPUT FILE?
```

2.2.2 Default File Names for MCR/CCL

For the MCR/CCL interface, there is a default file name when you have only one input file for your sort or merge operation: SRT.DAT. The default file specification for the output file is OUT.DAT. If you do not specify a file type, the default file type is .DAT.

When you use the MCR/CCL interface and enter one or more characters after the SRT> or MGE> prompt followed by the <RETURN> line terminator, SORT or MERGE substitutes default values for any missing values in the command expression. For example, if you enter only an equal sign then press <RETURN> at the SRT> prompt, SORT substitutes the default value of SRT.DAT for an input file and OUT.DAT for the output file.

2.3 Specifying Key Field Information

You are not required to specify key information for a sorting operation. By default, SORT reorders a file by sorting up to the first 255 bytes of a record's length, in ascending order, and assuming character data. However, if the field that you want to use as the key does not start in the first position or does not contain character data, you must then indicate to SORT or MERGE the position and size of the key. The following sections show how to identify keys to SORT/MERGE.

2.3.1 Identifying the Position, Size, and Order of a Key

You identify a field as being a key (on which to sort records) by giving the position of the field. The position of the key is determined by counting, inclusively, the number of bytes between the beginning of the record and the first byte in the field. For example, the following file named YRSALE.DAT contains employee sales records with quarterly and yearly sales totals (in thousands of dollars). The number ranges in parentheses indicate the positions of each of the fields in the records. The first byte of each record is considered to be POSITION=1. The blanks represent spaces. (Note that if you use the TAB character in a data file, it is treated as a single ASCII character.)

The field representing store begins in position 1, the field representing department begins in position 7, the field representing the employee's name begins in position 11, and so on.

```
                                    SALES REPORT

STORE    DEPT   NAME                 Q1        Q2        Q3        Q4        TOTAL
(1-2)    (7-8)  (11-35)              (36-38)   (44-46)   (52-54)   (60-62)   (68-71)
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 Stephan        7.1       7.0       6.9       7.1       28.1
2E       A1     Jamieson Robert      6.7       6.6       6.8       6.7       26.8
1E       A1     Albertson Ronald     6.9       6.7       6.8       6.9       27.2
```

Suppose you are to sort these records by the sales employees' names. The first byte in this key field is in position 11. The range of the field extends from positions 11 through 35, so the key field's size is 25. The size of the field is the number of bytes allocated to the field in the record.

With DCL, you could sort this file into an output file named SRTLIS.DAT using the following SORT command line.

```
$ SORT/KEY=(POSITION:11,SIZE:25) YRSALE.DAT SRTLIS.DAT
```

Remember to separate subqualifiers (POSITION,SIZE) with a comma.

To accomplish the same output using the MCR/CCL interface, you would use this command:

```
SRT> SRTLIS.DAT=YRSALE.DAT/KE:11.25
```

With MCR/CCL, the KEY switch (/KE:11.25) is part of the input file parameter. In this command line, the two numbers that follow the /KE represent the position of the key field (position 11) and its size (25 bytes). In MCR/CCL, the values for position and size are separated by a decimal point.

You are required to specify both the position and size of a key field whenever you use the /KEY qualifier or /KE switch (except that you *cannot* use SIZE with D_FLOATING or F_FLOATING data types in DCL). Later in this chapter, you learn how to specify the sizes of data types other than character.

The file SRTLIS.DAT from this sorting operation contains the list of sales records with the employees' names rearranged alphabetically. Alphabetical order (A to Z) or ascending numeric order (lowest to highest number) is the default.

Suppose, however, you want to sort in descending order on the numeric field in YRSALE.DAT that contains the total sales figures. Then you must specify the order of your sort.

To specify a descending sort order in DCL, use the command line

```
$  SORT/KEY=(POSITION:68,SIZE:4,DESCENDING) YRSALE.DAT SRTLIS.DAT
```

In MCR/CCL, the command line specifying a descending sort order is

```
SRT>  SRTLIS.DAT=YRSALE.DAT/KE:O68.4
```

The letter "O" in "KE:O68.4", represents "opposite" and means 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. Remember that you must indicate the sort order before you list the numbers indicating position and size.

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

Thus far, most examples have shown operations that sort on only one key field. The next section explains how to specify multiple keys — keys that are primary, secondary, and so on.

2.3.2 Multiple Keys

You can specify up to 16 key fields in a sorting operation, with a total key size of up to 512 bytes. If you do select multiple keys, decide which is the primary key, which is the secondary key, and so on, and then list them in the command string in the order of their priority.

For example, suppose you want to arrange the sales records in YRSALE.DAT from the highest to the lowest total sales for each department within each store. Your primary key is the store field, the secondary key is the department field, the third-level (tertiary) key is the total sales field, and you want the sorted data to be written to a file named KEYS3.DAT.

With DCL, you specify this information to SORT as follows:

```
$  SORT/KE=(PO:1,SIZ:2)/KE=(PO:7,SIZ:2)/KE=(PO:68,SIZ:4,DESC) -
_$  YRSALE.DAT KEYS3.DAT
```

As this example shows, you can abbreviate qualifiers and subqualifiers as long as the abbreviations are unique.

For each sort key, you must use a separate /KEY qualifier. If SORT finds key subqualifiers repeated after a single /KEY qualifier, it does not treat these as specifications for multiple keys. Instead, the subqualifiers that are most recently encountered override previous subqualifiers.

Using the MCR/CCL interface, the command line to accomplish the same operation is

```
SRT> KEYS3.DAT=YRSALE.DAT/KE:1.2:7.2:068.4
```

As this example shows, in MCR/CCL 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 /KE.

After this multiple-key operation is completed, the sorted sales records appear as follows in KEYS3.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 Stephan       7.1   7.0   6.9   7.1   28.1
2E      A1     Jamieson Robert     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
```

Thus far, the data type used in all the sorting examples has been character (text). It has not been necessary to specify the data type in the examples because character is the default data type. There are other ways, however, in which data can be represented. The procedure used to create your input file(s) determines the data type of each field in the records.

2.3.3 Data Type and Size

You must specify the data type if the data in your key field(s) is not stored as character data type. The data types recognized by PDP-11 SORT/MERGE with DCL are

ASCII_FLOATING
ASCII_ZONED
BINARY[,SIGNED]
BINARY,UNSIGNED
[CHARACTER]
DECIMAL[,TRAILING_SIGN,OVERPUNCHED_SIGN]
DECIMAL,LEADING_SIGN,OVERPUNCHED_SIGN
DECIMAL,LEADING_SIGN,SEPARATE_SIGN
DECIMAL,TRAILING_SIGN,SEPARATE_SIGN
DECIMAL,UNSIGNED
DIBOL_ZONED (decimal, trailing overpunched sign)
D_FLOATING
F_FLOATING
PACKED_DECIMAL

The data types enclosed in brackets are the defaults. For example, if you do not specify a data type for your key field, SORT/MERGE assumes character data type. If the data type in your key field is binary and signed, you can specify BINARY, since BINARY,SIGNED is the default. However, if your key data type is binary and unsigned, you must specify BINARY, UNSIGNED.

If your key data type is decimal and its sign is trailing and overpunched (the default), you may specify DECIMAL only. If your key data type is decimal but unsigned, you must specify DECIMAL, UNSIGNED.

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

For example, the numeric fields in the file YRSALE.DAT contain information that is stored as decimal unsigned. To specify a descending sort operation on the field with the total sales figures, you would use the following command line:

```
$ SORT/KEY=(POSITION:68,SIZE:4,DESCENDING,DECIMAL,UNSIGNED) -
_$ YRSALE.DAT SRTLIS.DAT
```

For MCR/CCL the data types are

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)

The numeric fields in the file YRSALE.DAT contain information that is stored as decimal unsigned. To specify a descending sort operation on the field with the total sales figures, use the following command line:

```
SRT> SRTLIS.DAT=YRSALE.DAT/KE:DO68.4
```

2.3.4 Specifying Key Information in MCR/CCL

The order for specifying key information in MCR/CCL is as follows:

1. Data type. In the example shown above, the data type is unsigned decimal, represented by 'D' (KE:DO68.4).
2. Sort order. In the example, the order is descending, represented by 'O', (KE:DO68.4).

3. Position, specified as a decimal number (KE:DO68.4).
4. Size, separated from position by a decimal point and specified as a decimal number (KE:DO68.4).

In MCR, you must always specify the size of the key field. The size specification, given in bytes, depends on the data type.

The following rules apply to the size of a key field for specific data types:

- For CHARACTER data type, the size cannot exceed 255 bytes
- For BINARY data type, you must specify size as 1, 2, 4, or 8 bytes
- For DECIMAL data type, the maximum is 31 bytes; if the sign of a decimal number is stored as a separate byte, that byte must be included in the count of the size
- For floating-point data types, the size must be either 4 or 8

2.3.5 Collating Sequences

SORT arranges characters in ASCII sequence by default. SORT also allows you to use either the 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 multinational character set. You can specify an ASCII collating sequence, which could be used in a command line to override a collating sequence defined in a specification file. (The use of specification files is discussed in Chapter 3.)

For DCL, choose one of the following command qualifiers to specify a collating sequence.

```
/COLLATING_SEQUENCE=ASCII
/COLLATING_SEQUENCE=EBCDIC
/COLLATING_SEQUENCE=MULTINATIONAL
```

For MCR/CCL, use one of the following qualifiers as an input file switch to specify a collating sequence.

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

Note that when you specify the EBCDIC sequence, the characters remain in ASCII representation; only their order is changed.

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. (Á , Ä , A collate as A)
- Lowercase characters are given the collating value of their uppercase equivalents. (a collates as A, ä collates as Ä)
- 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

Be careful when you use the MULTINATIONAL collating sequence for records and files that will be processed at a later time. 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 D lists the multinational collating sequence.

2.3.6 Options for Equal Keys

Your input file(s) may contain records with equal keys. These records will be grouped together in the output file, and, by default, their sorted order (with respect to each other) will be unpredictable. However, you can use the STABLE option, which causes records with equal keys to be directed to the output file in the order in which they were input to SORT. If you specify a stable sort when sorting multiple input files, on output, records in the first file with equal keys will precede those from the second file, and so on.

With DCL, use the command qualifier /STABLE. The default condition for DCL is /NOSTABLE.

With MCR/CCL, use the input file switch /ST. The default condition for MCR/CCL is /-ST.

As an alternative to the STABLE option, you can use the NODUPLICATES option to retain only one record when you have records with equal keys. If SORT/MERGE encounters duplicate records when you have specified the /NODUPLICATES option during a sort or merge operation, the record that is kept (from among the duplicates) is the first record that SORT/MERGE encounters. You cannot use both the NODUPLICATES option and the STABLE option in the same sort.

If you want to specify which of the duplicate records SORT/MERGE is to keep, you should use SORT/MERGE in an application program, write your own equal-key routine, and 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 named SRTCLB. Chapter 4 of this manual discusses calling SORT from an application program.

The default condition is /DUPLICATES. To specify the NODUPLICATES option in DCL, use the command qualifier /NODUPLICATES.

With MCR/CCL, use the input file switch /ND.

2.4 Defining the Sorting Process

SORT provides four processing methods for sorting your data: record, tag, address, or index sort. Table 2-2 describes these processes.

Table 2-2: SORT Processes

Process	Input Device	Output Device	Processing Notes
Record	Any RSTS/E or RSX-11M/M-PLUS input device	Any RSTS/E or RSX-11M/M-PLUS output device	Keeps record intact throughout sort. Output file contains complete records.
Tag	Disk only	Any RSTS/E or RSX-11M/M-PLUS output device	Sorts only key(s), then reaccesses input file records to create output file. Output file contains complete records.
Address	Disk only	Any device that accepts binary data	Sorts only key(s). Output file contains only binary RFAs, and file number when there is more than one input file.
Index	Disk only	Any device that accepts binary data	Sorts only key(s). Output file contains only binary RFAs, file number when there is more than one input file, and keys.

Record sort is the default process.

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

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

With the MCR/CCL interface, specify the sort process using one of the following as an input file switch:

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

None of the examples in this chapter has specified the sort process, so, by default, each has used record sort. Suppose, however, you wanted to use TAG sort for the descending sort operation on the YRSALE.DAT file. With the DCL interface, your command line would be

```
$ SORT/KEY=(PO:68,SIZ:4,DESC)/PROCESS=TAG YRSALE.DAT SRTLIS.DAT
```

With the MCR/CCL interface, your command line would be

```
SRT> SRTLIS.DAT = YRSALE.DAT/KE:068.4/PR:T
```

There are several criteria to consider when you select a sort process. The most important consideration is your use of the output file:

Record and *Tag* sort generate files that contain the entire records. These reordered files are ready to be printed and distributed, or stored.

Address sort creates a list of pointers to the records in the input file. This list takes the form of binary record's file addresses (RFAs). (See the *RMS-11 User's Guide* and *RMS-11 MACRO-11 Reference Manual* for more information about RFAs.) A program can use this list of pointers to access records in the rearranged order for further processing or later output.

Index sort creates an output file that contains both RFAs and key fields. The output format of these key fields is the same as the input format. If your program needs key field content for a decision during future processing, select index sort rather than address sort. (Do not confuse the index sort process with RMS indexed file organization.)

You may need to reorder records from one file in several ways for different purposes. Instead of storing several sorted versions from the same file, each with complete records, you can store several output files from address or index sort in much less space. Then use these files to access the records in the main file in the sorted order you want.

Another important consideration in determining your sort process is the amount of temporary storage space that you have available for the sort:

Tag, address, and index sort use relatively less temporary storage space than record sort. Because record sort keeps the record intact during the sort, it can take up much more work space when your files are large.

In general, the smaller the total key field size is with respect to the total input record size, the greater the saving in temporary storage space.

Your input and output devices can also determine the sort process that you use. Record sort is the only process that can accept input from cards and magnetic tape as well as disk. Output from address and index sort must go to a device that accepts binary data.

Finally, there are performance (speed) differences among the sort processes. Generally, address and index sort are the fastest processes. However, if you plan to retrieve the sorted records in order at some point in your operation, record sort will usually be the fastest process. Tag sort moves only keys instead of complete records, so it can be faster than record sort when record size is very large and key size is small. Tag sort can also be faster for extremely large files and devices with fast seek times. 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.

In the examples shown so far, the process selected for the sample sorting operations on the file YRSALE.DAT has been either record or tag sort. Record sort is usually the most appropriate choice if you want to print out your reordered file, if your record size is not large, and if there is adequate temporary storage space.

If, instead of printing the sorted records, you want to use them in an application program to calculate, for example, statistics, then index sort would be the appropriate process.

The format of the record generated by SORT depends upon the sort process that you use. Index and address sorts produce a record which has a 1-word input file number (only if there is more than one input file) and a 3-word binary record file address (RFA). For an index sort, the sort keys in the output file appear exactly as they were in the input file (that is, they are not converted to an internal sort format).

2.5 Specifying File Attributes

There are certain circumstances in which you are required to specify file attributes and others in which you may choose to change file attributes. This section explains how to describe 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.

2.5.1 Specifying Input File Organization and Format

If you are sorting file(s) not residing on disk or standard ANSI magnetic tape, then you must specify the size of the longest record in your file(s) as well as the size of your file(s). The record size that you specify will override the size defined in the file header or label.

You specify the record size for a file by giving the longest record length (LRL) in bytes. The longest record length allowed for various file organizations is

File Organization	Maximum 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 LRL is the length of the longest record among all of the files.

At the DCL interface, you specify the record size of the input file's format using the following input file qualifier:

```
/FORMAT=(RECORD_SIZE:n)
```

With MCR/CCL, you must specify the record format when you specify the record size. Use one of the following input file switches:

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

You can abbreviate record format values to one letter. Version 3 of PDP-11 SORT/MERGE supports the STREAM syntax for compatibility with Version 2; here, STREAM and RMS_STREAM have the same meaning. You must specify the record size if the longest record length (LRL) cannot be obtained from RMS (that is, if the file was created by a version of RMS that does not include the LRL information).

You specify file size in blocks. SORT uses file size information to estimate the file size of the temporary file(s) used for the sort operation. Maximum file size accepted is 4,294,967,295 or (2^{32}–1) blocks. For multiple input files, the size is the total size of all files. If the file size is not provided by either you or RMS, SORT allocates 1000 blocks by default.

In DCL, specify input file size with the input file qualifier /FORMAT=FILE SIZE:n.

With MCR/CCL, specify input file size with the input file switch /BK:n.

If the file is neither on disk nor ANSI magnetic tape, you must specify both record and file size.

For example, the DCL command line for a descending sort on the sales figures in YRSALE.DAT would be

```
$ SORT/KE=(PO:68,SIZ:4,DESC) -
_$ YRSALE.DAT/FORMAT=(RECORD_SIZE:71,FILE_SIZE:3) SRTLIS.DAT
```

The same command line in MCR/CCL is

```
SRT> SRTLIS.DAT = YRSALE.DAT/KE:068,4/FO:V:71/BK:3
```

If the organization of your input file(s) is indexed-sequential, then you must specify this organization to SORT. Also, you can indicate the number of keys in the indexed file; if you do not specify this information, the number of keys will default to 1. You should be sure to indicate the number of keys to ensure that SORT/MERGE allocates sufficient RMS space.

In DCL, specify indexed-sequential organization with the input file qualifier /INDEXED_SEQUENTIAL[=n], where n equals the number of keys.

In MCR/CCL, specify indexed-sequential organization with the input file switch /IN[:N].

2.5.2 Optimizing the SORT/MERGE Work Area

By default, SORT/MERGE divides available work area between tree-related data structures and I/O-related data structures in such a way as to ensure the best performance for a typical sort operation. However, there may be some

instances where the I/O requirements of your job require more space than the default provides. The /TREE_SPACE qualifier (in DCL) or /PT switch (in MCR/CCL) lets you 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% to the tree and 45% to I/O. For MERGE, the default division is 30% to the merge list and 70% to I/O. If the majority of the files you are sorting require a large number of I/O data structures (for example, an INDEXED file with many keys, or a consistently large number of input files), you may want to alter the split so that there will be enough room for the I/O requirements. (For example, if you are sorting several indexed files, each having many keys, it may be desirable to allow SORT a smaller tree, thereby allocating more room for RMS-required structures.)

In DCL, specify how you want the work area allocated with the input file qualifier /TREE_SPACE=n, where n is the percentage of work space allocated to the data tree structures.

With the MCR/CCL interface, use the input file switch /PT:n.

2.5.3 Specifying Shareable Files

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

2.5.4 Specifying Output File Organization and Format

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.

A) Use one of the following to specify the file organization:

- DCL (output file qualifier)

  /SEQUENTIAL
  /RELATIVE
  /INDEXED_SEQUENTIAL[=n]

  As with the input file qualifier for indexed-sequential organization for the output file, you can optionally specify the number of keys.

- MCR/CCL (output file switch)

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

  You can optionally specify the number of keys in an output file that is indexed, just as you can for an input file of indexed-sequential organization.

B) Use the /OVERLAY output file qualifier in DCL (/OV switch in MCR/CCL). When you use the /OVERLAY qualifier, the output file already exists (and is empty), and its file organization has been previously defined. If you use the /OVERLAY qualifier, you cannot use any of the file organization qualifiers.

If you specify indexed-sequential organization, SORT assumes that an empty indexed-sequential file of the same name already exists; SORT writes over the existing file. SORT/MERGE does not create an indexed output file if no such file exists. In general, however, to write the sorted records to an existing empty file, you should use the /OVERLAY qualifier.

If you want the output file record format to differ from that of your first input file, then 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 output record format, the format defaults to the (first) input file record format for record or tag sort, and to fixed record format for address or index sort.

In specifying the output record format, you can optionally indicate the maximum record size (in bytes) of the output records. The maximum record size allowed is 32,765 bytes for sequential files, 16,381 for relative files, and 16,369 for indexed files. These totals include control bytes. If you do not specify the size, the default is a length large enough to hold the longest output record.

Indicate the new record format by using the appropriate qualifier.

- DCL (output file qualifier)

```
/FORMAT= { FIXED:n                  }
         { VARIABLE:n               }
         { 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 defaults the value to 2 bytes. If you do not specify FSZ, the default is the maximum sizes 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/CCL

```
/FO: { FIXED:n              }
     { VARIABLE:n           }
     { RMS_STREAM:n         }
     { STREAM:n             }
     { CONTROLLED:n:m       }
```

  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 defaults the value to 2 bytes. If you do not specify this size, the default is the maximum sizes 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.

For both DCL and MCR/CCL, you can truncate record format values to the first letter. The STREAM value exists for compatibility with Version 2 and is the same as RMS_STREAM.

If you direct your output file to magnetic tape, you can specify the file's block size 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/CCL, specify the block size of a file with the output file switch /BL:n.

2.5.5 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/CCL interface with the /CN output switch. Chaining is not available with the DCL interface.

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/CCL command line 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 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 FQNENT field of the FIRQB block. Refer to the *RSTS/E System Directives Manual* for details.

It is also possible to chain into SORT or MERGE from other tasks. To accomplish this, put the sort command line into RSTS/E core common, place the value 30000 in the FQNENT field of the FIRQB, and then chain to the desired SORT or MERGE task file. Refer to 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.

2.6 Merging Files

The MERGE utility allows you to combine up to 10 presorted files. If you use MERGE from DCL or MCR/CCL interface, all your input files must already be sorted on the same key(s). You specify the same key information and file attributes for MERGE as you do for SORT. The prompt for MERGE is MGE>.

In MERGE, however, you do not specify either the sort processes or any work files. Also, there is a function unique to MERGE called *sequence checking* that verifies that your input files are sorted.

You can use qualifiers to indicate explicitly whether or not sequence checking should be done. Use these qualifiers if you do not want sequence checking (to override the default) or if you want to ensure that sequence checking is done (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/CCL, 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 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/CCL 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
```

2.7 Summary Tables of SORT/MERGE Qualifiers and Switches

The tables in this section list the DCL qualifiers and MCR/CCL switches that you can use with SORT or MERGE. All qualifiers and switches are optional. Use the qualifiers and switches in the command line format shown below.

- DCL

```
$ SORT/qualifiers INPUTS.DAT/qualifiers OUTPUT.DAT/qualifiers
$ MERGE/qualifiers INPUTS.DAT/qualifiers OUTPUT.DAT/qualifiers
```

- MCR/CCL

```
SRT> OUTPUT.DAT/switches = INPUTS.DAT/switches
MGE> OUTPUT.DAT/switches = INPUTS.DAT/switches
```

Refer to the Preface for an explanation of the notation used in these tables.

Table 2–3: SORT Qualifiers in a DCL Command Line

Qualifiers	Subqualifiers and Values	Notes
Command Qualifiers		
/KEY	POSITION:1–255	Always include a value for POSITION and a value for SIZE when using /KEY
	,SIZE: 1–255 1, 2, 4, or 8 1–31	For CHARACTER data For BINARY data For DECIMAL data
	[**,ASCENDING**] [,DESCENDING]	
	[**,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]	
/STABLE **/NOSTABLE**		
/DUPLICATES /NODUPLICATES		Do not use both /STABLE and /NODUPLICATES
/PROCESS=	{ **RECORD** } { TAG } { ADDRESS } { INDEX }	
/COLLATING_SEQUENCE=	{ **ASCII** } { EBCDIC } { MULTINATIONAL }	

Table 2-3 (Cont.): SORT Qualifiers in a DCL Command Line

Qualifiers	Subqualifiers and Values	Notes
/STATISTICS **/NOSTATISTICS**		
/WORK_FILES=	(NUMBER:{0,3-10} ,DEVICE:ddnn:[uic,ppn] ,ALLOCATION:1-(2^{32}-1) ,[NO]CONTIGUOUS ,SIZE:1-255)	
/SPECIFICATION=	file-specification	
Input File Qualifiers		
/FORMAT	([FILE_SIZE:1-(2^{32}-1)] ,RECORD_SIZE:1-32767])	Required for files not on disk or magnetic tape Required for files not on disk or magnetic tape, or when LRL is unavailable
/INDEXED_SEQUENTIAL=	1-255	
/SHAREABLE **/NOSHAREABLE**		
/TREE_SPACE=	0-100	
Output File Qualifiers		
/FORMAT=	([FIXED[:1-32767] VARIABLE[:1-32767] RMS_STREAM STREAM CONTROLLED[:1-32767] [,FSZ:1-255] [,BLOCK_SIZE:18-8192])	Use with CONTROLLED records Use with tape files
/SEQUENTIAL /RELATIVE /INDEXED_SEQUENTIAL=	1-255	
/OVERLAY /NOOVERLAY		
/ALLOCATION=	1-(2^{32}-1)	
/CONTIGUOUS /NOCONTIGUOUS		
/LOAD_FILL		Use with INDEXED-SEQUENTIAL files to specify fill factor
/BUCKET_SIZE=	1-32 1-15	For RSX systems For RSTS/E systems Use with disk files

Table 2-4: SORT Switches in an MCR/CCL Command Line

Switch	Values	Notes
Output File Switches		
/AL:	0-(2^{32}-1)	Specifies file allocation in blocks
/BL:	18-8192	Specifies block size for nonstandard magnetic tapes
/BU:	n	Specifies RMS bucket size — 1 to 32 for RSX-11M/M-PLUS, 1 to 15 for RSTS/E
/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)
/CO		Specifies contiguous allocation
/FO:	{CONTROLLED:n FIXED:n RMS_STREAM:n STREAM:n UNKNOWN:n VARIABLE:n}	Specifies record format. Also specifies maximum record size (from 1 to 32767)
/IN	[:1-255]	Requests indexed-sequential file organization. File must already exist and be empty; by default, it is overlaid
/LO		Use with indexed-sequential files to specify fill factor
/OV		Specifies that the output is to be written to a file that already exists and is empty
/RE		Requests relative file organization
/SE		Requests sequential file organization
/SI:	n	Specifies the clustersize for RSTS/E or the retrieval window size for RSX-11M/M-PLUS
Input File Switches		
/AL:	0-(2^{32}-1)	Specifies file size for the initial allocation of work files
/BK:	1-(2^{32}-1)	Specifies file size for files not on disk or magnetic tape
/BL:	18-8192	Specifies block size for nonstandard magnetic tapes
/CO		Specifies contiguous allocation for work files
/CS:	{ASCII EBCIDC MULTINATIONAL}	Specifies collating sequence for character data
/DE:	ddnn:[uic,ppn]	Use to place work files on an alternate device
/FI:	{0 3-10}	Specifies the maximum number of work files to be used
/FO:	{CONTROLLED:n FIXED:n RMS_STREAM:n STREAM:n UNKNOWN:n VARIABLE:n}	Specifies record format. Also specifies maximum record size [from 1 to 32767]

Table 2-4 (Cont.): SORT Switches in an MCR/CCL Command Line

Switch	Values	Notes
Input File Switches		
/IN	[:1-255]	Required for indexed-sequential files. The number of index keys defaults to 1 if n is not specified
/KE:		Specify key field information in this order: data type, sort order, key position, key size
	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
	1-65535	Position
	1-255	Size
/ND		Specifies that the output file will have no duplicate records; may not be used with /ST switch
/PR:		Specifies SORT process:
	{R, T}	Record, Tag
	{A, I}	Address, Index
/PT:n	0-100	Specifies percentage of available work area assigned to SORT/MERGE data structures
/SF		Identifies a specification file
/SH		Specifies that an input file is to be opened as write-shareable; specify for each shareable file
/SI:	n	Specifies RSTS/E clustersize or RSX retrieval window size for work files
/SS		Requests that statistics be written to the default output device
/ST /-ST		Requests a stable sort; use /-ST only to override a stable sort in a specification file

Table 2-5: MERGE Qualifiers in a DCL Command Line

Qualifiers	Subqualifiers and Values	Notes
Command Qualifiers		
/KEY	POSITION:1-255	Always include a value for POSITION and a value for SIZE when using /KEY
	,SIZE: 1-255 1, 2, 4, or 8 1-31	For CHARACTER data For BINARY data For DECIMAL data
	[**,ASCENDING**] [,DESCENDING]	
	[**,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]	
/STABLE **/NOSTABLE**		
/DUPLICATES /NODUPLICATES		Do not use both /STABLE and /NODUPLICATES
/CHECK_SEQUENCE /NOCHECK_SEQUENCE		
/PROCESS=	{ **RECORD** } { TAG } { **ADDRESS** } { INDEX }	
/COLLATING_SEQUENCE=	{ **ASCII** } { EBCDIC } { MULTINATIONAL }	
/STATISTICS **/NOSTATISTICS**		

Table 2-5 (Cont.): MERGE Qualifiers in a DCL Command Line

Qualifiers	Subqualifiers and Values	Notes
Command Qualifiers		
/WORK_FILES=	(NUMBER:{0,3-10} ,DEVICE:ddnn:[uic,ppn] ,ALLOCATION:1-(2^{32}-1) ,[NO]CONTIGUOUS ,SIZE:1-255)	
/SPECIFICATION=	file-specification	
Input File Qualifiers		
/FORMAT	([FILE_SIZE:1-(2^{32}-1)] ,RECORD_SIZE:1-32767)	Required for files not on disk or magnetic tape Required for above files, or when LRL is unavailable
/INDEXED_SEQUENTIAL=	1-255	
/NOSHAREABLE /SHAREABLE		
/TREE_SPACE=	0-100	
Output File Qualifiers		
/FORMAT=	(FIXED[:1-32767] VARIABLE[:1-32767] RMS_STREAM STREAM CONTROLLED[:1-32767] [,FSZ:1-255] [,BLOCK_SIZE:18-8192])	Use with CONTROLLED records Use with tape files
/SEQUENTIAL /RELATIVE /INDEXED_SEQUENTIAL=	1-255	
/OVERLAY /NOOVERLAY		
/ALLOCATION=	1-(2^{32}-1)	
/CONTIGUOUS /NOCONTIGUOUS		
/LOAD_FILL		Use with INDEXED-SEQUENTIAL files to specify fill factor
/BUCKET_SIZE=	1-32 1-15	For RSX systems For RSTS/E systems Use with disk files

Table 2-6: MERGE Switches in an MCR/CCL Command Line

Switch	Values	Notes
Output File Switches		
/AL:	$0-(2^{32}-1)$	Specifies file allocation in blocks
/BL:	18–8192	Specifies block size for nonstandard magnetic tapes
/BU:	n	Specifies RMS bucket size — 1 to 32 for RSX-11M/M-PLUS, 1 to 15 for RSTS/E
/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)
/CO		Specifies contiguous allocation
/FO:	{C:n, F:n, R:n, S:n, U:n, V:n}	Specifies record format — CONTROLLED, FIXED, RMS_STREAM, STREAM, UNKNOWN, or VARIABLE, where n represents maximum record size (from 1 to 32767)
/IN	[:1–255]	Requests indexed-sequential file organization. File must already exist and be empty; by default, it is overlaid
/LO		Use with indexed-sequential files to specify fill factor
/OV		Specifies that the output is to be written to a file that already exists and is empty
/PT:n	0–100	Specifies percentage of available work area allocated to MERGE data structures
/RE		Requests relative file organization
/SE		Requests sequential file organization
/SI:	n	Specifies the clustersize for RSTS/E or the retrieval window size for RSX-11M/M-PLUS
Input File Switches		
/BK:	$1-(2^{32}-1)$	Specifies file size for files not on disk or magnetic tape
/BL:	18–8192	Specifies block size for nonstandard magnetic tapes
/CH /-CH		Specifies sequence checking. Use /-CH if you want to override a request for sequence checking in a specification file
/CS:	{A, E, M}	Specifies collating sequence for character data — ASCII, EBCDIC, or MULTINATIONAL
/FO:	{C:n, F:n, R:n, S:n, U:n, V:n}	Specifies record format — CONTROLLED, FIXED, RMS_STREAM, STREAM, UNKNOWN, or VARIABLE, where n represents maximum record size (from 1 to 32767)
/IN	[:1–255]	Required for indexed-sequential files. The number of index keys defaults to 1 if n is not specified

Table 2-6 (Cont.): MERGE Switches in an MCR/CCL Command Line

Switch	Values	Notes
Input File Switches		
/KE:		Specify key field information in this order: data type, sort order, key position, key size
	A	ASCII Floating
	B	COBOL COMP-6, word signed binary
	C	Character
	D	Decimal, unsigned or trailing, overpunched
	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
	1-65535	Position
	1-255	Size
/ND		Specifies that the output file will have no duplicate records; cannot be used with /ST switch
/SF		Identifies a specification file
/SH		Specifies that an input file is to be opened as write-shareable; specify for each shareable file
/SS		Specifies that statistics be written to the default output device
/ST /-ST		Specifies a stable sort; use /-ST only to override a stable sort in a specification file

Chapter 3
Using a Specification File

A *specification file* is a file you create with a text editor that 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 sorting and merging operations. It can include instructions to accomplish any or all of the following:

- 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

You must use a specification file when you want to do any of the following:

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

You can also use a specification file to reassign work files or to define sort or merge operations that you use frequently. Many of the sort or merge tasks described in Chapter 2 can be done using a specification file.

This chapter describes how to create and use a specification file. A syntax summary and sample specification file are provided at the end of the chapter.

3.1 Creating A Specification File

Use a text editor to create a specification file. Many keywords used in specification files are similar to the DCL qualifiers used in the command line, as described in Chapter 2. Table 3–1 lists the main keywords and the function of each.

Table 3-1: Specification File Keywords

Keyword	Function
/KEY	Specifies the key(s)
/FIELD	Defines the fields in the input file(s)
/DATA	Specifies the fields in the output file
/STABLE /NOSTABLE	Specifies whether or not you want a stable sort or merge for equal keys
/PROCESS	Defines the processing method for your sorting operation (SORT only)
/COLLATING_SEQUENCE	Specifies one of three predefined collating orders or a user-defined sequence for character key field(s)
/WORK_FILES	Specifies the reassignment of work files
/CHECK_SEQUENCE /NOCHECK_SEQUENCE	Specifies whether or not you want sequence checking (MERGE only)
/CONDITION	Defines conditions for key and data handling and for record selection
/INCLUDE /OMIT	Specify record selection, as well as multiple record formats
/PAD	Specifies a new record-padding character

In most instances, you can specify the keywords in a specification file in any order. The order does become significant, however, when you use the specification file to do any of the following:

- Sort on more than one key field
- Describe the output format
- Define multiple record types
- Use subqualifiers within the /COLLATING_SEQUENCE keyword

To use a specification file at the DCL interface, you use a command line with the following format:

```
$ SORT/SPECIFICATION=specification-file input-file(s) output-file
```

For example, the following DCL command line means that the input files SALES1.LIS and SALES2.LIS are sorted into an output file named SALES.LIS based on the instructions in the specification file named SPEC1.SRT. (The default file type for a specification file is .SRT.)

```
$ SORT/SPECIFICATION=SPEC1 SALES1.LIS,SALES2.LIS SALES.LIS
```

With the MCR/CCL interface, the command line is

```
SRT> output-file=input-file(s),specification-file/SF
```

For example,

```
SRT> SALES.LIS=SALES1.LIS,SALES2.LIS,SPEC1/SF
```

Even though you use a specification file, you can also include in the command line any qualifiers that you might otherwise use in the SORT command line

(without a specification file). Any qualifier that you use in the DCL or MCR/CCL command line overrides a qualifier that is in the specification file. For example, assume you have a specification file named SPEC1.SRT that includes the qualifier /PROCESS = TAG and you use the DCL command line

```
$ SORT/PROCESS=INDEX/SPECIFICATION=SPEC1 SALES1.LIS SALES.LIS
```

Even though the specification file SPEC1.SRT calls for a TAG sort, the file SALES1.LIS is sorted based on the INDEX process, since the command line qualifier (/PROCESS=INDEX) overrides the specification file qualifier (/PROCESS=TAG).

When you call SORT or MERGE from an application program, you can use a specification file either by identifying a specification file in your command line buffer or placing the text of the specification file in a buffer within the program, then passing the buffer to SORT or MERGE. When you place text of a SORT or MERGE specification file within an application program, do *not* use comment characters (!), since they may not be correctly interpreted when running the program.

Chapter 4 of this manual discusses using SORT and MERGE directly from an application program.

3.1.1 Entering Text In A Specification File

When you create a specification file, it is good practice to place each qualifier on a separate line. You can use an exclamation point (!) anywhere on a line in a specification file for internal documentation; all text that follows the exclamation point (on the same line) is considered by SORT to be a comment. For example,

```
      .
      .
      .
/KEY = YEAR          !Primary Key
/KEY = MONTH         !Secondary Key
/KEY = DAY           !Third-level Key
      .
      .
      .
```

However, you should not use any comment characters if your specification file will be used by an application program.

You can begin your specification file text by specifying instructions for the sort process, equal keys, and sequence checking. The default values for keywords in a specification file are the same as the default values for the corresponding qualifiers used in a command line.

3.1.2 Using Qualifiers In The Specification File

Specify record, tag, address, or index sort with the /PROCESS statement.

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

Chapter 2 discusses the four different SORT processes in detail. Record sort is the default. If you intend to reformat the output records, you cannot use address or index sort. Use /PROCESS with SORT only.

Use the /STABLE qualifier to ensure that the order of records with equal keys will be the same in the output file as in the input files. The default is /NOSTABLE.

By default, MERGE checks that your input files are sorted. To override the default, use the /NOCHECK_SEQUENCE qualifier in your specification file.

3.2 Specifying Record Fields

When you use a specification file, you must provide information about each field in the records that you are using whenever you wish to override the default values for fields and keys. 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 using this syntax:

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

The field name must begin with an alphabetic character; it can include only alphabetic characters, numeric characters, or underscores (but no blank spaces); and it can have no more than eight characters.

The position of the field is evaluated in the same way as the POSITION subqualifier 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; you are not required to include the field's data type if it is character. These data types are recognized by PDP-11 SORT/MERGE.

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

DIBOL_ZONED (decimal, trailing overpunched sign)
D_FLOATING
F_FLOATING
PACKED_DECIMAL

For example,in the sample magazine subscription file shown in Chapter 1, the title headings and data were

```
Name              Street              City        State   Exp Date
(1-19)            (20-39)             (40-51)     (52-59) (60-65)

Yellen Mark       90 Lynwood Lane     Westfield   MA      841231
Germont Alfredo   15 Town House Dr    Waltham     MA      850501
Thompson Lynda    395 N Main St       Easton      MA      831130
Fallon Curtis     56 Juniper Lane     Lenox       MA      841101
Tosca Floria      108 Winfield Dr     Salisbury   MA      850630
```

You could use the following lines in the specification file to identify these fields.

```
/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.

PDP-11 SORT/MERGE assumes that the size of all data types is specified in byte lengths when you use the SIZE subqualifier. However, if you want to be compatible with VAX-11 SORT/MERGE (in order to transport the application from PDP-11 SORT/MERGE to VAX-11 SORT/MERGE), you can specify the size of decimal data types in digits by using the DIGITS subqualifier instead of SIZE. (VAX-11 SORT/MERGE requires the DIGITS subqualifier.) 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 key field that contains character data cannot exceed 32,767 characters. 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. Do not specify size for floating-point data, because these data types have implicit sizes.

You can shorten any keyword to its unique abbreviation (for example, /FI instead of /FIELD or POS instead of POSITION). Enclose the set of subqualifiers for each /FIELD clause entry in parentheses; separate the subqualifiers with commas. The default values for sizes and signs are the same as with DCL (see Table 2-1).

3.2.1 Specifying Keys

If you are sorting on the entire record and have character data, there is no need to specify your key field. Otherwise, you must use a /KEY clause for each

of the keys on which you want to sort, in the order of their priority. You can sort on as many as 16 key fields. Use the /KEY clause to provide information about how to sort for a particular field. You identify a field to SORT/MERGE by using the name that you assigned in the /FIELD clause.

You can indicate multiple keys to SORT/MERGE with a series of /KEY clauses. The first key that you list is the primary key, the next key is the secondary key, and so on. For example, assume your specification file includes the following three /KEY clauses:

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

The primary key is the field named CUSTNAME, the secondary key is the field named CITY, and the third-level key is the field named 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 in the /KEY clause, for example:

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

In this example, the name of the field is EXP_DATE and it is sorted in descending order. Separate the subqualifiers in a /KEY clause with commas.

3.2.2 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, assume you indicate the data for your input file as follows:

```
/FIELD = (NAME = CUSTNAME,POS = 1,SIZ = 20)
/FIELD = (NAME = STREET,POS = 21,SIZ = 20)
/FIELD = (NAME = CITY,POS = 41,SIZ = 15)
/FIELD = (NAME = STATE,POS = 56,SIZ = 2)
```

When you sort this file, the data in the output file has the same format. However, you can override this default format with the /DATA statement. For example, assume that you want the sorted (output) data from this example formatted as follows:

```
STATE       CITY        CUSTOMER NAME          STREET
(1-2)       (6-20)      (24-43)                (47-66)
```

You would use /DATA statements in your specification file.

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

The order in which you list the /DATA statements, using the field names defined by previously-issued /FIELD clauses, determines the ordering of fields in your output file. 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. If

you use the /DATA statement to change the formatting of your output file records, you must have a /DATA statement for each field that you direct to your output file.

3.2.3 Defining and Using Conditions

When you use a specification file, you can have your records sorted based upon certain conditions that you specify using the /CONDITION clause. The /CONDITION clause is used afer a /FIELD clause; it can establish a means of sorting or reordering a field based on data that does not exist (by itself) in any specific field.

For example, assume that you have a series of customer records in the format shown in the previous section, and that your sales area is divided into three regions, depending on the customer's home state. You can use the /CONDITION clause, followed by a /KEY clause and (optionally) a /DATA statement, to sort your records by sales region even though you do not have a field devoted specifically to sales region. Use the /CONDITION clause as follows:

```
/FIELD = (NAME = STATE,POS = 56,SIZ = 2)

/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 in the /FIELD clause identifies the field on which the conditional testing is to be done. The /CONDITION clause tests for matches between record data and the values that you specify; when data in a record field matches a value in a /CONDITION clause, the associated sorting tag is given to the record field. In the example shown, 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 clause then assigns a sorting value 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 on this key, then the values that you assign in the /KEY clause 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 clause attaches a text string to the output file. In this example, the records with the REGION1 sort tag 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) is output with the message "ERROR".

You use the TEST= clause with the following two-letter operators to define conditionals:

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

Use the keywords AND and OR to include more than one conditional test in a TEST= clause; enclose the TEST= clause in parentheses. If the data in the field is alphabetic and you use the greater than or less than operators (GT, GE, LT, or LE), then the ASCII value of the alphabetic data is compared to the ASCII value of the text in the TEST= statement.

You can use the /CONDITION clause to change the contents of a field. For example, suppose that some of the data in your records erroneously spelled the city "CINCINNATI" as "CINNCINATTI". You could use the /CONDITION clause to correct this 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 clause:

```
/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 there was no /KEY clause used with the /CONDITION clause, there is no effect on the order of the sorted output.

3.2.4 Specifying Record Selection

You can also use the /CONDITION clause to select those records to include or exclude from the sort or merge process by using the condition clause as a subqualifier to an /INCLUDE or /OMIT statement. For example, if you

wanted to sort only the records of customers in California, you could use the following:

```
/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 clause with an /OMIT statement, 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 statement and subsequently include the record with an /INCLUDE statement, the record will be included in the output file. However, only the keys that were indicated prior to the /OMIT statement and after the /INCLUDE statement will be 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 is /OMIT that specifies criteria, then everything not specifically omitted is included; if the last instruction is /INCLUDE that specifies criteria, then everything not specifically included is omitted. If the last instruction is /INCLUDE or /OMIT that does not specify criteria, then everything not specifically omitted is included (with /INCLUDE), or everything not specifically included is omitted (with /OMIT).

3.3 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 an application program that calls SORT or MERGE.)

```
/FIELD = (NAME=REC_TYPE, PO=1, SIZ=1)       ! Record's type, one-byte field
/FIELD = (NAME=PRICE, PO=2, SIZ=8)          ! Price field, both files
/FIELD = (NAME=TAXES, PO=10, SIZ=5)         ! Taxes field, both files
/FIELD = (NAME=STYLE_A, PO=14, SIZ=10)      ! Style field, format A file
/FIELD = (NAME=STYLE_B, PO=19, SIZ=10)      ! Style field, format B file
/FIELD = (NAME=ZIP_A, PO=24, SIZ=5)         ! Zip code field, format A file
/FIELD = (NAME=ZIP_B, PO=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 or data fields in an /INCLUDE clause, you must explicitly specify *all* the key and data fields for your sort operation in the /INCLUDE clause(s).

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.

At the end of this chapter is a sample specification file that shows the use of the /CONDITION statement.

3.4 Special Considerations

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. For example, if you specify any /KEY qualifier in the DCL or MCR/CCL command line, SORT/MERGE will ignore all /KEY, /DATA, /INCLUDE, and /OMIT clauses 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, you can specify /NOCHECK_SEQUENCE, so that MERGE will not check the order of the input records.

3.5 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:

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

3.5.1 Defining Your Own Collating Sequence

You can modify the ASCII, EBCDIC, and MULTINATIONAL collating sequences, as described in the next section, but if none of these collating sequences is suited to your particular application, you can define your own. (You can modify your own collating sequence also. See the next section.)

You define your own collating sequence by specifying 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"-"R","RR","S"-"Z"))
```

This sequence signifies that the double character LL collates as a single character between L and M, and that RR collates as a single character between R and S. These double characters would otherwise appear in their normal alphabetical order as two characters in alphabetical sequence (that is, "L" followed by "L" and "R" followed by "R").

When you specify a collating sequence, upper case characters and lower case characters are treated separately. For example, assume that you have the following clause in your specification file:

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

The records are collated first using upper case A-Z and then lower case a-z, as shown in the following example:

```
APPLES
MUSHROOMS
ZUCCHINI
```

```
cucumbers
daffodils
sorrel
```

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 will represent a complete specification of all the characters appearing in the character keys in your sort or merge operation. SORT/MERGE ignores any character to which you have not given a collating value.

You must observe the following rules when defining your collating sequence:

- Do not define a character more than once, by whatever representation, including octal, decimal, and so on.
- Do not specify the null character with the "" construct. Instead, use a radix operator, such as %X0.
- Specify the quotation mark character by doubling its occurrence within quotes: """", or by using a radix operator.

3.5.2 FOLD and TIE_BREAK

You can cause upper case and lower case characters to be collated together by using the keyword FOLD in the /COLLATING_SEQUENCE clause. That is, when you use FOLD, then SORT/MERGE does not discriminate between upper and lower case characters when collating the records. For example, when you sort the previous list using the clause

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

the output is as follows:

```
APPLES
cucumbers
daffodils
MUSHROOMS
sorrel
ZUCCHINI
```

You can also represent characters by their corresponding octal, decimal, or hexadecimal values, by using the radix operators %O, %D, or %X.

FOLD causes all lowercase characters to be given the collating value of their uppercase equivalents. In effect, FOLD is the same as using the 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 original 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 done after an initial comparison of collating values results in equal values. This tie-breaking process basically arranges the characters according to a predefined order, as shown in the multinational character set collating sequence table. 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.

For the multinational collating sequence, tie-breaking is automatically done unless you explicitly specify NOTIE_BREAK. If you use NOTIE_BREAK in a multinational collating sequence, only an initial comparison of the collating values is made, and some unexpected ordering may result.

3.5.3 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. You use the keyword MODIFICATION within the /COLLATING_SEQUENCE clause to indicate your instructions for any modifications, along with your request for the basic collating sequence, as follows:

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

Remember the following rules when you use the /COLLATING_SEQUENCE clause:

- You must use a comma between keyword clauses
- You must enclose the characters to be modified in quotation marks or use a radix
- You can make more than one modification to your collating sequence, separating them with commas

If you want to modify any of the predefined collating sequences (ASCII, EBCDIC, or MULTINATIONAL) in any way, or if you want to use FOLD, IGNORE, or TIE_BREAK, you are required to specify this sequence in the /COLLATING_SEQUENCE clause (for example, /COLLATING_SEQUENCE = (SEQUENCE = EBCDIC,MODIFICATION=...)).

The following information describes the kinds of changes permitted in the MODIFICATION clause.

- You can equate 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")
  ```

- You can have a single or double character 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")
  ```

- You can have a single or double character 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 two following ways:

  ```
  ,MODIFICATION("CH"<"D")

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

- You can equate 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")
  ```

- You can equate a single character to a two-character sequence.

  Thus, if you want ligature Æ to collate the same as the two-character sequence of AE, you specify ,MODIFICATION=("Æ"="AE").

  You can also request that SORT/MERGE ignore a character or character range within the given collating sequence, as follows:

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

For example, the clause

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

would cause the following fields to be compared as equal:

```
252-3412
252 3412
2523412
```

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

If you do not want the default tie-breaking algorithm when using the MULTINATIONAL collating sequence, specify the keyword NOTIE_BREAK. For tie breaking in the other collating sequences, specify the keyword TIE_BREAK.

3.5.4 Example of a User-Defined Collating Sequence

Suppose you want to reorder a file named SEMNAR.DAT that contains a schedule of seminars. The file was previously sorted by seminar title. Now you want to order it by seminar date. The file SEMNAR.DAT is set up as follows:

```
16 NOV 1984   Communication Skills
05 APR 1983   Coping with Alcoholism
```

```
11 Jan '83     How to Be Assertive
12 OCT 1984    Improving Productivity
15 MAR 1983    Living with Your Teenager
08 FEB 1983    Single Parenting
07 Dec '84     Stress -- Causes and Cures
14 SEP 1984    Time Management
```

The primary key is the year field; the secondary key is the month field. 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 -- giving each month a unique key value.

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
/KEY=MONTH                                         ! Secondary key
/KEY=DAY                                           ! Third-level key

/COLLATING_SEQUENCE=(SEQUENCE=                     ! User-defined sequence
  ("AN","EB","AR","PR","AY","UN","UL",             ! that gives each month
   "UG","EP","CT","OV","EC","0-9"),                ! a unique value in its
                                                   ! chronological order
   MODIFICATION=(""'="19"),

   FOLD)
```

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

- DCL -

```
$ SORT/SPECIFICATION=SPEC.SRT SEMNAR.DAT SCHED.DAT
```

- MCR/CCL -

```
SRT> SCHED.DAT=SEMNAR.DAT,SPEC.SRT/SF
```

The output from this sort operation appears as follows:

```
14 SEP 1984  Time Management
12 OCT 1984  Improving Productivity
16 NOV 1984  Communication Skills
07 Dec  '84  Stress - Causes and Cures
11 Jan  '83  How to Be Assertive
08 FEB 1983  Single Parenting
15 MAR 1983  Living with Your Teenager
05 APR 1983  Coping with Alcoholism
```

3.6 Reassigning Work Files

You can improve the performance of SORT by placing work files on different disk-structured devices. You must use a specification file if you want to reassign work files. Specify this reassignment as follows:

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

where **workfile** is in the format "ddnn:[uic,ppn]" (for example, /WORK_FILES = ("db0","dm0:[250,3]"). The first work file is placed on the first device listed, the second work file on the second device listed, and so on.

3.7 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. You cannot 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>"

3.8 Specification File Summary

Table 3–2 lists the statements and qualifiers that you can use in a specification file.

Table 3–2: Specification File Summary

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

/WORK_FILES=(workfile,workfile,...)
                    !where workfile is in the format "ddnn:[uic,ppn]"

/[NO]CHECK_SEQUENCE
```

Table 3-2 (Cont.): Specification File Summary

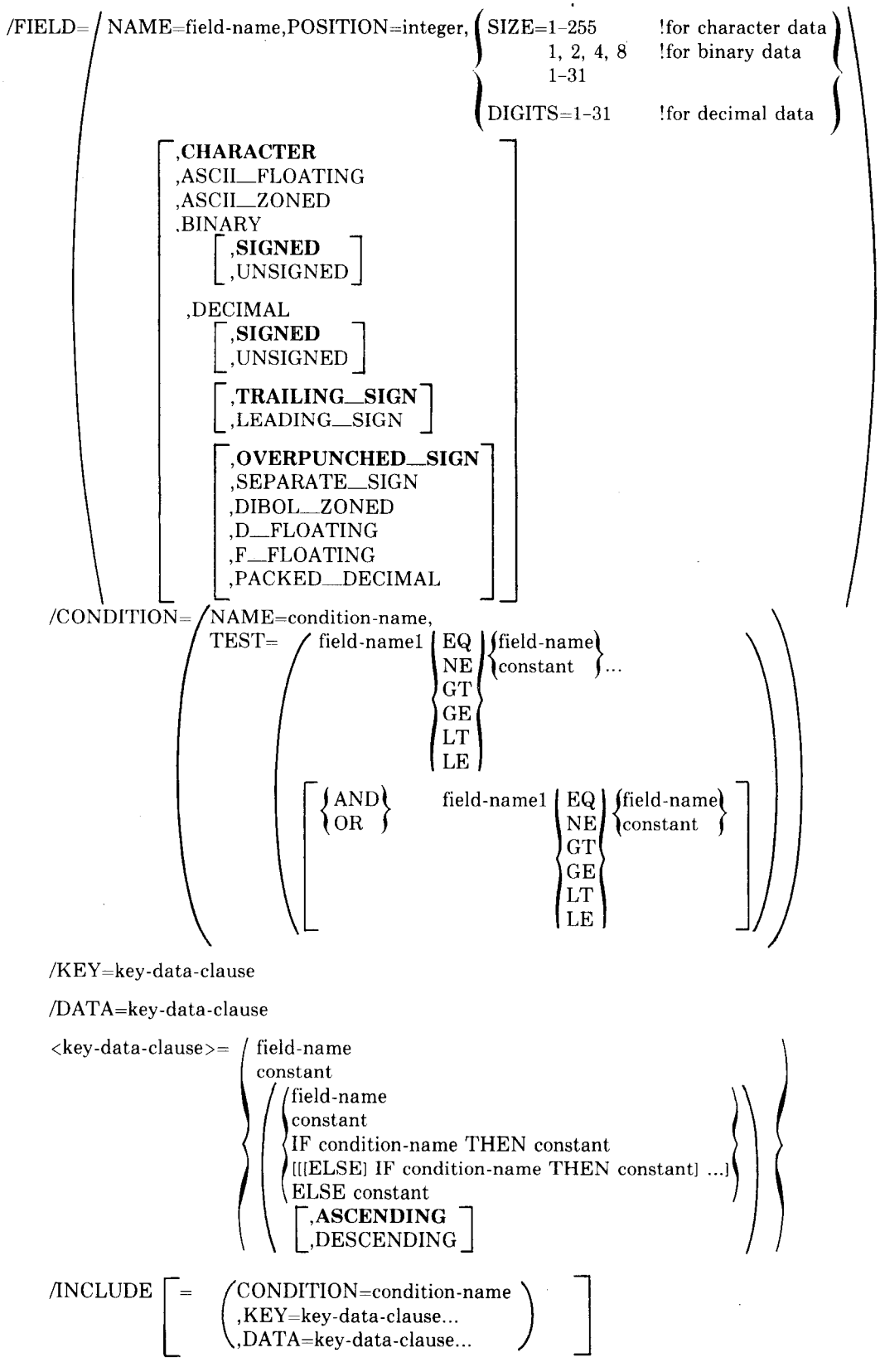

Table 3–2 (Cont.): Specification File Summary

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

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

/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>"     }
```

3.9 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

Comment characters (!) are included in this sample specification file. If you plan to use the specification file text in the buffer of an application program, do not use any comment characters.

Chapter 4
Using SORT and MERGE in Application Programs

You can invoke SORT and MERGE from an application program by using a sequence of callable subroutines that SORT/MERGE provides. This chapter describes how to use callable subroutines to access SORT/MERGE from a program at run time. This feature is especially useful when you want to process and then sort data from a program; you process the data and write data files using standard program logic, using SORT or MERGE to arrange the data before and/or after the data is processed by the program. This chapter

- Discusses the languages that support PDP-11 SORT/MERGE
- Describes the two callable interfaces for SORT and MERGE
- Discusses the SORT/MERGE subroutines and their parameters (mandatory and optional)

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

4.1 Language Support

You can call the SORT/MERGE subroutines from the following PDP-11 native-mode languages:

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

Note that COBOL-81 includes the COBOL syntax for many SORT and MERGE functions. For those functions that are not provided by the ANSI COBOL standard syntax, direct subroutine calls to PDP-11 SORT/MERGE can be made.

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

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 two words, containing the address and the length. The BASIC array descriptor is also two words, but it lists the length and then the address. The COBOL descriptors used for SORT/MERGE are all two-word descriptors, containing a length 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 help, 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 The Two Callable SORT and MERGE Interfaces

You can access the SORT or MERGE subroutines through one of two interfaces, the *file interface* or the *record interface*. These alternative interfaces allow you to submit your records for sorting or merging either as complete files (file interface) or as individual records (record interface). 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 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 in either of two ways:

- Have your program submit file(s) as input and receive the ordered records as output
- Have your program release 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 file(s) 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, then 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 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 using BASIC-PLUS-2.

Depending on what your programming language allows, you specify the use of your own routine(s) in one of two ways:

1. If permitted by the language you are using, you specify the address(es) of your routine(s) 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, then you 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 ODL file, 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. Register 5 (R5) must be used as the linkage register.

4.4 Calling the SORT Subroutines

You provide the same types of information when you use SORT at the program level as when you use SORT at the command level or from a specification file. Specifically, you provide SORT with:

- File specifications (when you use the file or mixed-mode interface)
- Information about keys (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 your program. One of the parameters to SORT and MERGE subroutine calls is a four-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 STS/STV values. You can have your program test the values of words 1–4 to determine success or failure conditions.

When you use a SORT subroutine, there are 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 between parameters. 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 any time 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(s), sorts the records, and writes the records to the output file
SRTEND	Does 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

File Interface

For a sort task using the file interface, first call the initialization subroutine SRTINI. Remember 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 keys and sort options. You also indicate whether you want key comparisons to be done by SORT 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 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.

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

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 keys 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.

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 and SRTRLS once for each record, 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.

4.4.1 Passing File Names and Initializing Your Sort

As described in the previous section, 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 information and key options.

When you call the SRTINI initializing subroutine, there are six required parameters that you must include and eight optional parameters that you may include, depending on your requirements. The SRTINB (for BASIC) and SRTINC (for COBOL) subroutines each have four required parameters and six 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. (You pass all parameters by reference unless otherwise noted.)

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]	
7. Specification file buffer [2]	Pass by descriptor
8. Spec. file buffer length [2]	Omit
9. 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]	

1. Required when LRL is unavailable
2. Optional

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 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 multi-blocking the I/O reads and writes. The following formulas can be used for rough calculations of the minimum memory needed for a particular sort. 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:

```
(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. In situations 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/CCL command line for your sort operation. Chapter 2 describes the use of the MCR/CCL switches.

   The command line in this buffer differs from the MCR/CCL 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/CCL 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 specification(s) and any switches that describe the input file(s). 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 buffer 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 clustersize (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. Also, remember that you should not 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/CCL 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) 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:

   LUN 2 - for specification file

   LUN 3 - for output file

   LUN 4 - for 1st input file

   After LUN 4, there is an additional LUN 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 by using:

    - 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 two parameters passed by reference, the error buffer address and a return status code address. The error buffer is the four-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
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 4.6.2 for information about the use of ODLs.

13. **Comparison routine address**

You can use this optional parameter to use your own comparison routine rather than the key comparisons that SORT provides. SORT calls this routine with five 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 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 4.6.2 for information about the use of ODLs.

14. **Equal-key routine address**

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

SORT calls the equal-key routine with five 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 4.6.2 for information about the use of ODLs.

4.4.2 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.

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.

4.4.3 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.

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

Parameters	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	

Note 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 record buffer length. 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.

4.4.4 Sorting the 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(s) 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.

4.4.5 Ending Your Sort

Call the SRTEND subroutine to end a sort process; 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

You use MERGE from an application program in the same way as you use 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 keys, 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 four-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.

When you use a MERGE subroutine, there are 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. The sections that follow describe each of these subroutine calls.

Table 4–5: MERGE Subroutines

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

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 and to direct the sorted records to the output file. Finally, call MRGEND (MRGENB for BASIC, MRGENC for COBOL) to end the sort and to release resources.

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

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 keys and merge options. When you use the record interface to MERGE, you must also provide the address of a user-defined input routine when you call MRGINI. This is explained later in this chapter 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. MERGE, unlike SORT, 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.

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.

4.5.1 Initializing Your Merge

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 which 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. Spec. 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]	

1. Required when LRL is unavailable
2. Required for record interface, or in mixed-mode interface with record interface on input
3. Optional

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:

((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 given to increase 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. 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 keys 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 4.6.2 for information about the use of ODLs.

14. **Warning routine address**

Call this routine with the global symbol MRGWRN. 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 SORT equal-key routine address parameter.

4.5.2 Summary of SORT Subroutine Calls

Table 4–7: Summary of SORT Subroutine Calls for 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]	
	Spec. file buffer [2]	Pass by descriptor
	Spec. 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		

1. Required if unavailable from record
2. Optional

Table 4–8: Summary of SORT Subroutine Calls for 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]	
	Spec. file buffer [2]	Pass by descriptor
	Spec. 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]	

1. Required if unavailable from record
2. Optional

Table 4-8 (Cont.): Summary of SORT Subroutine Calls for Record Interface

Subroutine	Parameters	BASIC/COBOL Usage Information
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		

3. Use either record buffer and record buffer length OR record location

Table 4-9: Summary of SORT Subroutine Calls for File Interface Input, 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]	
	Spec. file buffer [2]	Pass by descriptor
	Spec. 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		

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

Table 4–10: Summary of SORT Subroutine Calls for Record Interface Input, 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]	
	Spec. file buffer [2]	Pass by descriptor
	Spec. 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		

1. Required if unavailable from record
2. Optional

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, refer to the appropriate manual in your operating system documentation set.

4.6.1 Overlay Descriptor Language (ODL) Files

SORT/MERGE provides seven ODL files. When you use SORT/MERGE in an application program, you must create your own ODL file that references one of the SORT/MERGE ODL files. You 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. SORT/MERGE provides seven ODL files; the one that you reference depends on

- The operation that you are doing — 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 the SORT/MERGE resident 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	File or mixed-mode	STMGFL.ODL	10,250
Combined SORT/MERGE	Record	STMGRC.ODL	9100
Combined SORT/MERGE Resident Library	File, record, or mixed-mode	SMSHR.ODL	8350

Note that the file/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 non-overlaid 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 be sure 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, you 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 you alter the SORT/MERGE ODL file, make sure that you are changing 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 line

```
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 line

```
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 line

```
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 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

```
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 callback routine. The name of the object file for your equal key callback routine is EQUAL.OBJ.

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

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

to read

```
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.

```
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 LUN Usage

SORT requires the following number of LUNs:

1 LUN for a specification file (if used)

1 LUN for the output file

1 LUN for each input file

1 LUN for each work file

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

Chapter 5
Improving SORT Efficiency

PDP-11 SORT/MERGE is designed for an environment of random access disk devices, fairly large files, and medium size records. PDP-11 SORT/MERGE automatically provides an efficient sort or merge for the data type of your key(s).

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 things as types of I/O devices, key data types, file sizes, and key and record sizes.

So that you can better understand the factors to consider when fine-tuning your ordering operations, this chapter

- Provides a brief description of how SORT operates internally
- Explains the meaning and use of SORT/MERGE statistics
- Suggests procedures for tailoring ordering operations to your environment

5.1 How SORT Operates

The following description of SORT's internal operation does not try to give all the details of the process. Instead, it gives a summary of what happens during a sorting operation, paying special attention to the internal sorting data structure and work files, because these can be modified by the user for greater SORT efficiency.

You can use SORT in either of two ways:

- As a utility program that you invoke with a DCL or MCR/CCL command line
- As a package of subroutines that you call from an application program

The utility uses the subroutines to perform a sort operation. There is almost no difference 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 does the following:

1. Interprets the command line
2. Interprets the specification file (if any)
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 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 made up of nodes of information about the records being sorted. The internal node size varies depending upon the type of sort being performed. Node size is discussed further in Section 5.2.

First, certain data structures containing information about keys, 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 that there are in the tree, the faster is the sort.

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 multi-block reads and writes.

Before the polyphase merge 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 multi-block 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 together. This requires n nodes plus one or two extra nodes for certain kinds of processing during the merge. Since there usually are fewer merge nodes 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 multi-blocking occurs for I/O reads and writes.

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-comparison and equal-key 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 from the sort data structure (or the largest value if descending sort order is specified), 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 (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 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 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. 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* 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 longer the polyphase merge will take. 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. There is a tradeoff, however, since the more work files there are, the less I/O multi-blocking space that is available for each work file, thus increasing the time required for work file reads and writes.

The default number of work files that SORT/MERGE provides has been shown to demonstrate 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 particular SORT or MERGE operation, you can change the number of work files using the /WORK_FILES qualifier (with DCL) or the /FI:n switch (with MCR/CCL).

5.1.3 Merge Phase of the SORT

If the Fibonacci level reached is greater than Level One (that is, at least one work file contains more than one run), then SORT merges runs from (n–1) work files to the 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 One 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 (for utility sort and file interface), and then closes and deletes the work files. When requested by the user, 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 (with DCL) or the /SS switch (with MCR/CCL) causes SORT/MERGE to display statistics on your output device. You can also have statistics returned from a callable sort or merge.

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 to help you determine if the processing went as you had intended. The statistics include the following information:

- Identification — The version and maintenance release numbers. For example, V3.1 is version 3, maintenance release number 1.
- Elapsed time — The clock time from the beginning of the initialization phase to just before the statistics are output. Do not confuse this with CPU time. Clock time may vary considerably for different instances of the same sort or merge, depending upon other activity on the system.
- Process — Shows 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 — Shows whether the collating sequence used was 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) — Shows 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 — Shows exactly how many records were written to the output file or returned to the calling program.
- Number of records input — Shows how many records were read from the input files or passed to the callable subroutines.
- Number of records omitted — Shows how many records were omitted from the sort or merge. Records are omitted because of an /OMIT clause in a specification file, a /NODUPLICATES qualifier (or /ND switch), or because an equal key callback routine returned a request that deleted a record.
- Number of keys — Shows the number of key fields used in the sort or merge.
- Total key size — Shows the maximum length of the key fields of the record formats in the sort or merge.
- Total input allocation — Shows the total allocation (in blocks) for all input files.
- Total work allocation (SORT only) — Shows the total allocation (in blocks) for all work files in a sort. This is the final allocation after all necessary extensions have been made during the sort.
- Total output allocation — Shows the total allocation (in blocks) for the output file. This is the final allocation after all necessary extensions have been made during the sort.
- Longest input record found — Shows the length of the longest input record found 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 — The internal node size varies depending upon the type of sort being performed. For a record sort or for a merge, the node size is approximately the sum of the following:

  a. The maximum longest record length for the input files
  b. About 6 to 10 bytes of information about the record
  c. About 8 bytes of pointers for the replacement selection, polyphase merge, or merge algorithm

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

  a. The total key size for the sort
  b. About 12 to 16 bytes of information about the record
  c. About 8 bytes of pointers for the replacement selection or polyphase merge algorithm

  For large record size and small key 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. It 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 — Shows the number of nodes in the replacement selection tree for a sort, or the number of nodes in the merge list for a merge. In a sort, 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 — Shows the size (in bytes) of the area provided for I/O data structures and buffers. For a sort, 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) — Shows the number of ordered strings (runs) written to work files during the sort distribution phase.
- Fibonacci level (SORT only) — Shows the Fibonacci level, as discussed in the previous section. If the sort was internal (that is, it required no work files), the Fibonacci level is zero.
- Merge order (MERGE only) — Shows 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 3.0, for example, the two words would contain 000003 and 000000)

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 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 keys (1 word)
20. Total composite key 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 What The User Can Do

After evaluating the variables in the environment for your sort or merge operation, you can consider a number of possibilities for improving SORT efficiency:

- Sorting fewer records — You can use the specification file to include only those records that are needed in the output.
- Sorting shorter records — You can also use the specification file to reformat records to eliminate fields that are not needed in the output.

You should also consider these additional options for improving efficiency, described in detail in the following sections.

- 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

5.3.1 Work Files

Unless you specifically request that no work files be created, (/WORK_FILES=NUMBER:0 or /FI:0), SORT creates the 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. Sometimes, however, it is necessary or useful to increase the number of work files. For example, if the 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; you can also assign the work files to different devices. Remember that the more work files there are, the less I/O area each work file receives for multi-blocking, thus tending to slow down the sort.

There are also conditions where 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. Chapter 2 shows you how to specify the number of work files.

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 described 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. The initial block allocation for each work file is derived as follows: First, multiply the estimated number of input records by the node size; next, divide the product by one less than the number of work files. Divide this figure by 512 and round the result up to the next integer. This is the initial allocation, in blocks.

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

Chapter 2 shows the syntax for using the ALLOCATION:n and CONTIGUOUS subqualifiers with the /WORK_FILES qualifier (for DCL) and the /AL:n/CO input file switch (for MCR/CCL).

You can also specify the RSTS/E file clustersize or the RSX-11M/M-PLUS retrieval window size for your work files using the SIZE:n subqualifier or the /SI:n input file switch. See your operating system documentation for more information.

DCL users should remember to enclose any group of subqualifiers in parentheses.

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 assumes a file size of 1000 blocks. The default for the record interface on input is also 1000 blocks.

If this allocation is too large, SORT will overestimate its memory and work file requirements. In this instance, therefore, your sort operation is more efficient if you specify a smaller input file size. If the default of 1000 blocks is too small, SORT will underestimate its memory requirements, and you will need to specify a larger input file size.

Chapter 2 describes the use of the FILE_SIZE subqualifier and /BK:n switch to specify file size.

5.3.3 Output File Preallocation

SORT/MERGE preallocates space for your output file based on total input file allocation. This avoids the overhead of extending the file every time another few 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 qualifier or the /AL:n output file switch. See Chapter 2 for information about using these.

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 file(s), 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/M-PLUS and 15 for RSTS/E.

Syntax information for the fill factor and bucket size options is provided in Chapter 2.

The /TREE_SPACE qualifier (in DCL) or /PT switch (in MCR/CCL) lets you override the default division of work area space and choose the distribution of available work area between SORT/MERGE data structures and I/O data structures. By default, 55% of the work area is used for data structures for a SORT operation, and 30% of the work area is used for data structures for a MERGE operation. Chapter 2 shows the syntax for using 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 will be retrieved in order, record sort is usually the fastest sort process. If limited work space is a problem, or your records are very large relative to the total key size, consider using tag sort, which requires less space than record sort.

5.4 What the System Manager Can Do

The system manager can designate one batch queue for sorting jobs and provide this queue with characteristics that improve system and/or sort performance. In addition, job process parameters can be adjusted for greatest SORT efficiency.

Although the default SORT/MERGE installation yields optimum sort and merge performance for most applications, your special needs may require that different defaults be installed. See the *Installation Guide* for your operating system to learn how to install SORT/MERGE with different default parameters.

Appendix A
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/CCL 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.

The table that begins on page A–2 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.

Number Code	Message Code	Number Code	Message Code	Number 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
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_SPCSIS	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_CNOF00	83	%SORT_F_EMTTRP
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_CNSP00	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

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

Callable returned error buffer: First word = -62

%SORT_F_BADCMP: Bad return from comparison routine

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

Callable returned error buffer: First word = -61

%SORT_F_BADFLD: Bad field in record number:

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

Callable returned error buffer: First word = -20
Second word = low word of record number
Third word = high word of record number

%SORT_F_BADINP: Bad return from merge input routine

Explanation: User-supplied merge input routine has returned illegal status.

Callable returned error buffer: First word = -63

%SORTFBADORG: Organization of existing file incorrectly specified

Explanation: The user 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 the user is attempting to overlay an existing output file. For input files, this can occur if the user specifies as indexed-sequential a file that was not index-sequential, or if the user fails to specify an index-sequential file as being such.

Callable returned error buffer: First word = 133

%SORT_F_BADSEQ: Bad sequence in input file

Explanation: Given input file is out of sequence for merge

Callable returned error buffer: First word = -73
Second word = input file number

%SORT_F_BADVAL: Bad switch value

Explanation: Command line has bad switch value.

Callable returned error buffer: First word = -66
Second word = offset into command line of bad switch

%SORT_F_BPTBIT: Breakpoint or T-bit exception

Explanation: Register dump will follow. Please submit with SPR.

Callable returned error buffer: No return to callable.

%SORT_W_BUFOVR:

Explanation: User-supplied buffer was not big enough to hold returned record. Normally just a warning. Record is truncated. If this message occurs in utility, please submit SPR.

Callable returned error buffer: First word = 2
Second word = length of output record
Third word = buffer size

%SORT_F_CHNERR: Failed to chain: .RUN return status =

(RSTS/E only)

Explanation: SORT or MERGE was unable to chain to requested task.

Callable returned error buffer: First word = -76
Second word = .RUN directive return status

%SORT_F_CHNFIL: Bad chain file specification: XRB flag2/flag1:

(RSTS/E only)

Explanation: Chain file as specified by user was invalid.

Callable returned error buffer: First word = -77
Second word = XRB flag 2
Third word = XRB flag 1

%SORT_F_CHNPRS: Error parsing chain file: .FSS return status =

(RSTS/E only)

Explanation: Unable to parse chain file specification.

Callable returned error buffer: First word = -75
Second word = .FSS directive return status

%SORT_F_CLIF00: Error closing input file: RMS codes

Explanation: RMS could not close the given input file.

Callable returned error buffer: First word = -47
Second word = RMS STS code
Third word = RMS STV code
Fourth word = Input file number

%SORT_F_CLOF00: Error closing output file: RMS codes

Explanation: RMS could not close the output file.

Callable returned error buffer: First word = -53
Second word = RMS STS code
Third word = RMS STV code

%SORT_F_CLSF00: Error closing work file: RMS codes

Explanation: RMS could not close the given work file.

Callable returned error buffer: First word = -42
Second word = RMS STS code
Third word = RMS STV code
Fourth word = Work file number

%SORT_F_CLSP00: Error closing specification file: RMS codes

Explanation: RMS could not close the specification file.

Callable returned error buffer: First word = -58
Second word = RMS STS code
Third word = RMS STV code

%SORT_F_CNIF00: Error connecting to input file: RMS codes

Explanation: RMS could not connect to the given input file.

Callable returned error buffer: First word = -44
Second word = RMS STS code
Third word = RMS STV code
Fourth word = Input file number

%SORT_F_CNOF00: Error connecting to output file: RMS codes

Explanation: RMS could not connect to the output file.

Callable returned error buffer: First word = -50
Second word = RMS STS code
Third word = RMS STV code

%SORT_F_CNSF00: Error connecting to work file: RMS codes

Explanation: RMS could not connect to the given work file.

Callable returned error buffer: First word = -37
Second word = RMS STS code
Third word = RMS STV code
Fourth word = Work file number

%SORT_F_CNSP00: Error connecting to specification file: RMS codes

Explanation: RMS could not connect to the given specification file.

Callable returned error buffer: First word = -55
Second word = RMS STS code
Third word = RMS STV code

%SORT_F_CROF00: Error creating output file: RMS codes

Explanation: RMS could not create the output file.

Callable returned error buffer: First word = -48
Second word = RMS STS code
Third word = RMS STV code

%SORT_F_CRSF00: Error creating work file: RMS codes

Explanation: RMS could not create the given work file.

Callable returned error buffer: First word = -36
Second word = RMS STS code
Third word = RMS STV code
Fourth word = Work file number

%SORT_F_DCIF00: Error disconnecting from input file: RMS codes

Explanation: RMS could not disconnect from the given input file.

Callable returned error buffer: First word = -46
Second word = RMS STS code
Third word = RMS STV code
Fourth word = Input file number

%SORT_F_DCOF00: Error disconnecting from output file: RMS codes

Explanation: RMS could not disconnect from the given output file.

Callable returned error buffer: First word = -52
Second word = RMS STS code
Third word = RMS STV code

%SORT_F_DCSF00: Error disconnecting from work file: RMS codes

Explanation: RMS could not disconnect from the given work file.

Callable returned error buffer: First word = -40
Second word = RMS STS code
Third word = RMS STV code
Fourth word = Work file number

%SORT_F_DCSP00: Error disconnecting from specification file: RMS codes

Explanation: RMS could not disconnect from the given specification file.

Callable returned error buffer: First word = -57
Second word = RMS STS code
Third word = RMS STV code

%SORT_F_EMTTRP: Non-RSX EMT trap

Explanation: Register dump will follow. Please submit with SPR.

Callable returned error buffer: No return to callable.

%SORT_W_EOFEXC:

Explanation: End of file returned from callable interface. If this message occurs in utility, please submit SPR.

Callable returned error buffer: First word = 1

%SORT_F_EXTSRT: SORT requires work files: RMS codes

Explanation: User indicated no work files, but sort could not be done internally.

Callable returned error buffer: First word = -35
Second word = RMS STS code
Third word = RMS STV code
Fourth word = Input file number

%SORT_F_FPTRAP: Floating-point exception

Explanation: Register dump will follow. Please submit with SPR.

Callable returned error buffer: No return to callable.

%SORT_F_GCMBAD: Cannot get command line: GCML error

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 continuation character (-) to divide line into smaller units)
-40 Command line input buffer too small for total command line. (Shorten your command line)

Callable returned error buffer: First word = -22
Second word = GCML error code

%SORT_F_ILCALL: Illegal calling sequence: state

Explanation: User called sort or merge subroutines in incorrect order. If this message occurs in utility SORT or MERGE, please submit an SPR.

Callable returned error buffer: First word = -25
Second word = SORT or MERGE internal state code
If necessary submit with SPR

%SORT_F_ILOPTR: Illegal instruction trap

Explanation: Register dump will follow. Please submit with SPR.

Callable returned error buffer: No return to callable.

%SORT_F_INCNODATA: INCLUDE specification references no data

Explanation: In specification file, INCLUDE specification has to contain a DATA clause.

Callable returned error buffer: First word = -96
Second word = specification file line number

%SORT_F_INCNOKEY: INCLUDE specification references no keys

Explanation: In specification file, INCLUDE specification has to contain a KEY clause.

Callable returned error buffer: First word = -95
Second word = specification line number

%SORT_F_INTERR: Internal SORT/MERGE error

Explanation: Please submit an SPR.

Callable returned error buffer: First word = -60

%SORT_F_INVSWH: Invalid or redundant switch

Explanation: Command line has invalid switch or two switches which cannot occur together.

Callable returned error buffer: First word = -67
Second word = location (offset into command line) of bad switch

%SORT_F_IOTTRP: IOT trap

Explanation: Register dump will follow. Please submit with SPR.

Callable returned error buffer: No return to callable.

%SORT_W_LCKBKT: Locked bucket in input file

Explanation: RMS attempted to read bucket which was locked in input file. SORT/MERGE will retry reading bucket the number of times specified at installation. If retry fails, a read error will be issued.

Callable returned error buffer: First word = 4
Second word = Input file number

%SORT_F_LSTMSG:

Explanation: Too large message code detected. Probably an internal error. Please submit an SPR.

Callable returned error buffer: First word = -139
Second word = message number

%SORT_W_LSTWRN:

Explanation: Too large warning message code detected. Probably an internal error. Please submit an SPR.

Callable returned error buffer: First word = 19
Second word = message number

%SORT_F_MAXINP: Too many input files

Explanation: Too many input files specified in command line.

Callable returned error buffer: First word = -68

%SORT_F_MAXOUT: Too many output files

Explanation: Too many output files specified in command line.

Callable returned error buffer: First word = -69

%SORT_F_MEMPRO: Memory protect error

Explanation: Register dump will follow. Please submit with SPR.

Callable returned error buffer: No return to callable.

%SORT_F_MISLRL: No LRL found for file

Explanation: The longest record length for each input file must be made known to SORT or MERGE either through RMS or by format switch.

Callable returned error buffer: First word = -70
Second work = input file number

%SORT_F_MISPRM: Missing required parameter

Explanation: User did not pass a required parameter to callable subroutine.

Callable returned error buffer: First word = -65

%SORT_F_MIXKEY: Incompatible key comparison

Explanation: SORT or MERGE attempted to compare two keys which were not compatible. This should only occur when using a specification file to specify multiple record formats.

Callable returned error buffer: First word = -21

%SORT_W_MRGORD:

Explanation: User passed merge order to callable file interface which 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.

Callable returned error buffer: First word = 3

%SORT_F_MULSPC: Multiply defined specification file

Explanation: The user 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.

Callable returned error buffer: First word = -23

%SORT_F_NOIORM: No room for I/O pool space

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.

Callable returned error buffer: First word = -24

%SORT_F_NOMSG: Message number

Explanation: This is an internal error; please submit an SPR.

Callable returned error buffer: First word = -64
Second word = bad message number

%SORT_F_NOSCBF: Out of work file I/O buffer space

Explanation: This is probably an internal error; please submit an SPR.

Callable returned error buffer: First word = -31

%SORT_F_NOTMRG: Non-MERGE switch

Explanation: User specified a MERGE command line switch which is only valid for SORT.

Callable returned error buffer: First word = -71
Second word = location of switch (offset into MCR command line)

%SORT_F_NOTSRT: Non-SORT switch

Explanation: User specified a SORT command line switch which is only valid for MERGE.

Callable returned error buffer: First word = -72
Second word = offset into MCR command line

%SORT_F_NSFBUF: Out of work file buffer space

Explanation: This is probably an internal error; please submit an SPR.

Callable returned error buffer: First word = -30

%SORT_F_NSFRAB: Out of work file RAB space

Explanation: This is probably an internal error; please submit an SPR.

Callable returned error buffer: First word = -29

%SORT_F_NUFBUF: Out of user file buffer space

Explanation: This is probably an internal error; please submit an SPR.

Callable returned error buffer: First word = -34

%SORT_F_NUFRAB: Out of user file RAB space

Explanation: This is probably an internal error; please submit an SPR.

Callable returned error buffer: First word = -33

%SORT_W_NUMTRN: Number of records truncated:

Explanation: See warning message %SORT_W_TRNREC.

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

%SORT_F_ODADTR: Odd address trap

Explanation: Register dump will follow. Please submit with SPR.

Callable returned error buffer: No return to callable.

%SORT_F_OPIF00: Error opening input file: RMS codes

Explanation: RMS could not open the given input file.

Callable returned error buffer:
First word = -43
Second word = RMS STS code
Third word = RMS STV code
Fourth word = Input file number

%SORT_F_OPOF00: Error opening output file: RMS codes

Explanation: RMS could not open the given output file.

Callable returned error buffer:
First word = -49
Second word = RMS STS code
Third word = RMS STV code

%SORT_F_OPSP00: Error opening specification file: RMS codes

Explanation: RMS could not open the given specification file.

Callable returned error buffer:
First word = -54
Second word = RMS STS code
Third word = RMS STV code

%SORT_F_RDIF00: Error reading from input file: RMS codes

Explanation: RMS failed while trying to read the given input file.

Callable returned error buffer:
First word = -45
Second word = RMS STS code
Third word = RMS STV code
Fourth word = Input file number

%SORT_F_RDSF00: Error reading from work file: RMS codes

Explanation: RMS failed while trying to read the given work file.

Callable returned error buffer:
First word = -39
Second word = RMS STS code
Third word = RMS STV code
Fourth word = Work file number

%SORT_F_RDSP00: Error reading from specification file: RMS codes

Explanation: RMS failed while trying to read the specification file.

Callable returned error buffer:
First word = -56
Second word = RMS STS code
Third word = RMS STV code

%SORT_F_RLFAIL: Failure to release allocated pool block

Explanation: This is probably an internal error; please submit an SPR.

Callable returned error buffer: First word = -27

%SORT_F_RSFAIL: Failure to allocate requested pool block

Explanation: This is probably an internal error; please submit an SPR.

Callable returned error buffer: First word = -28

%SORT_F_RWSF00: Error rewinding work file: RMS codes

Explanation: RMS could not rewind the given work file.

Callable returned error buffer: First word = -41
Second word = RMS STS code
Third word = RMS STV code
Fourth word = Work file number

%SORT_F_SPCADJ: Invalid collating sequence definition

Explanation: Collating sequence in specification file is not valid.

Callable returned error buffer: First word = -90

%SORT_F_SPCCHR: Invalid character definition

Explanation: Character definition in specification file is not valid.

Callable returned error buffer: First word = -92

%SORT_W_SPCIVC: Invalid collating sequence, on line

Explanation: Collating sequence in specification file is not valid.

Callable returned error buffer: First word = 6
Second word = specification file line number

%SORT_W_SPCIVD: Invalid data type, on line

Explanation: Data type found in specification file is not valid.

Callable returned error buffer: First word = 7
Second word = specification file line number

%SORT_W_SPCIVF: Invalid field, on line

Explanation: Field definition in specification file is not valid.

Callable returned error buffer: First word = 8
Second word = specification file line number

%SORT_W_SPCIVI: Invalid include or omit, on line

Explanation: Include or omit definition in specification file is not valid.

Callable returned error buffer: First word = 9
Second word = specification file line number

%SORT_W_SPCIVK: Invalid key or data, on line

Explanation: Key definition in specification file is not valid.

Callable returned error buffer: First word = 10
Second word = specification file line number

%SORT_W_SPCIVP: Invalid sort process, on line

Explanation: Sort process found in specification file is not valid.

Callable returned error buffer: First word = 11
Second word = specification file line number

%SORT_W_SPCIVS: Invalid specification line

Explanation: Bad specification in specification file.

Callable returned error buffer: First word = 12
Second word = specification file line number

%SORT_W_SPCIVX: Invalid condition, on line

Explanation: Condition definition in specification file is not valid.

Callable returned error buffer: First word = 13
Second word = specification file line number

%SORT_W_SPCMIS: Invalid merge specification, on line

Explanation: Specification given in MERGE specification file that is only valid for SORT.

Callable returned error buffer: First word = 14
Second word = specification file line number

%SORT_W_SPCOVR: Specification overridden, on line

Explanation: Specification has been overridden by command line or callable parameter.

Callable returned error buffer: First word = 15
Second word = specification file line number

%SORT_F_SPCPAD: Invalid pad character

Explanation: Pad character definition in specification file is not valid.

Callable returned error buffer: First word = -93

%SORT_F_SPCPLX: Collating sequence too complex

Explanation: In the specification file, the user tried to define a collating sequence that had too many different collating values.

Callable returned error buffer: First word = -91

%SORT_W_SPCSIS: Invalid sort specification, on line

Explanation: Specification given in SORT specification file that is only valid for MERGE.

Callable returned error buffer: First word = 16
Second word = specification file line number

%SORT_F_SPCTHR: Cannot define three-byte collating value

Explanation: User attempted to define three-byte collating value in specification file collating sequence clause.

Callable returned error buffer: First word = -94

%SORT_W_SUCCESS:

Explanation: Success returned from callable interface. If this message occurs in utility, please submit SPR.

Callable returned error buffer: First word = 0

%SORT_W_TRNREC: Truncating records longer than specified LRL of

Explanation: At least one record in an input file was longer than the LRL specified by the user. All such records will be truncated.

Callable returned error buffer: First word = 17
Second word = specified LRL

%SORT_F_TRPTRP: TRAP instruction execution

Explanation: Register dump will follow. Please submit with SPR.

Callable returned error buffer: No return to callable.

%SORT_F_WKAREA: Insufficient work area (bytes):

Explanation: Work area supplied is insufficient for the SORT or MERGE.

Callable returned error buffer: First word = -26
Second word = number of bytes of work area supplied

%SORT_F_WROF00: Error writing to output file: RMS codes

Explanation: RMS failed while trying to write to the given output file.

Callable returned error buffer: First word = -51
Second word = RMS STS code
Third word = RMS STV code

%SORT_F_WRSF00: Error writing to work file

Explanation: RMS failed while trying to write to the given work file.

Callable returned error buffer: First word = -38
Second word = RMS STS code
Third word = RMS STV code
Fourth word = Work file number

%SORT_F_WRTI00: Error writing to terminal device

Explanation: RMS failed while trying to write to the terminal device.

Callable returned error buffer: First word = -97

%SORT_W_WRTSHR: Input file opened allowing writes - file number

Explanation: Warning that the input file could be modified during the SORT or MERGE.

Callable returned error buffer: First word = 5
Second word = input file number

%SORT_F_XSLUNS: Too many LUNs required (required/max allowed):

Explanation: The total number of LUNs required because of the number of input, output, work, and specification files exceeds the maximum allowed on the system.

Callable returned error buffer: First word = -32
Second word = number of LUNs required
Third word = maximum LUNs allowed

%SORT_F_ZMGORD: Invalid merge order

Explanation: User passed a zero merge order to a callable record merge.

Callable returned error buffer: First word = -74

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.

There are six sample programs in this appendix.

- 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).

BASIC-PLUS-2 program using the MERGE file interface

```
1  EXTEND
10 DECLARE                          &
        INTEGER                     &
            lun_buf(0%),            &
                err_buff(4%)
20 DECLARE                          &
        STRING                      &
            command_line            &
    !
40 MAP( WORK )                      &
       STRING wrk_area = 15000      &
    !
1910 command_line = "output=bmsfil/fo:v:80,bmsfil1/fo:v:80/ke:co1,1/ss" &
    !
1920 lun_buf(0) = 6%                &
    !
1990 PRINT "calling MRGINB"
2000 CALL MRGINB (err_buff() 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_buff(0) = 0 THEN GOTO 2300                  &
    !
2110 PRINT "error in MRGINB:  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 30000                                         &
    !
2300 PRINT "calling MRGMRB"
3000 CALL MRGMRB (err_buff() BY REF)
3100 IF err_buff(0) = 0 THEN GOTO 4990                  &
    !
3110 PRINT "error in MRGMRB:  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 30000                                         &
    !
4990 PRINT "calling MRGENB"
5000 CALL MRGENB (err_buff() BY REF)                    &
    !
5105 IF err_buff(0) = 0 THEN GOTO 30000                 &
    !
5110 PRINT "error in MRGENB:  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)   &
    !
30000 END
```

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:co1,1/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:co1,1/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:co1,1" &
     !
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 )                                 &
     !
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
```

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.
```

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.
```

FORTRAN program using the MERGE file interface

```
            PROGRAM FMGFIL
C
C           THIS PROGRAM TESTS THE MERGE FILE INTERFACE.
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
      INTEGER*2 ILUNLN
C               LUN BUFFER LENGTH
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
      INTEGER*2  INPSIZ, LENGTH, INRECS, OUTRCS
C                TOTAL   INPUT   INPUT   OUTPUT
C                INPUT   RECORD  RECORD  RECORD
C                FILES   LENGTH  COUNT   COUNT
C                SIZE
C
      INTEGER*2  ENDFIL
C                END OF FILE FLAG
C
C
      CHARACTER RUTINE*8,  COMAND*55,   A(80)
C               ROUTINE    MERGE        MISC.
C               RETURNING  COMMAND      STRING
C               THE ERROR  LINE
C
                EXTERNAL MRGINI, MRGMRG, MRGEND
C
C              MERGE-11 SUBROUTINES
C
C
      DATA I1LUN,ILUNLN,IWKSIZ/6,1,16000/
      DATA MAXREC,IERROR/20,0,0,0,0/
      COMAND = 'OUT=FMGFIL/FO:F:20,FMGFL1/FO:F:20/KE:C01.20'
      ICOMLN = 43
```

```
C
C                  INITIALIZE MERGE PARAMETERS
C
        RUTINE=' MRGINI '
        TYPE 902, RUTINE
        CALL MRGINI(IERROR,IWORK,IWKSIZ,COMAND,ICOMLN,MAXREC,0,0,0,
     X                  I1LUN,ILUNLN)
C
C             CALL THE MERGE-11 INITIALIZE ROUTINE
C
        IF (IERROR(1) .NE. 0) THEN
                GOTO 900
                ENDIF
C
C       EXIT AND TYPE AN ERROR MESSAGE IF MRGINI WAS UNSUCCESSFUL.
C
C
        INRECS = 0
        OUTRCS = 0
        ENDFIL = 0
        IFILE = 1
C
C
C
        RUTINE=' MRGMRG '
        TYPE 902, RUTINE
150     CALL MRGMRG(IERROR)
C
C            START THE MERGING PROCEDURE
C
        IF (IERROR(1) .NE. 0) THEN
                GOTO 900
                ENDIF
C
C       EXIT AND TYPE AN ERROR MESSAGE IF MRGRLS WAS UNSUCCESSFUL.
C
        RUTINE=' MRGEND '
        TYPE 902, RUTINE
        CALL MRGEND(IERROR)
C
C             CALL THE MERGE-11 CLEAN-UP ROUTINES.
C
C
        IF (IERROR(1) .NE. 0) THEN
                GOTO 900
                ENDIF
C
C       EXIT AND TYPE AN ERROR MESSAGE IF UNSUCCESSFUL.
C
        STOP 'SUCCESSFUL FORTRAN MERGE TEST.'
C
C
C
C
C
900     TYPE 901, RUTINE
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 MERGE FROM FORTRAN.'
      END
```

FORTRAN program using the SORT file interface

```
      PROGRAM FORFIL
C
C           THIS PROGRAM TESTS THE SORT FILE INTERFACE.
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
      INTEGER*2 ILUNLN
C               LUN BUFFER LENGTH
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
      INTEGER*2 INPSIZ,  LENGTH,  INRECS,  OUTRCS
C               TOTAL    INPUT    INPUT    OUTPUT
C               INPUT    RECORD   RECORD   RECORD
C               FILES    LENGTH   COUNT    COUNT
C               SIZE
C
      INTEGER*2 ENDFIL
C               END OF FILE FLAG
C
C
      CHARACTER RUTINE*8,  COMAND*45,   A(80)
C               ROUTINE    SORT         MISC.
C               RETURNING  COMMAND      STRING
C               THE ERROR  LINE
C
      EXTERNAL SRTINI, SRTSRT, SRTEND
C
C              SORT-11 SUBROUTINES
C
C
      DATA I1LUN,ILUNLN,IWKSIZ/6,1,16000/
      DATA MAXREC,IERROR/20,0,0,0,0/
      COMAND = 'OUT=FORFIL/FO:F:20/KE:CN1,20/FI:3/PT:30'
      ICOMLN = 39
C
C              INITIALIZE SORT PARAMETERS
C
      RUTINE=' SRTINI '
      TYPE 902, RUTINE
      CALL SRTINI(IERROR,IWORK,IWKSIZ,COMAND,ICOMLN,MAXREC,0,0,
     X                I1LUN,ILUNLN)
```

```
C
C                CALL THE SORT-11 INITIALIZE ROUTINE
C
      IF (IERROR(1) .NE. 0) THEN
              GOTO 900
              ENDIF
C
C       EXIT AND TYPE AN ERROR MESSAGE IF SRTINI WAS UNSUCCESSFUL.
C
C
      INRECS = 0
      OUTRCS = 0
      ENDFIL = 0
      IFILE = 1
C
C
C
      RUTINE=' SRTRLS '
      TYPE 902, RUTINE
150      CALL SRTSRT(IERROR)
C
C                START THE SORTING PROCEDURE
C
      IF (IERROR(1) .NE. 0) THEN
              GOTO 900
              ENDIF
C
C       EXIT AND TYPE AN ERROR MESSAGE IF SRTRLS WAS UNSUCCESSFUL.
C
      RUTINE=' SRTEND '
      TYPE 902, RUTINE
      CALL SRTEND(IERROR)
C
C                CALL THE SORT-11 CLEAN-UP ROUTINES.
C
C
      IF (IERROR(1) .NE. 0) THEN
              GOTO 900
              ENDIF
C
C       EXIT AND TYPE AN ERROR MESSAGE IF UNSUCCESSFUL.
C
      STOP 'SUCCESSFUL FORTRAN/SORT TEST.'
C
C
C
C
C
900      TYPE 901, RUTINE
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
```

Appendix C
Specification File Translator

C.1 Converting Version 2 Specification Files

You can convert specification files that used the format of the previous version of SORT-11 by using the translator utility provided with Version 3. To invoke the translator utility, use either of the following from the MCR/CCL interface:

- TRN output-file = input-file

  or

- TRN <RET>
  TRN> output-file = input-file

Input-file is the name of the specification file from a previous version. Output-file is the name of the specification file (having the new format) that the translator utility creates. There is no default file type for the input file; the default file type for the new output specification file is SRT.

The translator utility includes the specifications from the old format in the file with the new format, setting them off with comment markers. Any errors found during the conversion of the old specification file are displayed on your output device.

C.2 Translator Error Messages

The error messages listed in this section are generated by the specification file translator. When the translator encounters an error, it displays the message code (for example, SRTTRN-I-ADDINCKEY) followed by a brief explanation (for example, 'Keys added to avoid comparing incompatible keys'). The error message is displayed on your default output device.

For some of the error messages listed, a more detailed explanation is provided in this appendix. Other messages refer to an error within a given line of the specification file. Although the line number containing the error will not be specified, you can identify the line causing an error by directing the output from the translator to a terminal, and then noting where the errors are issued.

If a message is given indicating a syntax error in the old specification file, but the old specification file works with SORT-11 V2.0, please notify DIGITAL by submitting an SPR.

SRTTRN-I-ADDINCKEY, Keys added to avoid comparing incompatible keys

Multiple record formats as specified might cause incompatible keys to be compared. To avoid this, constant keys are added which ensure that the incompatible keys will never be compared.

SRTTRN-F-BADLOGIC, Internal logic error detected

Please submit an SPR.

SRTTRN-E-CLOSEDEL, Error closing <file_descriptor>
SRTRTN-I-ADDSTS, additional status <sts> / <stv>

See RMS documentation for meaning of STS and STV codes.

SRTTRN-E-CLOSEIN, Error closing <file_descriptor> as input
SRTRTN-I-ADDSTS, additional status <sts> / <stv>

See RMS documentation for meaning of STS and STV codes.

SRTTRN-E-CLOSEOUT, Error closing <file_descriptor> as output
SRTRTN-I-ADDSTS, additional status <sts> / <stv>

See RMS documentation for meaning of STS and STV codes.

SRTTRN-E-COMPLEX, Specification file is very complex - verify output

It is unlikely that you will encounter this error. It occurs when over 255 different record formats having incompatible key data types are specified in your specification file. See error message SRTTRN-I-ADDINCKEY.

SRTTRN-F-INSVIRMEM, Insufficient virtual memory

Old specification file was too long or too complex to be translated using available work area.

SRTTRN-F-INV_ALT, Invalid alternate collating sequence field in header

SRTTRN-F-INV_CONST, Invalid F/C field in include or omit line

SRTTRN-F-INV_CONT, Invalid continuation field in include or omit line

SRTTRN-F-INV_DATA, Invalid data type field in include, omit, or field line

SRTTRN-F-INV_FIELD, Invalid field type in field line

SRTTRN-E-INV_LIMIT, Invalid field limit in include, omit, or field line

SRTTRN-E-INV_PROCESS, Invalid sort process field in header line

SRTTRN-F-INV_QUAL, Invalid qualifier

You included a command line qualifier when you invoked the translator. There are no valid qualifiers.

SRTTRN-F-INV_SEQ, Invalid sequence field in header line

SRTTRN-F-INV_STRIP, Invalid key strip field in header line

SRTTRN-F-INV_TYPE, Invalid line type in specification file

SRTTRN-W-LINE_NUM, Line numbers out of sequence

SRTTRN-F-NO_COMMAND, Error reading command line

Translator was unable to obtain command line from system.

SRTTRN-F-NO_FORCE_TO, Missing force column information in force line

SRTTRN-W-NO_HEADER, Missing header line in specification file

Old specification file is missing the header line.

SRTTRN-W-NO_PAIRS, No replacement pairs in altseq line

SRTTRN-F-OPENIN, Error opening <file_descriptor> as input

SRTRTN-I-ADDSTS, additional status <sts> / <stv>

See RMS documentation for meaning of STS and STV codes.

SRTTRN-F-OPENOUT, Error opening <file_descriptor> as output
SRTRTN-I-ADDSTS, additional status <sts> / <stv>

See RMS documentation for meaning of STS and STV codes.

SRTTRN-E-READERR, Error reading <file_descriptor>
SRTRTN-I-ADDSTS, additional status <sts> / <stv>

See RMS documentation for meaning of STS and STV codes.

SRTTRN-F-SYSERROR, System service error

Should not occur; please submit an SPR.

SRTTRN-W-UNX_ALTSEQ, Unexpected altseq line, see column 26 of header

SRTTRN-F-UNX_FORCE, Unexpected continuation of force key

SRTTRN-E-WRITEERR, Error writing <file_descriptor>
SRTRTN-I-ADDSTS, additional status <sts> / <stv>

See RMS documentation for meaning of STS and STV codes.

Appendix D
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
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

HEX Code	Octal Code	Decimal Code	Char or Abbrev.	Description
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
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 accent
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

HEX Code	Octal Code	Decimal Code	Char or Abbrev.	Description
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
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 accent
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
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

HEX Code	Octal Code	Decimal Code	Char or Abbrev.	Description
75	165	117	u	lowercase u
55	125	085	U	uppercase U
F9	371	249	ù	lowercase u with grave accent
D9	331	217	Ù	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	Ü	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
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 started 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

HEX Code	Octal Code	Decimal Code	Char or Abbrev.	Description
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
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	a	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	o	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

Appendix E
Compatibility with PDP-11 SORT/MERGE Version 2

With some minor changes, the PDP-11 SORT/MERGE Version 3.0 SORT stand-alone utility package is upwardly compatible with SORT-11 Version 2.0. The incompatibilities detailed below were made necessary by the nature of the new functionality and design of Version 3.

The installation procedure for PDP-11 SORT/MERGE Version 3.0 leaves SORT-11 Version 2.0 intact on your system while installing the Version 3.0 product. While the Version 2.0 task image remains on your system, typing in SRT or issuing the DCL command SORT will call Version 3.0 instead of Version 2.0.

The callable subroutines shipped with PDP-11 SORT/MERGE Version 3.0 are not compatible with the SORT-11 Version 2.0 callable subroutines. The Version 2.0 subroutines are left on your system during the PDP-11 SORT/MERGE Version 3.0 installation. Since the entry point names are different in the two versions, there is no conflict.

In PDP-11 SORT/MERGE Version 3.0 the DCL /SPECIFICATION qualifier or MCR/CCL /SF switch is required for specification files. In SORT-11 Version 2.0, SORT accepted only a single input file. If a second input file was listed, it was assumed to be a specification file. In version 3.0, SORT and MERGE allow multiple input files. The /SPECIFICATION qualifier or /SF switch is mandatory for specification files in order for SORT/MERGE to identify them as such.

In the SORT-11 Version 2.0 the /FO switch is required for the input file. In PDP-11 SORT/MERGE Version 3.0, this switch, or the /FORMAT qualifier in DCL, is optional. If it is not specified, the information is assumed to be supplied by RMS-11 when the files are opened. If record length information cannot be obtained by SORT/MERGE, and the /FO switch or /FORMAT qualifier is not present, SORT/MERGE terminates with a fatal error. Additionally, the Version 2.0 input record format designation is no longer meaningful. The Version 3.0 command parser will accept them to ensure upward compatibility, but gets the record format information from RMS-11 when the files are opened. If specified, they must represent a valid Version 2.0 input record format.

The SORT-11 Version 2.0 specification files will not work with PDP-11 SORT/MERGE Version 3.0. The translator utility supplied with Version 3.0 will convert Version 2.0 specification files to the new Version 3.0 format. The translator utility is described in Appendix C.

When using a SORT-11 Version 2.0 specification file, the default condition for key stripping is not to strip the keys from the output record. In PDP-11 SORT/MERGE Version 3.0, keys are not prefixed to the output record. However, the same results can be obtained by using /FIELD and /DATA statements in the specification file. Appropriate changes to a Version 2.0 specification file are made by the specification file translator so that the intended processing is done.

Through the use of the specification file, SORT-11 Version 2.0 allows the comparison of an ascending key with a descending key. This is a Version 2.0 bug which has been fixed for Version 3.0.

In SORT-11 Version 2,0, you could use key data type "D" for FORTRAN data formats E, F, D, G and H. In PDP-11 SORT/MERGE Version 3.0, a new key data type, "A" (ASCII_FLOATING), has been introduced for these FORTRAN formats. Key data type "D" is no longer appropriate for these data types.

SORT-11 Version 2.0 does not handle multiword binary data correctly. The "B" designation for data type was designed to handle PDP-11 COBOL COMPUTATIONAL-6 type data, and normal multiword binary was sorted improperly. In PDP-11 SORT/MERGE Version 3.0, two new data types have been introduced to handle normal binary (integer) data. These are "S" (SIGNED BINARY) and "U" (UNSIGNED BINARY). Data type "B" has been implemented as it was in Version 2.0 for compatibility, but it can only be used at the MCR/CCL level. No support for data type "B" exists at the DCL level.

Index

A

Address sort, 2–10
Application program
  calling standard, 4–2
  calling subroutines, 4–3
  choosing between file and record interface, 4–3
  generating SORT/MERGE statistics from, 5–6
  initializing, 4–5
  MERGE file interface subroutines, 4–15
  MERGE mixed-mode interface subroutines, 4–16
  MERGE record interface subroutines, 4–16
  MERGE subroutines, 4–15t
  passing parameters in BASIC and COBOL, 4–2
  sample
    BASIC-PLUS-2 (MERGE), B–2
    BASIC-PLUS-2 (SORT and MERGE), B–3
    COBOL-81 (MERGE), B–6
    COBOL-81 (SORT), B–8
    FORTRAN (MERGE), B–11
    FORTRAN (SORT), B–13
  SORT file interface subroutines, 4–4
  SORT mixed-mode interface subroutines, 4–5
Application program (Cont.)
  SORT record interface subroutines, 4–5
  SORT subroutines, 4–4t
  statistics, space required, 5–6
  subroutine location, 4–2
  supported languages, 4–1
  task building, 4–21
  using a specification file with, 3–3
  using MERGE, 4–14
  using specification file text with, 3–3
  using your own routines, 4–3
Ascending order
  specifying, 2–5

B

Batch processing, 1–7
BINARY (data type)
  size rules for, 2–8
Block size, specifying for output file, 2–16
Bucket size, specifying, 5–9

C

Callable subroutines
  see Application program or individual subroutine
Chaining (RSTS/E only), 2–16
CHARACTER (data type)
  size rules for, 2–8

Collating sequence
default, 2-8
defining in a specification file, 3-11
defining your own, 3-11
example of, 3-14
modifying, 3-13
multinational, D-1
specifying, 2-8
using IGNORE clause in, 3-14
Command line buffer parameter, 4-7
Command line length parameter, 4-8
Comparison routine address parameter
MERGE, 4-18
SORT, 4-10
CONDITION clause
example of, 3-7
using in a specification file, 3-7
using with TEST, 3-8

D

DATA statement
in a specification file, 3-6
using to format output data, 3-6
Data types
BINARY, size rules for, 2-8
CHARACTER, size rules for, 2-8
DECIMAL, size rules for, 2-8
determining, 2-6
floating point, size rules for, 2-8
size rules for, 2-8
specifying in DCL, 2-7
supported, 2-6
DCL
data types, 2-6
default file types, 2-3
qualifiers
see Qualifiers
sequence checking, 2-17
specifying a collating sequence, 2-8
specifying file size, 2-13
specifying key field position in, 2-4
specifying key field size in, 2-4
specifying record size, 2-12
specifying work area division, 2-13
using a specification file with, 3-2
DCL command line
MERGE, 1-6
SORT, 1-2
DECIMAL (data type)
size rules for, 2-8
Descending order
specifying, 2-5
DIGITS qualifier
using in a specification file, 3-5

E

EBCDIC collating sequence
specifying, 2-8
Equal keys
ordering of, 2-9
using NODUPLICATES option with, 2-9
using STABLE option with, 2-9
Equal to, in TEST clause, 3-8
Equal-key routine address parameter
MERGE, 4-19
SORT, 4-11
Error address parameter, 4-6
Error messages
See Appendix A

F

FIELD clause
in a specification file, 3-4
Field name
in specification file, 3-4
rules for defining, 3-4
Fields
identifying in a specification file, 3-4
identifying position of, 2-4
identifying size of, 2-4
File attributes
specifying, 2-12
File format
rules for identifying, 2-12
File interface
SORT subroutines, 4-4
using in an application program, 4-3
File names
default for MCR/CCL, 2-3
File size
default allocation, 2-13
maximum, 2-13
specifying, 2-13
File support, 1-2
File type
default for DCL, 2-3
default for MCR/CCL, 2-3
Fill factor, specifying, 5-9
Floating point (data type)
size rules for, 2-8
FOLD, in a specification file, 3-12

G

Greater than, in TEST clause, 3-8

I

I/O requirements
  adapting work area for, 2-13
IGNORE clause, in a specification file, 3-14
INCLUDE statement
  using in a specification file, 3-8
Index sort, 2-10
Indexed-sequential files
  maximum LRL, 2-12
  rules for, 2-13
  specifying, 2-13
  using /OVERLAY with, 2-15
  writing records to, 2-15
Input file size parameter, 4-9
  calculating default values for, 4-9
Input files
  maximum number of, 1-2
  multiple, 1-5
  size allocation, 5-9
  specifying, 1-5
Input routine address parameter, 4-18

K

KEY clause
  in a specification file, 3-5
Key field
  specifying position of, 2-4
KEY qualifier
  example, 1-4
Keys
  order of information in MCR/CCL, 2-7
  specifying, 2-4
Keywords (specification file), 3-2t

L

Language support (application programs), 4-1
Less than, in TEST clause, 3-8
Logical unit
  see LUN
Longest Record Length
  see LRL
Longest record length parameter, 4-8
LRL
  calculating with multiple input files, 2-12
LRL (Cont.)
  maximum, by file organization type, 2-12
  specifying, 2-12
LUN buffer length parameter, 4-9
LUN buffer parameter, 4-9
  default values for, 4-9
LUN usage
  MERGE, 4-25
  SORT, 4-25

M

MCR/CCL
  chaining to another executable image (RSTS/E only), 2-16
  command line, 2-1
  data types, 2-7
  default file names and type, 2-3
  default values, 2-3
  key information, 2-7
  multiple key syntax rules, 2-6
  sequence checking, 2-17
  specifying a collating sequence, 2-8
  specifying file size, 2-13
  specifying key field position in, 2-4
  specifying key field size in, 2-4
  specifying keys
    order of information, 2-7
  specifying record format, 2-13
  specifying record size, 2-13
  specifying work area division, 2-13
  switches, 2-2
    see also Switches
  using a specification file with, 3-2
  using MERGE with, 2-1
  using SORT with, 2-1
  using specification file translator from, C-1
MERGE
  choosing a sort process, 2-17
  DCL command line, 1-6
  file interface, subroutines, 4-15
  MCR/CCL command line, 2-1
  mixed-mode interface, subroutines, 4-16
  record interface, subroutines, 4-16
  specifying work files with, 2-17
  using with an application program, 4-14
Merge order parameter, 4-18
Mixed-mode interface
  SORT subroutines, 4-5
  using in an application program, 4-3

MODIFICATION clause, in a
specification file, 3–13
MRGCLB subroutine, 4–19
MRGCMP subroutine, 4–18
MRGINB subroutine, 4–16
MRGINC subroutine, 4–16
MRGINI
description, 4–16
parameters, 4–17
MRGINP subroutine, 4–18
MRGWRN subroutine, 4–18
Multinational character set, D–1
MULTINATIONAL collating sequence
ordering procedure, 2–8
specifying, 2–8
Multiple keys
identifying a primary key, 2–5
identifying a secondary key, 2–5
MCR/CCL syntax rules, 2–6
specifying, 2–5

N

Not equal to, in TEST clause, 3–8

O

ODL files
list of, 4–22t
memory requirements, 4–22t
referencing, 4–21
selecting, 4–21
using with SORT/MERGE resident
library, 4–23
OMIT statement
using in a specification file, 3–8
Order
specifying ascending or descending, 2–5
Output file
format, 2–14
organization, 2–14
size allocation, 5–9
with different record format than first
input file, 2–15
Overlay descriptor language files
see ODL files

P

Pad character, specifying, 3–16
PAD statement
in a specification file, 3–16
Parameters (subroutines)
command line buffer, 4–7
Parameters (subroutines) (Cont.)
command line length, 4–8
comparison routine address (MERGE),
4–18
comparison routine address (SORT),
4–10
equal-key routine addess (MERGE),
4–19
equal-key routine address (SORT),
4–11
error address, 4–6
input file size, 4–9
input routine address, 4–18
longest record length, 4–8
LUN buffer, 4–9
LUN buffer length, 4–9
merge order, 4–18
record buffer, 4–12
record length, 4–12
record location, 4–13
specification file buffer, 4–8
specification file buffer length, 4–9
warning routine address (MERGE),
4–18
warning routine address (SORT), 4–10
with MERGE, 4–14
work area address (MERGE), 4–17
work area address (SORT), 4–6
work area length, 4–7
Position
specifying in DCL, 2–4
specifying in MCR/CCL, 2–4
POSITION subqualifier, 2–4
example, 1–4
identifying, 1–4
Primary key
definition, 1–4
identifying, 2–5
identifying in a specification file, 3–6
Programming
see Application program

Q

Qualifiers
abbreviating, 2–6
/BUCKET_SIZE, 5–9
/CHECK_SEQUENCE, 2–17
/COLLATING_SEQUENCE, 2–8
/DUPLICATES, 2–9
/FORMAT, 2–12
/INDEXED_SEQUENTIAL, 2–13,
2–14
/KEY, 2–4

Qualifiers (Cont.)
list of, 2-2t
/LOAD_FILL, 5-9
/NOCHECK_SEQUENCE, 2-17
/NODUPLICATES, 2-9
/NOSHAREABLE, 2-14
/NOSTABLE, 2-9
/OVERLAY, 2-15
/PROCESS, 2-10
purpose, 1-3
/RELATIVE, 2-14
/SEQUENTIAL, 2-14
/SHAREABLE, 2-14
specifying, 1-3
/STABLE, 2-9
/STATISTICS, 5-4
summary (for MERGE), 2-22t, 2-24t
summary (for SORT), 2-18t
/TREE_SPACE, 2-13
/WORK_FILES
using to improve performance, 5-3

R

Radix operators, in a specification file, 3-12
Record buffer parameter, 4-12
Record interface
SORT subroutines, 4-5
using in an application program, 4-3
Record length parameter, 4-12
Record location parameter, 4-13
Record size
specifying, 2-12
Record sort, 2-10
Records
using CONDITION to selectively include, 3-8
Relative files
maximum LRL, 2-12

S

Secondary key
definition, 1-4
example, 1-4
identifying, 2-5
identifying in a specification file, 3-6
Sequence checking, 2-17
Sequential files
maximum LRL, 2-12
Shareable files, specifying, 2-14
Size
specifying in DCL, 2-4
Size (Cont.)
specifying in MCR/CCL, 2-4
SIZE subqualifier, 2-4
example, 1-4
SORT
cleanup phase, 5-4
DCL command line, 1-2
file interface, subroutines, 4-4
initialization phase, 5-2
MCR/CCL command line, 2-1
merge phase, 5-4
mixed-mode interface, subroutines, 4-5
record interface, subroutines, 4-5
replacement selection algorithm, 5-2
SORT phase, 5-2
Sort process
default, 2-10
description, 2-10t, 2-10
performance considerations, 5-10
selection criteria, 2-11
specifying, 2-10
use of temporary storage space, 2-11
with MERGE, 2-17
Specification file
abbreviating keywords in, 3-5
CHECK_SEQUENCE statement, 3-4
default file type, 3-2
defining a collating sequence in, 3-11
defining your own collating sequence in, 3-11
evaluating field position with, 3-4
evaluating field size (length) with, 3-4
format of, 3-3
formatting output data in, 3-6
identifying data type with, 3-4
identifying field names in, 3-4
keywords, 3-2t
modifying a collating sequence in, 3-13
NOCHECK_SEQUENCE statement, 3-4
NOSTABLE statement, 3-4
order of instructions, 3-2
placing in an application program buffer, 4-8
PROCESS statement, 3-3
reassigning work files from, 3-15
sample, 3-18
separating subqualifiers in, 3-5
sorting records with varying formats, 3-9
specifying a pad character from, 3-16
specifying descending order in, 3-6
specifying key fields, 3-5
specifying record fields in, 3-4

Specification file (Cont.)
STABLE statement, 3-4
summary of statements and qualifiers, 3-16t
upgrading from Version 2, C-1
using /KEY command line qualifier with, 3-10
using a comment character in, 3-3
using command line qualifiers with, 3-2
using conditional statements in, 3-7
using FOLD with, 3-12
using from an application program, 3-3
using IGNORE clause in, 3-14
using radix operators in, 3-12
using the DIGITS subqualifier, 3-5
using TIE_BREAK with, 3-12
using to combine upper and lower case characters, 3-12
using with DCL interface, 3-2
using with MCR/CCL, 3-2
WORK_FILES statement, 3-15
Specification file buffer length parameter, 4-9
Specification file buffer parameter, 4-8
Specification file translator, C-1 to C-4
error messages, C-1
using, C-1
SRTCLB subroutine, 4-11
SRTCMP subroutine, 4-11
SRTEND
description, 4-14
parameter, 4-14
SRTINI
description, 4-5
parameters, 4-6t
SRTRLS
description, 4-12
parameters, 4-12t
SRTRTN
description, 4-12
parameters, 4-13t
SRTSRT
description, 4-13
parameters, 4-14
SRTWRN subroutine, 4-10
Statistics
collating sequence, 5-4
elapsed time, 5-4
Fibonacci level, 5-6
generating from an application program, 5-6
identification, 5-4
initial I/O area size, 5-6
Statistics (Cont.)
input files, 5-4
key size, 5-5
longest input record, 5-5
merge order, 5-6
node size, 5-5
number of initial runs, 5-6
number of keys, 5-5
number of nodes, 5-6
number of records input, 5-5
number of records output, 5-5
process, 5-4
records omitted, 5-5
total input allocation, 5-5
total output allocation, 5-5
total work allocation, 5-5
using the DCL qualifier, 5-4
work files, 5-4
Subqualifiers
abbreviating, 2-6
specifying, 1-3
using more than one, 1-3
Subroutines
calling, 4-3
location, 4-2
MERGE file interface, 4-15
MERGE mixed-mode interface, 4-16
MERGE record interface, 4-16
MERGE, summary of, 4-15t
MRGCLB, 4-19
MRGCMP, 4-18
MRGINB, 4-16
MRGINC, 4-16
MRGINI, 4-16
MRGINP, 4-18
MRGWRN, 4-18
SORT file interface, 4-4
summary, 4-19t
SORT mixed-mode interface, 4-5
summary, 4-20t to 4-21t
SORT record interface, 4-5
summary, 4-19t
SORT, summary of, 4-4t
SRTCLB, 4-11
SRTCMP, 4-10
SRTENB, 4-14
SRTENC, 4-14
SRTEND, 4-14
SRTINB, 4-5
SRTINC, 4-5
SRTINI, 4-5
SRTRLB, 4-12
SRTRLC, 4-12
SRTRLS, 4-12

Subroutines (Cont.)
SRTRTB, 4-12
SRTRTC, 4-12
SRTRTN, 4-12
SRTSRB, 4-13
SRTSRC, 4-13
SRTSRT, 4-13
SRTWRN, 4-10
using your own, 4-3
comparison routine (MERGE), 4-18
comparison routine (SORT), 4-10
equal-key routine (MERGE), 4-19
equal-key routine (SORT), 4-11
task building, 4-23
warning routine (MERGE), 4-18
warning routine (SORT), 4-10
Switches
/BK, 2-13
/BU, 5-9
/-CH, 2-17
/CH, 2-17
/CN, 2-16
/CS, 2-8
/FI
using to improve performance, 5-3
/FO, 2-13
/IN, 2-13, 2-14
/KE, 2-4
list of, 2-2t
/LO, 5-9
/ND, 2-9
negative, 2-2
/OV, 2-15
/PR, 2-10
/PT, 2-13
/RE, 2-14
/SE, 2-14
Switches (Cont.)
/SF, 4-8
/SH, 2-14
/SS, 5-4
/-ST, 2-9
/ST, 2-9
summary (for SORT), 2-20t
using in MCR/CCL, 2-2

T

Tag sort, 2-10
Task building, 4-21 to 4-25
with your own subroutines, 4-23
TEST clause
defining conditions with, 3-8
example of, 3-8
using with CONDITION, 3-8
TIE_BREAK, in a specification file, 3-12

W

Warning routine address parameter
MERGE, 4-18
SORT, 4-10
Work area
adapting for I/O requirements, 2-13
division of, 2-13
Work area address parameter
calculating size for, 4-7
MERGE, 4-17
SORT, 4-6
Work area length parameter, 4-7
Work files
optimum use of, 5-3
reassigning in a specification file, 3-15
with MERGE, 2-17

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or Country

— — 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

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698

— — Do Not Tear - Fold Here —

Cut Along Dotted Line