English

Fujitsu Software BS2000

SORT

User Guide

Valid for:
V8.0A

Edition November 2014

Comments… Suggestions… Corrections…

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to: bs2000.info@fujitsu.com

Certified documentation according to DIN EN ISO 9001:2015

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2015.

Copyright and Trademarks

Contents

1 Preface . . . 11

1.1 Objectives and target groups of this manual . . . 11

1.2 Summary of contents . . . 12

1.3 Changes since the last edition of the manual . . . 14

1.4 Notational conventions . . . 15

2 SORT functions and definitions . . . 17

2.1 Command and statement sequences in sort and merge runs . . . 20

2.2 Sort types . . . 24
2.2.1 Full sort . . . 25
2.2.2 Selection sort . . . 27
2.2.3 Tag sort . . . 29
2.2.4 Merge . . . 35

2.3 Control fields for sort/merge runs . . . 37
2.3.1 Sort fields . . . 38
2.3.2 Remainder fields . . . 54
2.3.3 Constant fields . . . 56
2.3.4 Sum fields . . . 58
2.3.5 Mask fields . . . 60
2.3.6 Match fields and match constants . . . 66
2.3.7 Symbolic names . . . 69
2.3.8 Overlap table for the different field types . . . 71

2.4 Record processing and modification in SORT . . . 73

2.5 SORT in an XS environment (31-bit addressing) . . . 80

2.6 Using extended character sets in SORT . . . 81
2.6.1 SORT-specific applications of extended character sets . . . 82
2.6.1.1 Sorting with extended codes . . . 82
2.6.1.2 CCSN entry in the SORT files . . . 83
2.6.1.3 Explicit CCSN entry for data records and output file . . . 83
2.6.1.4 Converting character constants into the code of the data records . . . 84
2.6.1.5 Conditions for mask fields . . . 84
2.6.1.6 Using extended character sets in user exits . . . 84

2.7 Use of Unicode character sets with SORT . . . 85
2.7.1 Normalization . . . 86
2.7.2 Characters with special processing . . . 87
2.7.3 Characters which are not supported . . . 88

3 Files of the sort/merge program SORT . . . 89

3.1 Input files . . . 93
3.1.1 Input files for sort runs . . . 93
3.1.2 Input files for merge runs . . . 97
3.1.3 PAM key elimination for input files . . . 98

3.2 Output file for sort/merge runs . . . 99
3.2.1 PAM key elimination for the output file . . . 103
3.2.2 POSIX output file . . . 104

3.3 Work files . . . 105
3.3.1 PAM key elimination for work files . . . 107

3.4 Auxiliary files . . . 108
3.4.1 PAM key elimination for auxiliary files . . . 111

3.5 Checkpoint file . . . 112
3.5.1 PAM key elimination for checkpoint files . . . 113

3.6 Object module library SORTMODS . . . 114

3.7 Statement files . . . 114

3.8 Close processing for SORT files . . . 115

3.9 Processing POSIX files with SORT . . . 117
3.9.1 POSIX in BS2000 . . . 117
3.9.2 Sorting POSIX files with SORT . . . 118

3.10 SORT and ACS . . . 119

3.11 Working with files larger than 32 GB . . . 120
3.11.1 Creating files larger than 32 GB . . . 120
3.11.2 Tag sort . . . 121
3.11.3 Recommendations when working with large files . . . 122

4 SORT statements . . . 125

4.1 Input sources . . . 125

4.2 SDF syntax representation . . . 126

4.3 Error handling . . . 126

4.4 Overview of the SORT statements . . . 127
ADD-SYMBOLIC-NAMES . . . 128
ASSIGN-EXITS . . . 132
ASSIGN-FILES . . . 139
ASSIGN-RESOURCES . . . 142
END . . . 144
MERGE-RECORDS . . . 145
MODIFY-CODE . . . 151
MODIFY-SORT-DEFAULTS . . . 152
SELECT-INPUT-RECORDS . . . 156
SET-RECORD-ATTRIBUTES . . . 160
SET-SORT-OPTIONS . . . 166
SHOW-SORT-DEFAULTS . . . 174
SORT-RECORDS . . . 175
SUM-RECORDS . . . 187

5 Calling SORT . . . 191

5.1 Calling SORT as a standalone program . . . 191
START-SORT . . . 192
SORT-FILE . . . 194
Command return codes . . . 202

5.2 Calling SORT as a subroutine . . . 206
5.2.1 Passing control information to SORT . . . 209
5.2.1.1 Level 0 . . . 210
5.2.1.2 Level 1 . . . 211
5.2.2 SORT macros . . . 212
5.2.2.1 SRT0: call to SORT at level 0 . . . 212
5.2.2.2 SRT1: call to SORT at level 1 . . . 215

5.3 SORT access method SORTZM . . . 220
5.3.1 Function of the SORT access method SORTZM . . . 220
5.3.2 Macros of the SORT access method SORTZM . . . 223
5.3.3 SRTOPEN: initiate sorting . . . 223
5.3.4 SRTPUT: pass record to SORT . . . 225
5.3.5 SRTGET: fetch record from SORT . . . 227
5.3.6 SRTCLSE: terminate sorting . . . 229
5.3.7 Example . . . 230

6 SORT user exits . . . 233

6.1 PLANNING: planning completed . . . 238
6.2 INPUT: input record processing . . . 239
6.3 OUTPUT: output record processing . . . 244
6.4 WORK-FILE-OVERFLOW . . . 249
6.5 EXLST-FOR-INPUT EXLST: exit for input files . . . 251
6.6 EXLST-FOR-OUTPUT: user exit for output files . . . 253
6.7 PHYSICAL-TRANSLATE: special character conversion table . . . 255
6.8 VIRTUAL-TRANSLATE: special character conversion table . . . 256
6.9 EXTERNAL-COMPARE: sequence defined by user routine . . . 258
6.10 TRANSLATE-CHARACTER: sequence defined by equating tables and coded character set . . . 260
6.11 INT: sort/merge run interrupt . . . 261

7 Checkpoint processing . . . 263

8 Optimization of sort runs . . . 265

8.1 Suitable CORE allocation . . . 265
8.2 Virtual merging . . . 267
8.3 Choice of sort method . . . 268
8.3.1 Code conversion . . . 268
8.3.2 Cycle sorting . . . 268
8.3.3 Multi-task sorting . . . 271

8.4 Suitable choice of file characteristics . . . 278
8.5 Record summation . . . 278
8.6 SORT as subsystem . . . 279
8.7 Use of the OPTIMIZATION operand in the SET-SORT-OPTIONS statement . . . 279
8.8 Modifying the preset default values for SORT . . . 279

9 Installation . . . 281

10 Examples . . . 287

10.1 Introduction . . . 287
10.1.1 Calling SORT . . . 288
10.1.2 Assigning the files . . . 288
10.1.3 Defining the sort criteria . . . 289
10.1.4 Terminating statement input and starting the sort run . . . 291
10.2 Example: Sorting a file with fixed-length records . . . 292
10.3 Example: Sorting a SAM file with variable record format . . . 295
10.4 Overview of the application examples . . . 298
10.4.1 SORT as main program . . . 298
10.4.2 Connection of user routines . . . 299
10.4.3 SORT as a subroutine . . . 299
10.4.4 Sorting according to Unicode . . . 299
10.5 Examples . . . 300
10.5.1 Example 1: Full sort of fixed format records . . . 301
10.5.2 Example 2: Full sort of variable format records . . . 303
10.5.3 Example 3: Full sort of an ISAM input file into a SAM output file . . . 305
10.5.4 Example 4: Full sort of ISAM files with variable record format . . . 308
10.5.5 Example 5: Full sort of multiple files with variable record format . . . 310
10.5.6 Example 6: Full sort (input file = output file) . . . 313
10.5.7 Example 7: Full sort EBCDIC to DIN standard for text ordering . . . 315
10.5.8 Example 8: Full sort using FORMAT=*MODIFY-CODE . . . 318
10.5.9 Example 9: Full sort using FORMAT=*EXTENDED-CHARACTER and FORMAT=*TRANSLATE-CHARACTER . . . 320
10.5.10 Example 10: Full sort with summation and SELECT-INPUT-RECORDS . . . 325
10.5.11 Example 11: Selection sort of variable format records . . . 329
10.5.12 Example 12: Selection sort (binary) of fixed-format records . . . 332
10.5.13 Example 13: Selection sort of a POSIX file . . . 334

10.5.14 Example 14: Tag sort of fixed-format records . . . 337
10.5.15 Example 15: Merging files . . . 339
10.5.16 Example 16: INPUT user exit . . . 341
10.5.17 Example 17: OUTPUT user exit . . . 345
10.5.18 Example 18: PHYSICAL-TRANSLATE user exit . . . 349
10.5.19 Example 19: VIRTUAL-TRANSLATE user exit . . . 353
10.5.20 Example 20: SORT as a subroutine (level 0) . . . 357
10.5.21 Example 21: SORT as a subroutine (level 1) . . . 361
10.5.22 Example 22: SORT access method . . . 366
10.5.23 Example 23: SORT access method (multiple sort) . . . 371
10.5.24 Example 24: Full sort according to data in Unicode . . . 380

10.6 Contents of the example files . . . 383
10.6.1 Preliminary remark . . . 383
10.6.2 Contents of the file RESTAURANT . . . 384
10.6.3 Contents of the file LITERATURE . . . 384
10.6.4 Contents of the file MUSEUM . . . 385
10.6.5 Contents of the file CULTURE.1 . . . 386
10.6.6 Contents of the file CULTURE.2 . . . 386
10.6.7 Contents of the file CULTURE.3 . . . 387

11 Messages of the sort/merge program . . . 389

11.1 Message output to SYSOUT . . . 389

11.2 Message output in S variables . . . 391

11.3 SORT/MERGE messages . . . 396

12 Appendix . . . 397

12.1 SORT error handling . . . 397
12.1.1 Handling internal SORT errors . . . 397
12.1.2 Error information when SORT is called as a standalone program . . . 398
12.1.3 Error information when SORT is called as a subroutine . . . 400
12.1.4 Structure of the RCF area . . . 401

12.2 Structure of SORT control tables . . . 402
12.2.1 Table overview . . . 402
12.2.2 Input block SVB . . . 404
12.2.3 Transfer control area . . . 406
12.2.4 SORT statement tables . . . 408

12.3 Sort table UTF-16 . . . 409

Related publications 435

Index 437

1 Preface

This manual describes how to use the software product SORT in BS2000 with the SDF command format.

SORT V8.0A is executable in BS2000/OSD-BC® V8.0 and higher.

SORT V8.0 can be called via a SRT80 starter module, versions SORT V7.3 and higher. However, there is no guarantee that older SORT versions with a SRT80 starter module, version SORT V8.0, will function.

With SORT you can sort and merge data records according to specific criteria. You can also modify the contents and format of data records. SORT is equipped with user exits for connecting user routines and may be called as a subroutine. SORT also provides the access method SORTZM.

1.1 Objectives and target groups of this manual

This manual is designed for BS2000 users.

The user should have experience with BS2000, and in particular be conversant with the most important BS2000 commands. Useful reference documentation includes the BS2000 manuals “Commands” [1], “Introductory Guide to DMS” [2] and “DMS Macros” [3].

If you wish to invoke SORT as a subroutine and initiate actions via the user exits, you should be familiar with the BS2000 Assembler and BS2000 system facilities. Useful sources of reference in this case are the manuals “Assembler Instructions (BS2000)” [4], “ASSEMBH” [5] and “Executive Macros” [6].

The SORT Ready Reference [12] is available as a supplement to this manual. This contains all the SORT statements and macros and their syntax, and is intended as a reference work for users familiar with SORT.

1.2 Summary of contents

Chapter 2 provides a brief overview of SORT, explains the various sort types and control fields, the record processing and modification possible, the switch to the XS environment, the use of extended character sets, and the special features of Unicode support in SORT.

Chapter 3 provides information on the various files with which SORT works.

Chapter 4 describes the input sources for SORT statements, as well as the syntax of the statements in SDF format.

Chapter 5 describes the various ways in which SORT may be called and how control information is passed to SORT.

Chapter 6 provides an overview of the various user exits and the associated measures which the user can initiate.

Chapter 7 describes the processing of checkpoints for logging and checking the sort run.

Chapter 8 provides information on how to optimize sort runs.

Chapter 9 lists all the important information which must be borne in mind when installing SORT.

Chapter 10 is especially well suited as an introduction fort the first-time user. In addition to a number of application examples, this chapter also contains an introductory section. It chiefly addresses first-time users of SORT and is intended to make the definition and execution of SORT more easily understandable.

Chapter 11 lists all messages which can occur in a SORT/MERGE run.

The Appendix describes SORT error handling, explains the structure of the SORT control tables, and contains the sort table UTF-16.
It is followed by the “Related publications” and “Index” chapters.

Readme file

The functional changes to the current product version and revisions to this manual are described in the product-specific Readme file.

Readme files are available to you online in addition to the product manuals under the various products at *http://manuals.ts.fujitsu.com*. You will also find the Readme files on the Softbook DVD.

Information under BS2000

When a Readme file exists for a product version, you will find the following file on the BS2000 system:

`SYSRME.<product>.<version>.<lang>`

This file contains brief information on the Readme file in English or German (<lang>=E/D). You can view this information on screen using the `/SHOW-FILE` command or an editor. The `/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>` command shows the user ID under which the product's files are stored.

Additional product information

Current information, version and hardware dependencies, and instructions for installing and using a product version are contained in the associated Release Notice. These Release Notices are available online at *http://manuals.ts.fujitsu.com*.

1.3 Changes since the last edition of the manual

This new version is SORT V8.0A, and the following functional extensions and changes have been made since the last edition for SORT V7.9:

Chapter / Section	Extensions / New features
All chapters	SORT V8.0A is executable in BS2000/OSD-BC V8.0 and higher.
2.6.1.2	Depending on the type of file, SORT handles the CCSN entry differently.
4.4	SET-SORT-OPTIONS statement: New operand IGNORE-CHARACTER Specifies the characters that SORT is to ignore.
4.4	ASSIGN-EXITS statement: Extension of the operand WORK-FILE-OVERFLOW= *MODULE(...) by INTERFACE-VERSION Indicates a record count in 8 bytes or in 4 bytes.
4.4	SORT-RECORDS statement: Extension of the operands ESTIMATED-RECORDS, FROM-RECORD and NUMBER-OF-RECORDS by <alphanum-name 1..19> Extends the specification of the number of records beyond 2.147.483.639.
5.2.2	Macro calls via SRT0 and SRT1: MULTI=NOIMON: The SYSPAR parameter file is read at the start.
6.4	WORK-FILE-OVERFLOW user exit: Extension of MODULE by INTERFACE-VERSION=1/2

1.4 Notational conventions

The following notational conventions are used in this manual:

- References in the text to other publications are given in an abbreviated form. The full titles of all publications referred to can be found under “Related publications” at the back of the manual. This includes instructions for ordering these publications.
- In the examples, user input appears in `bold Courier typeface` and system output in `ordinary Courier typeface`.
- The following pictogram is used to highlight important information:

  to indicate information of particular importance
- The syntax of the statements is in SDF command format.

2 SORT functions and definitions

SORT is a program which allows data records from up to 99 input files to be sorted or merged into a single output file.

SORT is executable in interactive or batch mode in BS2000 and supports the sort/merge processing of SAM, ISAM, BTAM, PAM and POSIX files.

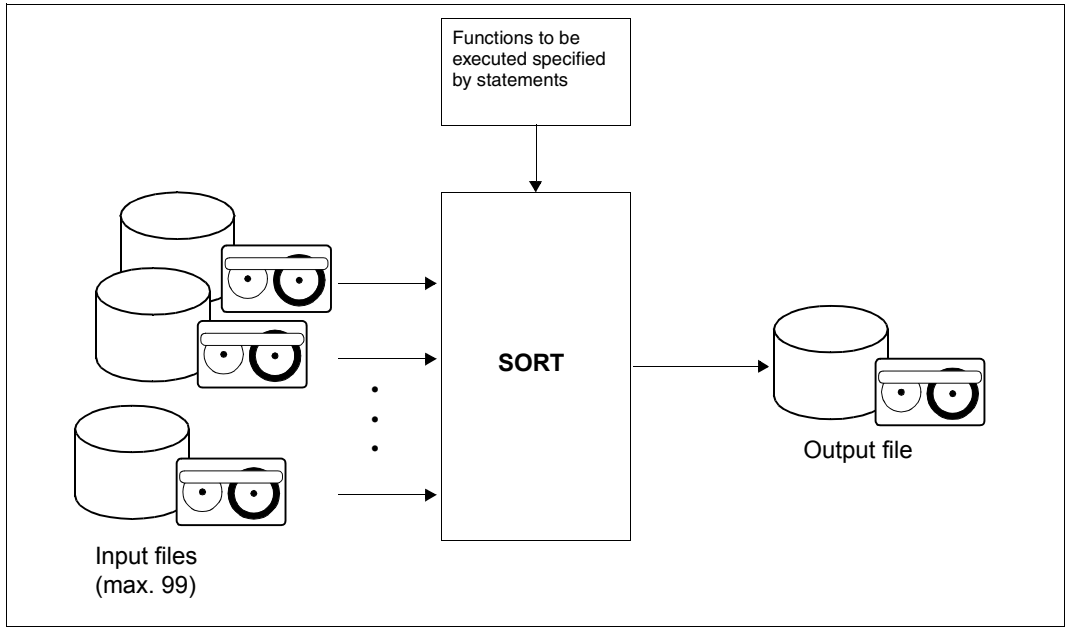


Figure 1: Sorting and merging with SORT

Sorting means arranging data records into a certain order according to criteria specified by the user.

Merging means combining two or more ordered input files into a single output file. The input files must first have been sorted according to the criteria applicable to the merge run.

The order of the output records for sorting and merging is determined by one or more fields of the input record, as defined by the user (see section “Sort fields” on page 38). These fields can be sorted in

- ascending order,
- descending order, or
- via a user routine (see chapter “SORT user exits” on page 233).

In addition to the sort and merge functions, SORT offers the following facilities:

- Specification of selection criteria for the records to be sorted:
  - logical selection of records for sorting
  - selection of a sequence of records as an extract from the specified input files
  - compaction of records with identical sort keys, with or without summation of sum fields in the resulting record
  - retention of input sequence for records with identical sort keys.
- Modification of format and content of records and record areas:
  - modification of record format
  - modification of record length
  - conversion of input data
  - record creation through record selection sort.
- Insertion, deletion or modification of records or parts of records:
  - editing of output records via print masks
  - elimination of sort fields
  - insertion of freely selectable constants.
- User control of logging and runtime parameters:
  - user control of logging and program execution
  - setting of checkpoints
  - interruption of the SORT run followed by interactive dialog with SORT at the display terminal.

- Interface to subroutines for additional functions:
  - user exits for interfacing to user routines.

These functions, as well as calling SORT as a subroutine and the sort access method SORTZM, are described and explained in the following chapters.

Internal processing sequence for sort runs

A sort run can be subdivided into the following distinct phases:

1. **Preparatory phase**

   In the preparatory phase, SORT decodes all the statements.

2. **Planning phase**

   In the planning phase, SORT determines the sorting strategy and requests the requisite resources (e.g. memory space).

3. **Input and presorting phase**

   In this phase, SORT reads the records to be sorted and generates ordered sequences of blocks via a presort run.

4. **DOMINO phase**

   In this phase SORT attempts to achieve an optimal concatenation of the presorted blocks into block sequences. This is intended to minimize data movements between the internal main memory and the work file and shorten internal merging operations in the subsequent internal merge phase.

5. **Internal merge phase**

   In the internal merge phase the block sequences are ordered in one or more internal merge passes into a number suitable for the end merge. In cycle and multi-task sorting (see chapter “Optimization of sort runs” on page 265), these block sequences are buffered in separate auxiliary files.

6. **End merge and output phase**

   In this phase SORT merges the presorted sequences (some of which may be stored in auxiliary files) into a single block sequence and outputs this to the output file.

The input and presorting phase, and the internal merge phase, can be repeated several times in succession in a cycle or multi-task sort.

2.1 Command and statement sequences in sort and merge runs

Invoking SORT as a standalone program requires the following definition steps:

– Assignment of SORT files via ADD-FILE-LINK commands, unless the file assignment is made during the SORT run via the ASSIGN-FILES statement (which can be used in SORT only).

– Invocation of SORT.

– Assignment of SORT files using the ASSIGN-FILES command, unless the file assignment has already been made via the ADD-FILE-LINK command. Input/output files must be assigned for each run, unless the user exits INPUT and OUTPUT have been specified for this. For more information on this refer to the description of the ASSIGN-FILES and ASSIGN-EXITS statements and see also chapter “Files of the sort/merge program SORT” on page 89.

– Statements to SORT.
A SORT-RECORDS or MERGE-RECORDS statement is a required specification for each run. Other optional statements may follow. Statement files can be used for statement input. See the description of the ASSIGN-FILES statement and see also chapter “Files of the sort/merge program SORT” on page 89 for further details.

– Specification of an END statement to complete statement input.

General overview of the command and statement sequence in sort/merge runs

The statements and commands must be entered in the order defined by the numbered sections. Within these sections the statements can be specified in any order.

1. File assignment with ADD-FILE-LINK commands
   (can be omitted if the files have been assigned with the ASSIGN-FILES statement, see point 3 of this overview)

   – Input file(s)

```
/ADD-FILE-LINK LINK-NAME=SORTIN[,...], -          Sort using one input file
/              FILE-NAME=sortfilename
                        or
/ADD-FILE-LINK LINK-NAME=SORTIN01[,...], -        Sort using more than one
/              FILE-NAME=sortfilename1            input file
/ADD-FILE-LINK LINK-NAME=SORTIN02[,...], -
/              FILE-NAME=sortfilename2
/ ...
/ADD-FILE-LINK LINK-NAME=SORTIN99[,...], -
/              FILE-NAME=sortfilename99
                        or
/ADD-FILE-LINK LINK-NAME=MERGE01[,...], -         Merge run
/              FILE-NAME=mergefilename1
/ADD-FILE-LINK LINK-NAME=MERGE02[,...], -
/              FILE-NAME=mergefilename2
/ ...
/ADD-FILE-LINK LINK-NAME=MERGE99[,...], -
/              FILE-NAME=mergefilename99
```

   – Output file

```
/CREATE-FILE FILE-NAME=outputfilename
/ADD-FILE-LINK LINK-NAME=SORTOUT[,...],FILE-NAME=outputfilename
```

– Work files (optional)

Assignment of **one** work file

```
/CREATE-FILE FILE-NAME=workfilename
/SET-FILE-LINK LINK-NAME=SORTWK[,...], -
/              FILE-NAME=workfilename
```

or

Assignment of **more than one** work file

```
/CREATE-FILE FILE-NAME=workfilename
/ADD-FILE-LINK LINK-NAME=SORTWK1[,...], -
/              FILE-NAME=workfilename1
/CREATE-FILE FILE-NAME=workfilename1
/ADD-FILE-LINK LINK-NAME=SORTWK2[,...], -
/              FILE-NAME=workfilename2
/ ...
/CREATE-FILE FILE-NAME=workfilename9
/ADD-FILE-LINK LINK-NAME=SORTWK9[,...], -
/              FILE-NAME=workfilename9
```

– Auxiliary files (as required)

```
/CREATE-FILE FILE-NAME=auxfilename1
/ADD-FILE-LINK LINK-NAME=SORTWKO1[,...],FILE-NAME=auxfilename1
/CREATE-FILE FILE-NAME=auxfilename2
/ADD-FILE-LINK LINK-NAME=SORTWKO2[,...],FILE-NAME=auxfilename2
/ ...
/CREATE-FILE FILE-NAME=auxfilename99
/ADD-FILE-LINK LINK-NAME=SORTWK99[,...],FILE-NAME=auxfilename99
```

– Module library (optional)

```
/ADD-FILE-LINK LINK-NAME=SORTMODS,FILE-NAME=modulelibraryname
```

– Checkpoint file (optional)

```
/CREATE-FILE FILE-NAME=checkpointfilename
/ADD-FILE-LINK LINK-NAME=SORTCKPT,FILE-NAME=checkpointfilename
```

2. Calling SORT

```
/START-SORT
```

3. Control statements to SORT

   – File assignment using the ASSIGN-FILES statement
   (only necessary for files which have not already been assigned with SET-FILE-LINK commands, see point 1 of this overview)

```
//ASSIGN-FILES -
//    INPUT-FILES=inputfilename(s), -          Input file(s)
                                               (max. 99)
//    OUTPUT-FILES=outputfilename, -           Output file
//    WORK-FILES=workfilename(s), -            Work file(s)
                                               (max. 9, optional)
//    AUXILIARY-FILES=auxfilename(s), -        Auxiliary file(s)
                                               (max. 99, optional)
//    CHECKPOINT-FILE=checkpointfilename, -    Checkpoint file (optional)
//    MODULE-LIBRARY=modulelibraryname, -      Module library (optional)
//    STATEMENT-FILES=statementfilename(s)     Statement file(s)
                                               (max. 10, optional)
```

   – Specification of the sort criteria and definition of the structure of the output file

```
//SORT-RECORDS sort-criteria                   Sort
```

   or

```
//MERGE-RECORDS sort-criteria                  Merge
```

   – Further (optional) statements

```
//SET-RECORD-ATTRIBUTES ...
//SET-SORT-OPTIONS ...
//SELECT-INPUT-RECORDS ...
```

4. End of the input and start of the sort/merge run

```
//END
```

2.2 Sort types

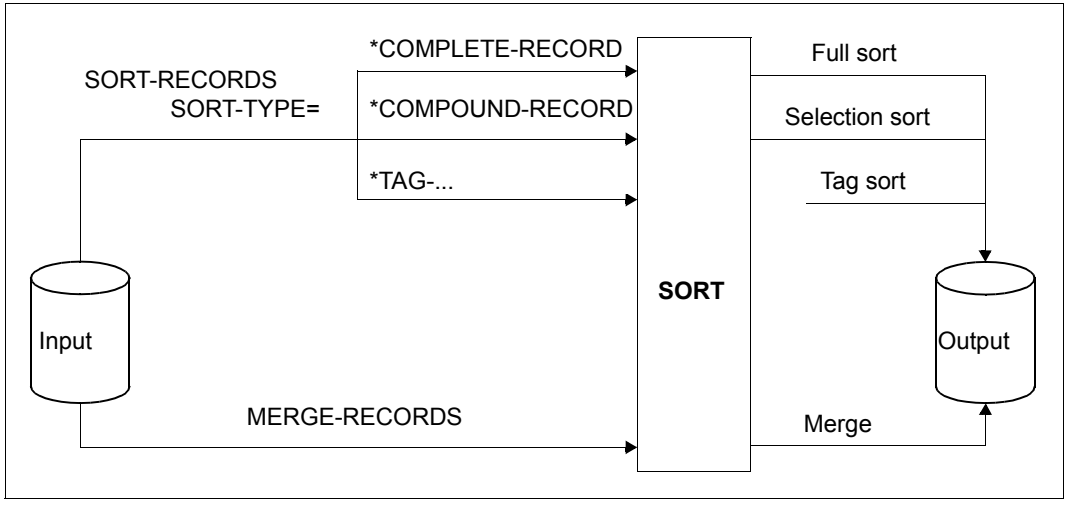


Figure 2: Sort functions of the SORT sort/merge program

SORT supports the following types of sort:

– **Full sort**
The complete input record is written to the output file according to the specified sort criteria.

– **Selection sort**
Selected parts of the input record are combined into a new output record. The output records are written to an output file according to the specified sort criteria.

– **Tag sort**
The records are sorted according to the sort criteria; SORT then tags each output record with an address that specifies the position of the original record (retrieval address) in the input file.

– **Merge**
Up to 99 input files with the same record format (and length in the case of fixed-length records), after having been sorted according to identical criteria, can be merged into a single output file on the basis of the sort criteria.

2.2.1 Full sort

In a full sort, the entire input record is processed by SORT and sorted in ascending or descending order according to the sort fields (see section “Sort fields” on page 38). Records of fixed or variable length can be processed.

Under certain defined conditions, the output records may have a different record length and/or format than the input records (see section “Record processing and modification in SORT” on page 73). This type of sort is selected by specifying SORT-TYPE=*COMPLETE-RECORD in the SORT-RECORDS statement. As SORT-TYPE=*COMPLETE-RECORD is the default setting, it does not have to be specified explicitly.

Because the complete input record is processed in a full sort, no print masks are permitted.

Example

The input file ADDRESSES contains the following variable-length records:

rl	MILLER	ANDREW	LONDON	PETER HOOGANFORD DRIVE 5
rl	BECKETT	SAM	LIVERPOOL	SEASIDE 37
rl	SOLOW	BRIGITTE	LEEDS	DEARSBY STREET 49
rl	BECKETT	FRANK	MANCHESTER	LOREAN STREET 2

The input records are to be sorted by SORT by last name and first name and then written unchanged to the output file ADDRESSES.SORT.

```
/start-sort ------------------------------------------------------------------ (1)
//assign-files input-files=addresses,output-file=addresses.sort ------------- (2)
//sort-records fields=(*field-explicit(position=5,length=9), - ---------------- (3)
//                     *field-explicit(position=14,length=10))
//set-record-attributes input=*variable(maximum-record-size=64) ------------- (4)
//end -------------------------------------------------------------------------- (5)
```

(1) SORT is invoked.

(2) The input and output files are assigned.

(3) The sort fields “first name” and “last name” are defined; sort type “full sort” and sorting order “ascending” are preset by default.

(4) The maximum length of the input records is defined.

(5) End of statement input and start of the sort run.

Output file ADDRESSES.SORT:

rl	BECKETT	FRANK	MANCHESTER	LOREAN STREET 2
rl	BECKETT	SAM	LIVERPOOL	SEASIDE 37
rl	MILLER	ANDREW	LONDON	PETER HOOGANFORD DRIVE 5
rl	SOLOW	BRIGITTE	LEEDS	DEARSBY STREET 49

2.2.2 Selection sort

In a selection sort, the record to be sorted is composed of those parts of the input record specified in the FIELDS operand of the SORT-RECORDS statement as sort, remainder, and constant fields. In the output record, the order of these parts corresponds to the order in which these fields were defined.

SORT generates fixed-length output records by default in a selection sort, regardless of the record format of the input records. The sort type “selection sort” must be specified by means of the operand SORT-TYPE=*COMPOUND-RECORD in the SORT-RECORDS statement.

Special points to note concerning variable-length input records:

– For output of fixed-length records, the specified sort and remainder fields must be in the area common to all the records in the input file (minimum record length).

– To generate variable-length selection records, OUTPUT=*VARIABLE must be specified in the SET-RECORD-ATTRIBUTES statement. The following applies to the last field specified:
If a remainder field, it may be located wholly or partly within the “variable” part of the input record, i.e. that notional area of the record that lies outside the minimum record length and extends to the maximum record length.

  If the field is a sort field having CHARACTER, EBCDIC-DIN or EBCDIC-INTERNATIONAL format, then at least the first character must be located in the fixed part of the input record. SORT automatically calculates the new record length and prefixes this as a record length field to the new selection record. No remainder field is required for this record length field.

Under certain conditions, or upon request, the output records may have a different record length and record format than the original selection record, or be made longer or shorter than the input record (see section “Record processing and modification in SORT” on page 73).

In a selection sort it is possible to edit sort, remainder and sum fields in BINARY, FIXED-POINT, PACKED-DECIMAL and ZONED-DECIMAL format ready for printing. This requires the use of a print mask which corresponds to the edit mask of the Assembler instruction ED (see section “Mask fields” on page 60).

Example

An output file ADDRESSES.SORT is to be created from the variable-length records of the ADRESSES input file. This file should only contain the areas bounded by bytes 5 - 13 (last name) and bytes 24 - 38 (place of residence) of the input records, and should be sorted according to these fields.

Input file ADDRESSES:

rl	MILLER	ANDREW	LONDON	PETER HOOGANFORD DRIVE 5
rl	BECKETT	SAM	LIVERPOOL	SEASIDE 37
rl	SOLOW	BRIGITTE	LEEDS	DEARSBY STREET 49
rl	BECKETT	FRANK	MANCHESTER	LOREAN STREET 2

```
/start-sort ──────────────────────────────────────────────────────────── (1)
//assign-files input-files=addresses, - ──────────────────────────────── (2)
//             output-file=addresses.sort
//sort-records fields=(*field-explicit(position=5,length=9), - ──────── (3)
//                     *field-explicit(position=24,length=15)), -
//             sort-type=*compound-record
//set-record-attributes input=*variable(maximum-record-size=64) ──────── (4)
//end ──────────────────────────────────────────────────────────────────── (5)
```

(1) SORT is invoked.

(2) The input and output files are assigned.

(3) The sort fields are defined and the sort type “selection sort” is specified.

(4) The record type and maximum record length are defined.

(5) End of statement input.

Output file ADDRESSES.SORT:

BECKETT	LIVERPOOL
BECKETT	MANCHESTER
MILLER	LONDON
SOLOW	LEEDS

2.2.3 Tag sort

In a tag sort, as in a selection sort, selection records are constructed from the individual sort fields. In addition, SORT tags each record of the output file with the record address of the associated record from the input file (see the SORT-TYPE operand of the SORT-RECORDS statement). As the address and sort key always have a fixed length in a selection sort, SORT also generates fixed-length output records in a tag sort, regardless of the record format of the input file, its DMS attributes, or the specifications made via the SET-FILE-LINK command. These default settings of the SORT program can be changed by means of the SET-RECORD-ATTRIBUTES statement.

For special points to note concerning variable-length records, refer to selection sorting above (see section “Selection sort” on page 27).

The sort type “tag sort” must be specified in the SORT-RECORDS statement by means of the SORT-TYPE operand, as follows:

– SORT-TYPE = *TAG-TRAILER (appended address)
– SORT-TYPE = *TAG-HEADER (prefixed address)
– SORT-TYPE = *TAG-COMPOUND (prefixed and extended address).

Conditions for tag sorting

– The input file must be a disk file.
– The input file must be a SAM, ISAM or PAM file.
– The input file must not be a POSIX file.
– Multiple input files of the same type are permitted, but since the retrieval address does not include a file identifier, the user must make sure that the position (and thus the address) of the input records can be distinguished.
– Output and input file must be different.
– Use of print masks is not permitted.
– The input file must not be a POSIX file, since it cannot be guaranteed that the retrieval address does not contain characters which POSIX might interpret as record separators (see section “Sorting POSIX files with SORT” on page 118).

Advantages of tag sorting

Tag sorting provides the following advantages if several files are to be created from one input file and sorted according to different criteria, or if the (unsorted) input file is to be retained:

– Storage space for output files can be saved, since these only contain the tag assigned to the record of the input file and possibly the sort key (which can be suppressed by means of the ELIMINATE operand).

– Changes to the data need only be made in one file, the input file (SORTIN file).
If such changes only affect parts of the record which do not constitute a sort key, no further action is necessary because every record in the input file can be directly accessed via the retrieval address in the address field of the records in the (sorted) output files.
Similarly, if a sort key is modified, only the sort run whose sort key is affected must be repeated.

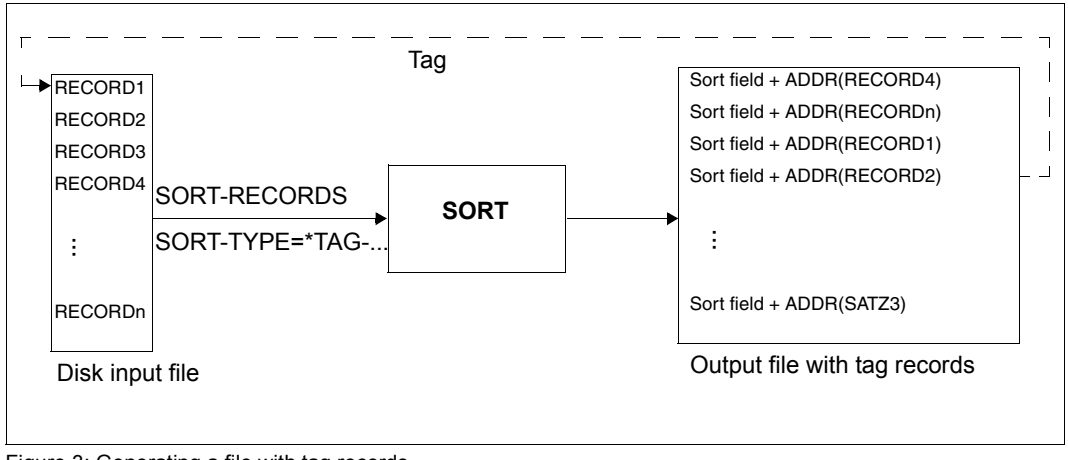


Figure 3: Generating a file with tag records

To tag-sort a file (on disk) according to a sort key, the user must first generate a file with tag records in a sort run using SORT-TYPE=*TAG-COMPOUND SORT-TYPE=*TAG-HEADER, or SORT-TYPE=*TAG-TRAILER. These tag records consist of sort and remainder fields and an address or “tag” field (prefixed to the record in the case of *TAG-COMPOUND or *TAG-HEADER, appended in the case of *TAG-TRAILER). It is, however, possible to eliminate sort fields using the ELIMINATE operand.

Address field

The format of the address field with which the tag record begins (with *TAG-COMPOUND or *TAG-HEADER) or ends (with *TAG-TRAILER) is dependent on the access method of the input file, as follows:

- **SAM**

  Depending on the sort type, the retrieval address is either four or six bytes long:

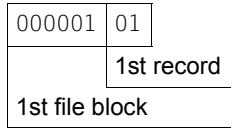
  – With *TAG-HEADER and *TAG-TRAILER a four-byte address with the format bbbbbbrr is created.

  bbbbbb Number of the accompanying file block in which the record is located.

  rr Relative record number in the file block ($1 \leq rr \leq 255$, i.e. the file block must not contain more than 255 records).

  Example

  Address field for the first record of a file (in hexadecimal notation):

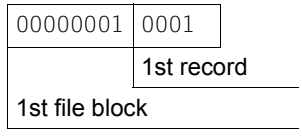
  – With *TAG-COMPOUND a six-byte address with the format bbbbbbbbrrrr is created.

  bbbbbbbb Number of the accompanying file block in which the record is located.

  rrrr Relative record number in the file block ($1 \leq rrrr \leq 65535$).

  Example

  Address field for the first record of a file (in hexadecimal notation):

- **ISAM**

  The address field contains the ISAM key of the associated input record. The length corresponds to the file attribute KEY-LENGTH in the input file and can be between 1 and 255 bytes long.

- **PAM**

  The address field comprises bytes 4 - 7 of the PAM key in the format vppp.

  v Version number of the file (byte 4)

  ppp Logical number of the PAM block (bytes 5 - 7)

  Example

  Address field (in hexadecimal notation):

01	000003
	3rd PAM block
1st version	

Example

"Personnel" is a SAM file consisting of records of fixed length (59 bytes) and having the following format and contents:

Name	First name	Address	Dept.	Pers. no.
Miller	Andrew	Poplar Avenue 47	KT25	544507
Allan	Hillary	High Street 101	AY4	345670
Smith	Albert	Gardener Street 14	PX453	047913
Majors	Christine	Railway Cuttings 12	PX23	987650
Smythe	Brenda	Thomas Square 1	BT34	965471
Kennedy	George	Edgeware Road 62	NY211	873250
Stevens	Henry	Market Square 13	NY12	987234
Baker	Fred	Scott Street 34	KT23	765921
Johnston	Annette	Richmond Street 98	BT342	345678
Mellors	Ingrid	Salford Drive 4	TI34	456372
Brown	Tony	Skyview Terrace 9	UB81	786534
Charles	Ernest	Millhouse Street 23	TI32	537892
Walthers	Claudia	Millford Crescent 31	ZY21	342108
Richards	Bernard	Illsley Square 3	UB12	518376
Drever	James	Rose Drive 31	PX3	875211

This file is to be tag-sorted according to (last) name, first name, address, department and personnel no. This requires a total of 5 sort runs using SORT-TYPE=*TAG-TRAILER or SORT-TYPE=*TAG-COMPOUND/*TAG-HEADER. The sort run for the record area "Name" is described here. Similar runs are performed for the other record areas.

```
/start-sort
//assign-files input-files=personnel, -
//             output-file=name
//sort-records fields=(*field-explicit(position=1,length=13)), -
//             sort-type=*tag-trailer
//end
```

On completion of this sort run, the output file “Name” contains the following records, each 17 bytes long (13 data bytes and 4 bytes for the retrieval address):

Name	Address field*	
	Blk. no.	A.
A l l a n	000001	02
B a k e r	000001	08
B r o w n	000001	0B
C h a r l e s	000001	0C
D r e v e r	000001	0F
J o h n s t o n	000001	09
K e n n e d y	000001	06
M a j o r s	000001	04
M e l l o r s	000001	0A
M i l l e r	000001	01
R i c h a r d s	000001	0E
S m i t h	000001	03
S m y t h e	000001	05
S t e v e n s	000001	07
W a l t e r s	000001	0D

* The contents of the address field are shown in hexadecimal form to make them easier to read

Blk no.: File block number

A.: Logical record address in block,
e.g. 0D is the 13th record of the input file “Personnel”

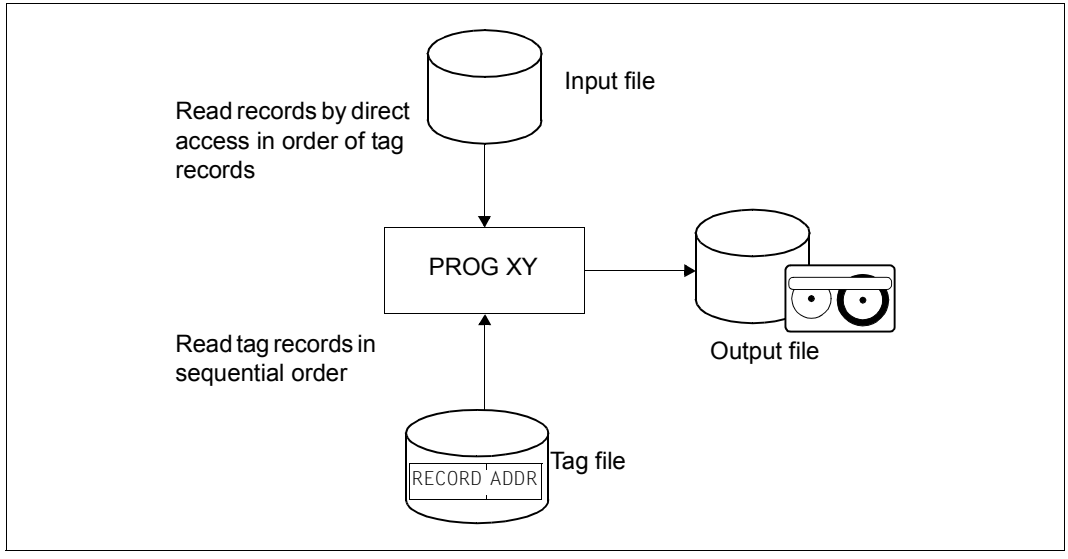


Figure 4: Processing a file using tag records

A user program can make use of the address field (retrieval address, see the “DMS Macros” manual [3]) of the records in the “Name” file to access the associated (complete) record in the “Personnel” file. For example, it is possible to tell from the “Name” file that the record containing the data field “Kennedy” has the logical record number 06 and the block number 1. Thus, the address field can be used to retrieve the complete record “Kennedy George Edgeware Road 62 NY211 873250” from the personnel file.

Suppose, for example, that a change of address needs to be registered for an employee in the personnel file: only a new sort run to generate a new “Address” file is required. All the other files (“Name”, “First name”, “Department”, etc.) remain unchanged.

Note

By eliminating the sort fields it is possible to reduce the output records produced during tag sorting to the retrieval address and thus save on storage space (see also the ELIMINATE operand of the SORT-RECORDS statement).

2.2.4 Merge

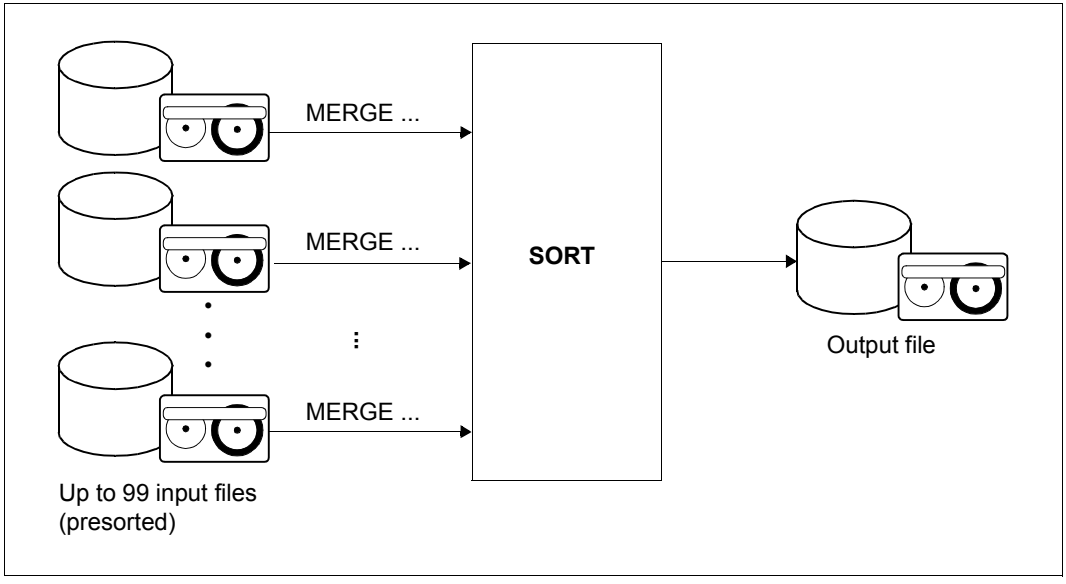


Figure 5: Merging files

In a merge run, up to 99 input files having identical record format (and identical record length in the case of fixed-length record format) can be merged into a single output file according to specified sort criteria. All the input files must already have been sorted according to these same sort criteria before the merge run.

Example

Two input files, sorted in ascending order by name, are to be merged into an output file called “PERSONNEL.TOTAL”:

Input files:

File “PERSONNEL1”

Name	First name
Allan	Hilary
Brown	Tony
Drever	James
Kennedy	George

File “PERSONNEL2”

Name	First name
Baker	Fred
Charles	Ernest
Johnston	Annette
Majors	Christine

```
/start-sort ──────────────────────────────────────────────────────────── (1)
//assign-files input-files=(personnel1,personnel2), - ──────────────── (2)
//             output-file=personnel.total
//merge-records fields=*field-explicit(position=1,length=9) ─────────── (3)
//end ──────────────────────────────────────────────────────────────────── (4)
```

(1) SORT is invoked

(2) The input and output files are assigned

(3) Merge statement

(4) End of statement input

Output file PERSONNEL.TOTAL:

Name	First name
Allan	Hilary
Baker	Fred
Brown	Tony
Charles	Ernest
Drever	James
Johnston	Annette
Kennedy	George
Majors	Christine

2.3 Control fields for sort/merge runs

Control fields are areas of the records to be sorted that are evaluated by SORT when determining the sequence in a given sort/merge run.

Control fields must be specified in the statements for the sort/merge run; they may be:

– sort fields
– remainder fields
– constant fields
– sum fields
– mask fields
– match fields
– match constants

Symbolic names can be defined for all control fields by means of the statement ADD-SYMBOLIC-NAMES. They can then be used in any subsequent statements.

SORT generates different processing routines depending on the control field specifications. No single routine may occupy more than 4096 bytes of main memory space. Making these specifications too complex may cause problems with internal addressing (error message SRT1250). In such a case the statements should be simplified and the sort/merge run repeated.

2.3.1 Sort fields

Sort fields are byte sequences in a record that are evaluated by SORT in order to determine the order of the output records. One or more All the sort fields with a defined sequence (see the PRIORITY operand of the SORT-RECORDS and MERGE-RECORDS statements) are combined into a sort key. The sort field value, which is the determinant factor for the sort comparisons, and the permissible length are format-dependent.

The following formats are permitted in SORT:

BINARY	Binary	1 bit - 256 bytes
CHARACTER	Character	1 - max. record length
FIXED-POINT	Fixed-point	1 - 256
FLOATING-POINT	Floating-point	1 - 256
PACKED-DECIMAL	Packed decimal	1 - 16
ZONED-DECIMAL	Zoned decimal	1 - 16
EBCDIC-DIN	EBCDIC to DIN standard	1 - 256
EBCDIC-INTERNATIONAL	EBCDIC to international standard	1 - 256
VIRTUAL-TRANSLATE	Order specified by conversion table (ASSIGN-EXITS)	1 - 256
PHYSICAL-TRANSLATE	Order specified by conversion table (ASSIGN-EXITS)	1 - 256
MODIFY-CODE	Order specified by MODIFY-CODE statement	1 - 256
EBCDIC-ISO-EBCDIC	Input in EBCDIC code, sort in ISO code, output in EBCDIC code	1 - 256
EXTENDED-CHARACTER	Order specified by coded character set	1 - 256
TRANSLATE-CHARACTER	Order specified by equating tables and coded character set (ASSIGN-EXITS/load module)	1 - 256
UNICODE-CHARACTER	Order specified by the Unicode Default Collation Table supplied by XHCS	2 - 256

Sort fields can be specified in the SORT-RECORDS and MERGE-RECORDS statements. How to define sort fields is explained in the descriptions of these statements.

The mandatory and optional specifications for sort fields are summarized in the following table:

POSITION	mandatory	Starting point of sort field
LENGTH	mandatory	Length of sort field
FORMAT	optional	Format of sort field
PRIORITY	optional	Sequence number of sort field
SORTING-ORDER	optional	Sorting sequence
ELIMINATE	optional	Sort field to be eliminated
PRINT-MASK	optional	Print mask to be used to edit field

Example

Format of the input records:

Name	First name	Address	Telephone

Variant 1

The sort key contains only one sort field, the “Name” field, which begins at position 1 and is 13 bytes long. Both these numbers must be specified in the sort statement.

```
/start-sort
//sort-records fields=*field-explicit(position=1,length=13)
//end
```

To sort according to the 10-byte field “Telephone” as of position 48, the following statements are necessary:

```
/start-sort
//sort-records fields=*field-explicit(position=48,length=10)
//end
```

Variant 2

Records are to be sorted according to the “Name” field and (for records with identical contents in the “Name” field) the “Address” field. The sort key therefore contains more than one sort field.

```
/start-sort
//sort-records fields=(*field-explicit(position=1,length=13), -
//                     *field-explicit(position=26,length=22))
//end
```

Sort fields for variable-length records

With variable-length records, sort fields in all formats except CHARACTER, EBCDIC-DIN and EBCDIC-INTERNATIONAL may only be located in an area of the record which does not belong to the “variable part”, i.e. that notional area of the record that lies outside the minimum record length and extends to the maximum record length.

This means that (except for the aforementioned formats) a sort field must not be longer than the shortest record to be processed. Remember also that it is subject to the permitted (format-dependent) lengths.

With the CHARACTER, EBCDIC-DIN and EBCDIC-INTERNATIONAL formats, the sort fields may also extend into the “variable part” of the record.

In sort comparisons, the notional (non-existent) record area is treated as if it were padded with the filler character (see the FILLER operand of the SET-RECORD-ATTRIBUTES statement). If no filler character is specified, X’00’ is assumed.

In a selection sort, a sort field of this type must also be the last selection field. Then only the existing part of the field is transferred and the record length recalculated (see section “Remainder fields” on page 54).

Example

SORT is to sort variable-format records with the following structure according to the sort field “Department” (CHARACTER format):

RL	Name	First name	Address	Department
		minimum record length		var. part
		maximum record length		

Contents of the file

RL	Miller	Andrew	Platerton Road 47	DV8
RL	Summer	Brenda	Thomas Circuit 1	D
RL	Becker	Frank	Solomon Hill 34	DSTQM217

The “Department” field lies in the variable part of the records to be sorted. For this field to be used as the sort key, *FIELD-EXPLICIT(POSITION=54,LENGTH=8) must be specified in the FIELDS operand, since the first byte of the field (byte 54) has to be within the minimum record length (record 2), and the maximum record length extends to byte 61.

The record length field as sort field

In variable-length records, the record length field can also be used as a sort field; for example, to sort the records according to length.
To do so, the format of the sort field should be either BINARY, CHARACTER or FIXED-POINT.
If other formats are used, an error message is issued and SORT is aborted.

Overlapping of sort fields

Sort fields may overlap provided their formats may be combined with each other. The permitted combinations are shown in the table below.

	BI	CH	VT	ED	EI	PD	ZD	FI	FL	PT	MC	EE	EC	TC	UC
BINARY (BI)	+	+	+	+	+	+	o	o	o	o	o	o	o	+	+
CHARACTER (CH)	+	+	+	+	+	+	o	o	o	o	o	o	o	+	+
VIRTUAL-TRANSLATE (VT)	+	+	+	+	+	+	o	o	o	o	o	o	o	+	+
EBCDIC-DIN (ED)	+	+	+	+	+	+	o	o	o	o	o	o	o	+	+
EBCDIC-INTERNATIONAL (EI)	+	+	+	+	+	+	o	o	o	o	o	o	o	+	+
PACKED-DECIMAL (PD)	+	+	+	+	+	-	-	-	-	-	-	-	-	+	+
ZONED-DECIMAL (ZD)	o	o	o	o	o	-	-	-	-	-	-	-	-	o	o
FIXED-POINT (FI)	o	o	o	o	o	-	-	o	o	o	o	o	o	o	o
FLOATING-POINT (FL)	o	o	o	o	o	-	o	o	o	o	o	o	o	o	o
PHYSICAL-TRANSLATE (PT)	o	o	o	o	o	-	o	o	o	o	o	o	o	o	o
MODIFY-CODE (MC)	o	o	o	o	o	-	o	o	o	o	o	o	o	o	o
EBCDIC-ISO-EBCDIC (EE)	o	o	o	o	o	-	o	o	o	o	o	o	o	o	o
EXTENDED-CHARACTER (EC)	o	o	o	o	o	-	o	o	o	o	o	o	o	o	o
TRANSLATE-CHARACTER (TC)	+	+	+	+	+	+	o	o	o	o	o	o	o	+	+
UNICODE-CHARACTER (UC)	+	+	+	+	+	+	o	o	o	o	o	o	o	+	+

Key:

+ This format combination is permitted.

- This format combination is not permitted, because it can cause data errors during the comparison. If the FIELDS operand contains such a combination, SORT issues an error message.

o This format combination can produce an undefined result during the comparison. If the FIELDS operand contains such a combination, SORT issues a warning.

Example

The following example illustrates how two sort fields may overlap. The first field starts at byte 5 and is 10 bytes long. The sort field format is CHARACTER, the sorting order ascending. As these are default values, they apply even though they are not specified explicitly in the example.

The second sort field starts at byte 10 and has a length of 7 bytes. The field format is BINARY, the sorting order descending.

Both sort fields therefore overlap at bytes 10 through 14.

```
/start-sort
//sort-records fields=(*field-explicit(position=5,length=10), -
//                     *field-explicit(position=10,length=7, -
//                                     format=*binary, -
//                                     sorting-order=*descending))
//end
```

Conversion of sort fields

The EBCDIC-DIN, EBCDIC-INTERNATIONAL, EBCDIC-ISO-EBCDIC, VIRTUAL-TRANSLATE, MODIFY-CODE, PHYSICAL-TRANSLATE, EXTENDED-CHARACTER and TRANSLATE-CHARACTER and UNICODE-CHARACTER formats include built-in conversions which are used for comparisons during sorting or merging and partly also for SORT output.

For the VIRTUAL-TRANSLATE, PHYSICAL-TRANSLATE and MODIFY-CODE formats, the code tables must be defined by the user in order to match SORT to specific requirements. Two basic code tables are available to the user for the TRANSLATE-CHARACTER format; the user is provided with the Unicode Default Collation Table (UTF-16) for the UNICODE-CHARACTER format. Further tables can easily be created using model sources (see page 50).

EBCDIC-DIN format

Sort fields in EBCDIC-DIN format enable textual sorting in conformance with the DIN standard. The aim of this textual ordering is to achieve a sequence that matches the ISO 7-bit code (German Reference Version according to DIN 66003; numbers come before letters in this sequence) but equates lowercase letters with the corresponding uppercase letters. Sort fields in EBCDIC-DIN format that contain German special characters are equated by SORT with other character combinations, as follows:

– X'00' is treated as X'40' (space character)

– 'ä' and 'Ä' are treated as 'AE'

– 'ö' and 'Ö' are treated as 'OE'

– 'ü' and 'Ü' are treated as 'UE', and

– 'ß' is treated as 'SS'.

SORT does not modify the sort fields in EBCDIC-DIN format in the records themselves, but uses auxiliary fields instead.

In EBCDIC-DIN format, SORT requires the sort fields in the following codes:

– EBCDIC.SRV.10 (Reference Version 10 of the 8-bit code) or

– EBCDIC.DF.03 (International/German DF Version 03).

SORT uses a common conversion table for both codes.

EBCDIC-INTERNATIONAL format

Sort fields in EBCDIC-INTERNATIONAL format enable textual sorting according to international conventions derived from the DIN standard. The aim of this textual ordering is to achieve a sequence that matches the ISO 7-bit code (German Reference Version to DIN 66003; numbers come before letters in this sort sequence) while equating lowercase letters with the corresponding uppercase ones. (Here, German umlauts are *not* equated, but instead are treated as special characters.)

VIRTUAL-TRANSLATE format

For sort/merge runs using sort fields in VIRTUAL-TRANSLATE format, the user must specify an ASSIGN-EXITS statement in which the user exit VIRTUAL-TRANSLATE is assigned the name of a module containing a conversion table. There is no need for the character conversion to be one-to-one, i.e. more than one character may be converted to the same value.

In VIRTUAL-TRANSLATE format each of the sort fields is converted into an auxiliary field; the auxiliary fields are then compared to determine the sorting order. The sort fields themselves are **not** modified.

PHYSICAL-TRANSLATE format

For a sort/merge run using sort fields in PHYSICAL-TRANSLATE format, the user must specify an ASSIGN-EXITS statement in which the user exit PHYSICAL-TRANSLATE is assigned the name of a module containing two user-defined conversion tables. These must contain the character strings for performing the conversion at the start and end of the sort/merge run. The second code table is necessary in order to reconvert the sort field at the end of the run. Unlike the VIRTUAL-TRANSLATE format, in this case the sort fields are converted, compared, and then reconverted. This means that the user has to preserve the uniqueness of the characters during and after the conversion and that no two characters may have the same value.

MODIFY-CODE format

Sort fields in MODIFY-CODE format refer to a MODIFY-CODE statement. Using this statement, SORT creates two conversion tables containing character assignments. The advantage of this type of conversion compared with the PHYSICAL-TRANSLATE format is that SORT is responsible for ensuring a unique character assignment. SORT uses the two conversion tables to convert the sort field character strings in MODIFY-CODE format in the record itself at the start and end of the sort/merge run.

EBCDIC-ISO-EBCDIC format

Based on the format specification of the individual sort fields, SORT converts the character strings of sort fields in EBCDIC-ISO-EBCDIC format into the internal comparison code and then back again to the output code.

Code for SORT input	Code for internal merging	Code for SORT output
EBCDIC	ASCII	EBCDIC

The conversion tables for doing so are shown on the next pages.

Code conversion table for EBCDIC to extended ASCII

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	00	21	24	25	03	09	0E	FF	26	27	2A	0B	0C	31	34	35
1	36	11	37	60	15	0D	08	10	61	6A	1A	6B	1C	1D	1E	1F
2	20	12	22	23	1B	0A	17	01	28	29	06	2B	2C	2D	2E	2F
3	30	13	32	33	02	18	0F	04	38	39	3A	3B	3C	3D	3E	3F
4	40	14	6C	6D	6E	6F	71	7A	7B	7C	E0	4E	5C	48	4B	FC
5	46	16	7D	7E	7F	81	82	83	84	85	41	44	4A	49	5B	19
6	4D	4F	62	63	64	65	66	67	68	69	BE	4C	45	BF	5E	5F
7	70	05	72	73	74	75	76	77	78	79	5A	43	A0	47	5D	42
8	80	E1	E2	E3	E4	E5	E6	E7	E8	E9	8A	8B	8C	8D	8E	8F
9	90	EA	EB	EC	ED	EE	EF	F0	F1	F2	9A	9B	9C	9D	9E	9F
A	86	87	F3	F4	F5	F6	F7	F8	F9	FA	88	89	91	92	93	94
B	95	96	97	98	99	C1	C2	C3	C4	C5	C6	BB	BC	BD	C7	C8
C	C0	A1	A2	A3	A4	A5	A6	A7	A8	A9	CA	CB	CC	CD	CE	CF
D	D0	AA	AB	AC	AD	AE	AF	B0	B1	B2	DA	DB	DC	DD	DE	DF
E	07	C9	B3	B4	B5	B6	B7	B8	B9	BA	D1	D2	D3	D4	D5	D6
F	50	51	52	53	54	55	56	57	58	59	D7	FB	D8	FD	D9	FE

Code conversion table for extended ASCII to EBCDIC

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	00	27	34	04	37	71	2A	E0	16	05	25	0B	0C	15	06	36
1	17	11	21	31	41	14	51	26	35	5F	1A	24	1C	1D	1E	1F
2	20	01	22	23	02	03	08	09	28	29	0A	2B	2C	2D	2E	2F
3	30	0D	32	33	0E	0F	10	12	38	39	3A	3B	3C	3D	3E	3F
4	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
5	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
6	13	18	62	63	64	65	66	67	68	69	19	1B	42	43	44	45
7	70	46	72	73	74	75	76	77	78	79	47	48	49	52	53	54
8	80	55	56	57	58	59	A0	A1	AA	AB	8A	8B	8C	8D	8E	8F
9	90	AC	AD	AE	AF	B0	B1	B2	B3	B4	9A	9B	9C	9D	9E	9F
A	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
B	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	BB	BC	BD	6A	6D
C	C0	B5	B6	B7	B8	B9	BA	BE	BF	E1	CA	CB	CC	CD	CE	CF
D	D0	EA	EB	EC	ED	EE	EF	FA	FC	FE	DA	DB	DC	DD	DE	DF
E	4A	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
F	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	FB	4F	FD	FF	07

EXTENDED-CHARACTER format

Sort fields in EXTENDED-CHARACTER format are used for sorting text in the sequence which is compatible with the coded character set. As with PHYSICAL-TRANSLATE, the sort fields are converted, compared, and then reconverted. The internal code table depends on the coded character set (CCS) used.

SORT tries to locate the CCS for sorting the data records though evaluating the following sources:

– catalog entry of the input file

– SET-RECORD-ATTRIBUTES statements of the sort run
(no input file available)

– if no input file is available and no CCSN is specified via the SET-RECORD-ATTRIBUTES statement, then SORT uses the EDF03IRV code.

If the CCSN of a code is located which is not defined in XHCS, the sort run is terminated with message SRT1258.

With an ISAM output file, the EXTENDED-CHARACTER format must not be used for sort fields which form the ISAM key. The ISAM keys must be sorted in ascending order as specified by the host code, not according to any other code. If an error is detected, the sort run is terminated with message SRT1261.

TRANSLATE-CHARACTER format

Two conversion tables are required for the TRANSLATE-CHARACTER format. SORT uses these tables to generate the code of the character to be sorted for the sort operation. The first table contains the code with which the character to be sorted is equated. The second table contains the code of the character which is added to the character to be sorted. SORT decides whether to simply equate the character or to also expand it on the basis of the location of the character's code in the second table. If it is X'00', the character is equated; otherwise, the character from the second table is added to the equated character. After equating and expanding the character, SORT arranges the sort fields in the sorting order of the CCS.

The characters of the sort field can be influenced in two ways for sorting depending on the entries in the two tables:

– **Equating of characters**

  For equating, the code of the character with which the character to be sorted will be equated is entered in the first table in the code location of the character to be sorted. The value X'00' must then be entered in the second table in the code location of the character.

  Example: 'a' is to be equated with 'A'.

  The code of the character 'A' (=X'C1') must occupy the code location of 'a' (=X'81') in the first table. The value X'00' must be entered in the code location of X'81' in the second table (see the tables on page 51f).

– **Expanding a character to two replacement characters**

  For expanding, the code of the *first* replacement character is entered in the first table in the code location of the character; the code of the *second* replacement character is entered in the second table.

  Example: 'ä' is to be converted to 'AE'.

  The first replacement character 'A' (=X'C1') must be entered in the first table in the code location of 'ä' (=X'AB' for EBCDIC.SRV.10 or X'FB' for EBCDIC.DF.03). The second replacement character 'E'(=X'C5') must be entered in the same code location in the second table (see the tables on page 51f).

The sorting process is the same as with VIRTUAL-TRANSLATE, i.e. the sort fields are converted into auxiliary fields and the auxiliary fields are compared with each other to determine the sorting order. The internal code table depends on the coded character set used. The CCS for sorting the data records is specified in the same way as it is with EXTENDED-CHARACTER.

The sort fields are not modified.

Character conversion does not necessarily have to be unique, i.e. more than one character can be represented by the same value.

In the case of an ISAM output file, the TRANSLATE-CHARACTER format cannot be used for sort fields which create the ISAM key. The ISAM keys must be arranged in ascending order as specified by the host code, not according to any other code. If an error is detected, the sort run will be terminated with the message SRT1261.

Notes on creating the TRANSLATE-CHARACTER tables:

Both tables for each coded character set are stored as object modules in the library with the logical name SYSLNK.TAB.
The name of the object module corresponds to the name of the CCS (e.g. “EDF03DRV”). In addition to the standard object modules “EDF03DRV” and “EDF03IRV”, the library also contains their corresponding sources as well as a model source which can be used by the operator for creating further object modules with conversion tables for the TRANSLATE-CHARACTER format.

The tables can also be provided via the user exit TRANSLATE-CHARACTER (see page 132). These tables then take precedence over the standard tables which are stored in the library with the logical name SYSLNK.TAB.

Please note the following when creating new tables:

– The table value X’00’ is not a valid replacement character, but rather an indication that the character in question is not to be converted.

– If X’00’ is entered in a location in the first table, it must also be entered in the same location in the second table.

– In the code location of a character that is used as a replacement character, X’00’ must be entered in the second table. Concatenation is not allowed. The characters below cannot therefore be equated as shown:

  e.g. ’ä’ = ’ae’ **and** ’a’ = ’A’ **and** ’e’ = ’E’ or
  ’ä’ = ’a’ **and** ’a’ = ’A’.

Converting with TRANSLATE-CHARACTER

Using the tables shown below, which are provided as standard with the CCS “EDF03DRV”, the characters below are equated as follows:

– lowercase letters = uppercase letters (equating of characters)

– ’ä’, ’Ä’ = ’AE’; ’ö’, ’Ö’ = ’OE’; ’ü’, ’Ü’ = ’UE’; ’ß’ = ’SS’ (equating a character with two replacement characters).

First table for TRANSLATE-CHARACTER (EDF03DRV):

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1	00	00	00	00	00	00	00	00	C1	D1	00	00	00	00	00	00
2	00	00	00	00	00	00	00	00	C2	D2	E2	00	00	00	00	00
3	00	00	00	00	00	00	00	00	C3	D3	E3	00	00	00	00	00
4	00	00	00	00	00	00	00	00	C4	D4	E4	00	00	00	00	00
5	00	00	00	00	00	00	00	00	C5	D5	E5	00	00	00	00	00
6	00	00	00	00	00	00	00	00	C6	D6	E6	00	00	00	00	00
7	00	00	00	00	00	00	E2	00	C7	D7	E7	00	00	00	00	00
8	00	00	00	00	00	00	00	00	C8	D8	E8	00	00	00	00	00
9	00	00	00	00	00	00	00	00	C9	D9	E9	00	00	00	00	00
A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B	00	00	00	00	00	00	00	00	C1	00	C1	C1	00	00	00	C1
C	00	00	00	00	00	00	00	00	D6	00	D6	D6	00	00	00	00
D	00	00	00	00	00	00	00	00	E4	00	E4	E4	00	00	00	E4
E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F	00	00	00	00	D6	00	00	00	00	00	00	00	00	00	00	E2

Second table for TRANSLATE-CHARACTER (EDF03DRV):

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
3	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
5	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
6	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
7	00	00	00	00	00	00	E2	00	00	00	00	00	00	00	00	00
8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
9	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B	00	00	00	00	00	00	00	00	C5	00	C5	C5	00	00	00	C5
C	00	00	00	00	00	00	00	00	C5	00	C5	C5	00	00	00	00
D	00	00	00	00	00	00	00	00	C5	00	C5	C5	00	00	00	C5
E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F	00	00	00	00	C5	00	00	00	00	00	00	00	00	00	00	E2

UNICODE-CHARACTER format

Sort fields in UNICODE-CHARACTER format permit text to be sorted according to the Unicode Default Collation Table. SORT currently supports the Unicode character set UTF-16.

The length of the field is specified in bytes.
The maximum length of the field is 256 bytes (128 Unicode characters).
An uneven length or a length greater than 256 bytes is acknowledged with SRT1157.
In the case of variable-length records, part of the sort field can be outside the record.
If, in the case of variable-length records, a record ends within a Unicode character (i.e. after the first byte), the incomplete Unicode character is ignored totally and comparison continues with the fill character.

The values for the sort elements (Unicode Default Collation Table) published on the Unicode Consortium website
http://www.unicode.org/Public/UCA/4.0/allkeys-4.0.0txt
can change.

Detailed information on Unicode is provided in the manual “Unicode in BS2000 [17]” and the section “Use of Unicode character sets with SORT” on page 85.

2.3.2 Remainder fields

Remainder fields are the parts of an input record having neutral format which are transferred to the output record but are not significant for sorting. They are relevant only in selection sorts (and thus also in tag sorts).

In a selection sort, only those parts of the input record contained in the SORT-RECORDS statement are transferred. Thus, if parts of the input record are to be transferred to the output record but not evaluated in the sort, they should be defined as remainder fields.

Remainder fields can be used to arrange the fields in the output file in a different order to the input file. The order of the individual fields is determined by the sequence of the specifications in the FIELDS operand of the SORT-RECORDS statement.

Remainder fields are defined by specifying *REMAINDER-EXPLICIT or *REMAINDER-SYMBOLIC in the FIELDS operand. No format specification may be made for a remainder field if no print mask has been defined for it. If this rule is violated, SORT issues a warning and ignores the specification.

– Remainder fields in variable-length records

  In variable-length input records, a remainder field may begin and end at any bit position within the fixed part of the record.
  If the output record is to be of variable length (INTERNAL=*VARIABLE in the SET-RECORD-ATTRIBUTES statement), then the last remainder field (last field of the FIELDS operand) is the only record field which may also refer to the variable part of the record. All other record fields must relate to the fixed part of the input record. SORT automatically calculates the new record length, including the variable part, and adds a corresponding record length field without the user having to supply a remainder field.

Example

From a file with the following record structure, SORT is to create an output file which contains only the fields “ZIP”, “Prod.no.” and “Qty” and is sorted according to “ZIP”:

RL	ZIP	Desti-nation	Prod.no.	Date of shipment	Qty

“ZIP” must be defined as the sort field and “Prod.no.” and “Qty” as remainder fields. This is done by means of the following statements:

```
/start-sort
//sort-records -
//     fields=(*field-explicit(position=5,length=5), -          Sort field “ZIP”
//             *remainder-explicit(position=18,length=9), -     Remainder field
                                                                “Prod.no.”
//             *remainder-explicit(position=36,length=5)), -    Remainder field
//     sort-type=*compound-record                               “Qty”
//end
```

Format of the output records:

ZIP	Prod.no.	Qty

2.3.3 Constant fields

Constant fields are used in selection and tag sorting in a similar manner to remainder fields for building a new selection record. They are fields whose contents do not change and which are included in the output record. They are particularly useful for constructing tables:

A better visual presentation of lists or tables can be achieved by using a series of one-character constant fields containing the vertical bar '|' as a separator.

Like remainder fields, constant fields have no effect on sorting. They are specified in the FIELDS operand of the SORT-RECORDS statement. The following types of constants are possible:

Decimal number	{ +nn...n / -nn...n }
Hexadecimal string	X'ss...s'
Character string	[C]'cc...c'

Key:

n Digit from 0 through 9.

h Hexadecimal digit from 0 through F.

c Any representable character. An apostrophe (') in a character string must be represented by two consecutive apostrophes ('').

Decimal numbers must be in the range $+2^{31}-1$ to -2^{31}. SORT converts the decimal number into a fixed-point number which it stores as a 4-byte constant field in the new output record. Unsigned decimal numbers are treated as positive numbers.
Hexadecimal and character strings can be up to 256 bytes in length. SORT converts the strings into constant fields whose length is determined by the specified character pattern and its length.

The total length allowed for all constant fields is 4000 bytes.

Example

SORT is to perform a selection sort in which the input records are to be sorted on the “Name” field. The “Telephone” field is to be specified as a remainder field. The constants C’|␣’, C’␣|␣’ and C’␣|’ are to be entered in the selection records for optical reasons.

Format of the input records:

First name	Name	Telephone
ANTON	SCHERER	789543
ELVIRA	MAUS	453214
KLAUS	EPPLER	814300

The associated statements are as follows:

```
/start-sort
//sort-records fields=(*constant-explicit(constant=c'| '), -
//                     *field-explicit(position=14,length=9), -
//                     *constant-explicit(constant=c' | '), -
//                     *remainder-explicit(position=23,length=10), -
//                     *constant-explicit(constant=c' |')), -
//             sort-type=*compound-record
//end
```

Output:

Format of the output records:

	Name		Telephone	
|	EPPLER	|	814300	|
|	MAUS	|	453214	|
|	SCHERER	|	789543	|

2.3.4 Sum fields

Sum fields are defined in the SUM-RECORDS statement. They are record fields which contain addable values. **Sum fields must not overlap other sum fields or sort fields.**

SORT adds together the values of these fields when records with identical sort fields are to be combined. An addition which produces an overflow is suppressed by SORT and the two records concerned are not compacted. For this reason the user should extend the sum fields by a specifiable number of bytes to the left if there is a risk of an overflow.

Formats and lengths of sum fields:

Format	Format description	Length in bytes
BINARY	Binary	2, 4, 8
FIXED-POINT	Fixed-point	2, 4, 8
PACKED-DECIMAL	Packed decimal	1 - 16
ZONED-DECIMAL	Zoned decimal	1 - 16

Depending on the particular format, the fields are padded as necessary according to the following table:

Format	Type of padding	Filler character
BINARY	Left-justified	Zero
FIXED-POINT	Left-justified	Arithmetic sign
PACKED-DECIMAL	Left-justified	Zero
ZONED-DECIMAL	Left-justified	X' F0' (EBCDIC zero)

With fields in ZONED-DECIMAL format, blanks (X’40’) are automatically converted into zeros (X’F0’). In addition, positive numbers in this format have the sign zone of their final digit position set to X’Fx’ ($0 \leq x \leq 9$).

Notes

– In selection sorts, the sum field positions refer to the selection record. If there is a change in record format, the record length field to be inserted or omitted must also be taken into account. Sum fields may be remainder and constant fields only. Extended sum fields must always refer to the first position of a remainder or constant field.

– In tag sorts, the record address fields (retrieval addresses) to be generated by SORT on the basis of *TAG-HEADER or *TAG-COMPOUND must *not* be taken into account when specifying the position of the sum fields.

Example

The file DELIVERY contains a record with the structure shown below for every completed delivery. You want to create an output file which contains a record with the number of deliveries and their total value for each delivery destination. This file is to be sorted according to “Destination”.

RL	ZIP	Destination	Prod.no.	Date of shipment	Order value

To do this, you need to define “Destination” as the sort field and “Order value”, which is to be added up to give the total value of the deliveries, as the remainder field. You also need a constant field with the value 1 in the selection record so that the deliveries for each destination can be counted.
To do this, you enter the following statements:

```
/start-sort
//sort-records -
//   fields=(*field-explicit(position=10,length=18), -        Sort field “Destination”
//           *remainder-explicit(position=47,length=10), -  Remainder field
                                                             “Order value”
//           *constant-explicit(constant=1)), -             Constant field
//     sort-type=*compound-record
```

The selection record to which the specifications in the SUM-RECORDS statement must refer in a selection sort has the following structure:

Destination	Order value	1

Here the constant field occupies four bytes because SORT stores decimal numbers by default as 4-byte fixed-point numbers. With the SUM-RECORDS statement you now define the fields which SORT is to add up in the case of the same sort key. To avoid overflows you should extend the sum field “Total value” by means of the FIELD-EXTENSION operand. The sum field “Qty.” does not need to be extended because it is already large enough with four bytes.

```
//sum-records -
//    fields=(*field-explicit(position=19,length=10, -
//                            format=*zoned-decimal,field-extension=3), -
//            *field-explicit(position=29,length=4))
//end
```

Structure of the output records:

Destination	Total value	Qty.

2.3.5 Mask fields

In selection sorting, it is possible to edit sort, remainder and sum fields in BINARY, FIXED-POINT, PACKED-DECIMAL and ZONED-DECIMAL format ready for printing. For this, a print mask is used which corresponds to the edit mask of Assembler instruction ED.

The print mask has the following format:

C’xxx...x’

The following characters are allowed as mask characters xxx...x:

– A freely definable *filler character* as the first character of the print mask. Substitute characters are not converted into filler characters.

– Control characters for editing.

  The control characters (numeric characters) of the Assembler ED instruction are represented in the print mask by substitute characters. SORT converts these into the correct control characters.

  The following substitute characters should be used in the print mask:

Digit select	# (number sign)	X' 20'
Significant start	^ (circumflex)	X' 21'

– Characters to be inserted

  All characters except “#” and “^” are treated as characters to be inserted. An apostrophe (’) for insertion must be represented in the print mask by two successive apostrophes (’’).

Note

If the edit mask contains an *even* number of control characters (“#” and/or “^”), then SORT includes an additional control character “#” immediately before the first control character. This is due to the manner in which the Assembler ED instruction operates.

Example

Specified mask: C'␣#^#.#'

␣	#	^	#	.	#

Mask as expanded by SORT:

␣	#	#	^	#	.	#

↑

Inserted control character

Restrictions

– Mask fields are permitted only in selection sorting. The length of the output records is determined by the sum of the length of the mask fields and the remaining selection fields.

– Mask fields are permitted only for the BINARY, FIXED-POINT, PACKED-DECIMAL and ZONED-DECIMAL formats.

– No bit position specifications or bit lengths are allowed with BINARY format.

– Fields in BINARY or FIXED-POINT format that are longer than 4 bytes have only their four low-order bytes taken into account during editing (warning).

– The ELIMINATE parameter is not permitted with mask fields and is ignored if specified (warning).

– Overlapping of remainder fields and sum fields specified as mask fields results in the mask field being ignored in the case of the remainder field; a warning message is also issued.

  If SORT is called as a subroutine and if the statements are transferred via a program interface, the control characters “#” and “^” of the print masks must always contain the code X’76’ or X’6A’ in order to be recognized as control characters. They are not converted according to the CCS, but are represented as X’20’ or X’21’ for the ED command of the assembler.

Rules for processing print masks

The print mask of a mask field is processed according to the same rules as an edit mask of the Assembler ED instruction.

These rules are:

1. The field to be edited (source field) and the print mask are processed from left to right.
2. The filler character (first character of the mask) is transferred unchanged as the first character of the output field.
3. The digits in the source field are then moved to the output field in accordance with the characters of the print mask (control characters and characters to be inserted), as follows:
   - Each digit in the source field is moved to the output field in zoned format, replacing the associated control character. Leading zeros are replaced by the filler character until the first non-zero digit is encountered in the source field or the control character "^" (significant start digit) occurs in the print mask.
   - Characters to be inserted are replaced in the output field by the filler character until the first non-zero digit is encountered in the source field or the control character "^" occurs in the print mask. All further characters transferred are left unchanged.
   - A positive sign in the source field causes the remainder of the print mask to be replaced by the filler character. A negative sign results in it being transferred unchanged.

Special cases

If there are more digits in the source field than control characters in the print mask, SORT truncates the source field, starting from the left. If this results in non-zero digits being lost, SORT terminates with an error message.

If there are fewer digits in the source field than control characters in the print mask, SORT pads out the output field with leading zeros.

Examples 1-4 below illustrate how the print mask is processed.

Example 1

Source field (PACKED-DECIMAL format):

0	0	1	2	0	2	0	D

Mask specified:

*	#	#	#	#	#	.	#	#	–

The source field is edited as follows:

1. The first character to be transferred is the filler character “*”, which goes into the left-justified byte of the output field.
2. Leading zeros are replaced by the filler character.
3. The first non-zero digit in the source field causes all subsequent digits to be transferred in zoned decimal format (i.e. unpacked).
4. The punctuation character “.” is inserted.
5. The negative sign in the source field is transferred to the output field as a minus sign.

1.	2.		3.			4.	3.		5.
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓

This produces the output field:

*	*	*	1	2	0	.	2	0	–

Example 2

Source field (PACKED-DECIMAL format):

0	0	0	0	0	6	8	C

Mask specified:

␣	#	#	.	#	^	#	,	#	#	−

The source field is edited as follows:

(1) The first character to be transferred is the filler character "␣", which goes into the left-justified byte of the output field.

(2) Leading zeros are replaced by the filler character until the control character "^" is encountered in the mask. The punctuation character "." and the control character "^" are also replaced by the filler character.

(3) The zero immediately to the right of the control character "^" is transferred.

(4) The punctuation character "," is inserted.

(5) The digits 6 and 8 are transferred.

(6) As the packed decimal number is positive, the minus sign in the print mask is replaced by the filler character.

1.	2.					3.	4.	5.		6.
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓

The output field then looks like this:

␣	␣	␣	␣	␣	␣	0	,	6	8	␣

Example 3

Source field (PACKED-DECIMAL format):

0	0	0	0	0	0	0	3	8	C

Mask specified:

␣	#	^	#	.	#	#	#	–

Because the specified mask contains six control characters, i.e. an even number, SORT extends the specified mask to include a digit selection character “#”.

Mask as expanded by SORT:

␣	#	#	^	#	.	#	#	#	–

The source field is edited as follows:

The number of digits in the source field is 2 greater than the number of control characters. Therefore the two zeros in the first byte of the source field are truncated. Non-zero digits in the first byte would cause SORT to terminate with an error message.

Output field:

␣	␣	␣	␣	␣	0	.	0	3	8	␣

Example 4

Source field (FIXED-POINT format):

F	F	F	6	8	F	A	D

Equivalent to decimal: -618579

Mask specified:

*	#	#	#	␣	#	^	#	␣	#	#	#	–

The number of control characters in the mask is 3 greater than the number of digits in the source field (9 control characters for 6 digits). SORT therefore expands the source field on the left with 3 leading zeros.

Source field (expanded):

0	0	0	6	1	8	5	7	9	D

As the packed decimal number is negative, the minus sign from the print mask is transferred to the output field.

Output field:

*	*	*	*	*	6	1	8	␣	5	7	9	–

2.3.6 Match fields and match constants

Match fields and match constants are used to define conditions in the SELECT-INPUT-RECORDS statement to control whether records are to be included in or omitted from the sorting process. In these conditions, an area of the input record, a “match field”, can be compared either with a second match field or with a fixed value, a “match constant”.

Match fields

The following formats and lengths are permissible for the match fields:

BI	Binary	1 - 256
CH	Character	1 - 256
FI	Fixed-point	1 - 256
PD	Packed decimal	1 - 16
ZD	Zoned decimal	1 - 16

Match fields may overlap within a record provided the user ensures that there is no violation of the format representation and that no data errors occur as a result of the comparison operations.
Match fields may also overlap without constraints with sort fields or sum fields. Relations between match fields with different formats are, however, subject to restrictions.

Match constants

Match constants have the same format as constant fields.

Decimal number	$\left\{\begin{matrix}+nn...n\\-nn...n\end{matrix}\right\}$
Hexadecimal string	X'ss...s'
Character string	[C]'cc...c'

Key:

n Digit from 0 through 9.

h Hexadecimal digit from 0 through F.

c Any representable character. An apostrophe (') in a character string must be represented by two consecutive apostrophes ('').

Decimal numbers must be within the range $+2^{31}-1$ to -2^{31}.

The total length of all match constants must not be more than 4000 bytes.

Match relations

With match relations the format of the first (or only) match field must be compatible with that of the second, or with the type of the match constant. The combinations allowed are shown in the table below.

Permitted format combinations

1st match field	2nd match field					Match constant		
	BI	CH	FI	PD	ZD	Decimal number	Hexa-decimal string	Character string
BI	+	+	-	-	-	-	+	+
CH	+	+	-	-	-	-	+	+
FI	-	-	+	-	-	+	-	-
PD	-	-	-	+	+	+	-	-
ZD	-	-	-	+	+	+	-	-

Key:

+ This format combination is permitted.

- This format combination is not permitted. If such a combination is specified, SORT issues an error message.

If the match fields used in relations are not the same length, SORT pads out the shorter to the same length as the longer according to the table below. Match fields in numerical format (FI, PD, ZD) are always filled on the left, and match fields in string format (BI, CH) on the right.

FI	right-justified	on left	Arithmetic sign
PD	right-justified	on left	Zero
ZD	right-justified	on left	X' F0' (zero)
BI	left-justified	on right	Zero
CH	left-justified	on right	X' 40' (blank)

With variable-length records, the match fields (or parts thereof) may extend into the “variable part” of the record. Depending on the format, SORT pads out these notional match fields according to the above table.

Match constants too may have their length aligned with the associated first match field. Numerical constants are lengthened or shortened to or from the left, character constants to or from the right.

Example

The output of this sort run should only contain input file records which have C’@@@@@’ in the first five bytes, and which do not have the same contents in the eight bytes as of byte 7 and in the eight bytes as of byte 22.

```
/start-sort
/sort-records ...
//select-input-records condition=(1,5),eq,c'@@@@@',and,not((7,8),eq,(22,8))
//end
```

2.3.7 Symbolic names

Sort, remainder, sum and match fields, as well as insertion and match constants and print masks, can be assigned symbolic names by means of the ADD-SYMBOLIC-NAMES statement. These can then be used in subsequent statements to refer to the fields.

Symbolic names are useful for assigning a meaning to the objects addressed, thus making an application easier to use and increasing its security.

For fields, names are assigned by specifying their position, length and format; for constants and masks, via their value.

Up to 255 different symbolic names may be used. Lowercase letters are converted to uppercase. The length of a name is limited to 20 characters.

Before it can be used, a symbolic name must be defined by means of the ADD-SYMBOLIC-NAMES statement according to its later use as a field, constant or print mask.

Each symbolic name may only be used once in the ADD-SYMBOLIC-NAMES statement.

Example

Contents of file RESTAURANT.SAM.FIX

```
Orlando's           Thompson Street 62        220061  Italian
Java                Hope Street 51            522221  Indonesian
Golden Fleece       Arran Street 44           242437  Yugoslavian
Le Gourmet          Lime Street 46            505397  French
Palenque Mexico     Millwood Drive 2          980149  Mexican
Strawberry          Sauchiehall Street 8      595521  Vegetarian
Persepolis          Salford Square 20         597004  Persian
Vietnam             Thurston Street 47        522518  Vietnamese
Chayota's           Thurston Street 60        292742  Japanese
Willi's Bar         Westland Street 113       748293  German
1                   21                        48      56
```

The ADD-SYMBOLIC-NAMES statement can be used to define symbolic names and to assign these to fields. In the SORT-RECORDS statement, the sort fields are addressed via these symbolic names.

```
/start-sort
//assign-files input-files=restaurant.sam.fix,output-file=restaurant.sort
//add-symbolic-names fields=(restaurantname(position=1,length=20), -
//                           adresse(position=21,length=27))
//sort-records fields=(*field-symbolic(sort-field-name=restaurantname), -
//                     *field-symbolic(sort-field-name=adresse)
//end
```

Contents of output file RESTAURANT.SORT

```
Chayota's          Thurston Street 60
Golden Fleece      Arran Street 44
Java               Hope Street 51
Le Gourmet         Lime Street 46
Orlando's          Thompson Street 62
Palenque Mexico    Millwood Drive 2
Persepolis         Salford Square 20
Strawberry         Sauchiehall Street 8
Vietnam            Thurston Street 47
Willi's Bar        Westland Street 113
```

2.3.8 Overlap table for the different field types

	Sort field	Sum field	Remainder field	Match field	Constant field
Sort field	*	-	+	+	-
Sum field	-	-	+	+	+
Remainder field	+	+	+	+	-
Match field	+	+	+	+	-
Constant field	-	+	-	-	-

Key:

* Permissible overlaps are specified for each field type according to the relevant format.

\+ Overlapping is permitted.

\- Overlapping is not permitted.

Example of overlapping field types

The input to SORT is to consist of records with the following structure:

Name	First name	Address	Telephone
HILLER	PETER	CHURCH ROAD 2	700781
BECKER	THEODORE	MARKET SQUARE 3	378543

SORT is to process these input records as follows:

A selection sort is to produce selection records which are sorted according to the sort key “Name – First name” and contain the field “Telephone” as a remainder field. In addition, another field “IT”, containing the initials of the names, is to be prefixed. This field comprises two remainder fields, each with the length 1, whose position corresponds to the positions of both sort fields.

The statements required are as follows:

```
/start-sort
//sort-records fields=(*remainder-explicit(position=1,length=1), -
//                     *remainder-explicit(position=10,length=1), -
//                     *field-explicit(position=1,length=18), -
//                     *remainder-explicit(position=38,length=8)), -
//             sort-type=*compound-record
//end
```

Output:

IT	Name	First name	Telephone

This example contains the following overlapping fields:

The first byte of the sort field overlaps with the remainder field located at the same position. There is also an overlap from byte 1 to byte 9 with the match field. A further overlap results with the second remainder field at byte 10.

2.4 Record processing and modification in SORT

During a sort/merge run the records read in by SORT can be processed or modified internally as follows (see next page, Figure 6):

- Record selection during input
- Record format and record length modification based on specifications in the SET-RECORD-ATTRIBUTES statement
- Record length modification based on specifications in the SORT-RECORDS statement during selection sorting
- Record length modification based on specifications in the SUM-RECORDS statement when extending the sum fields
- Modification of contents
- Summation of records (SUM-RECORDS statement)

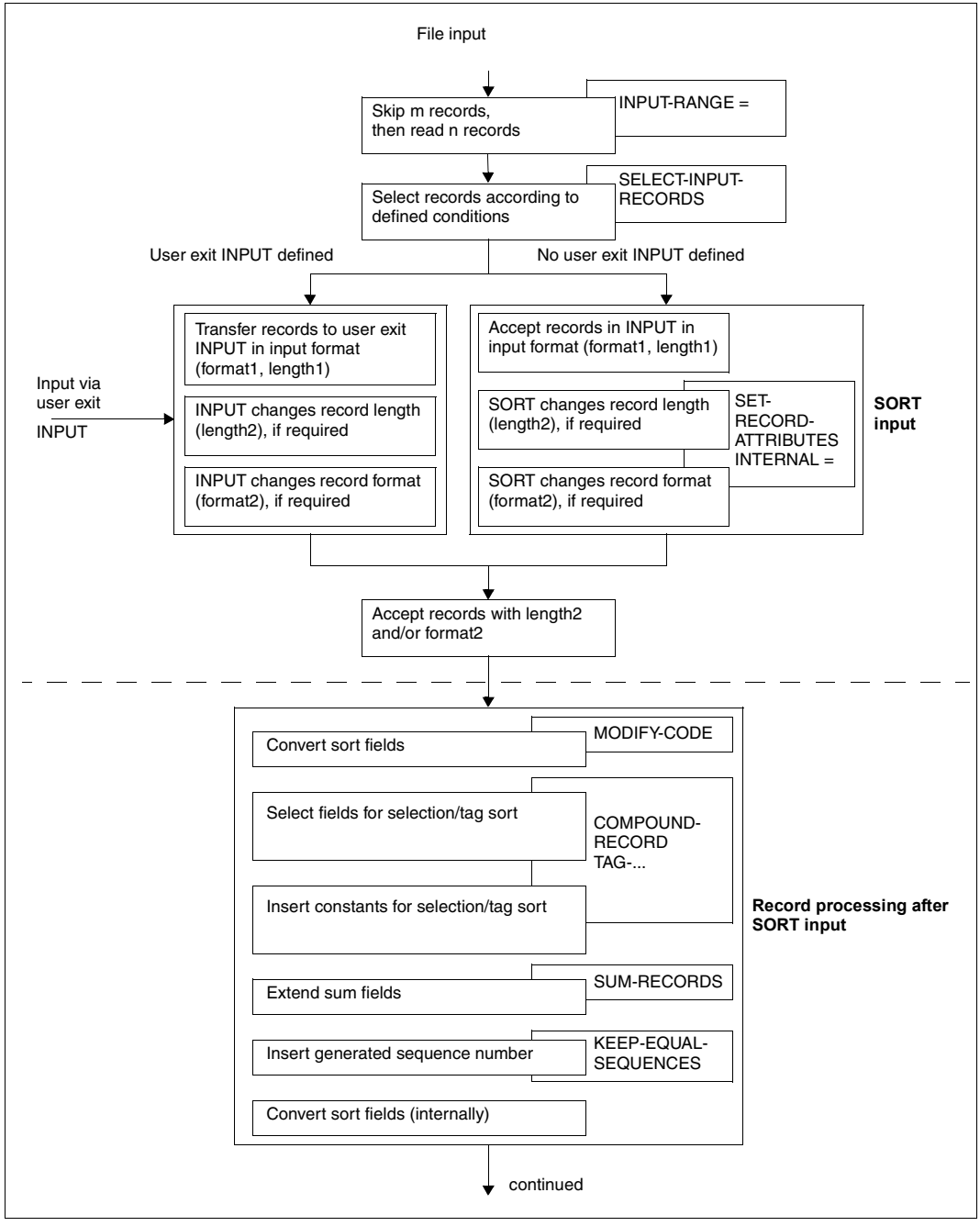


Figure 6: Record processing and modification in SORT (part 1 of 2)

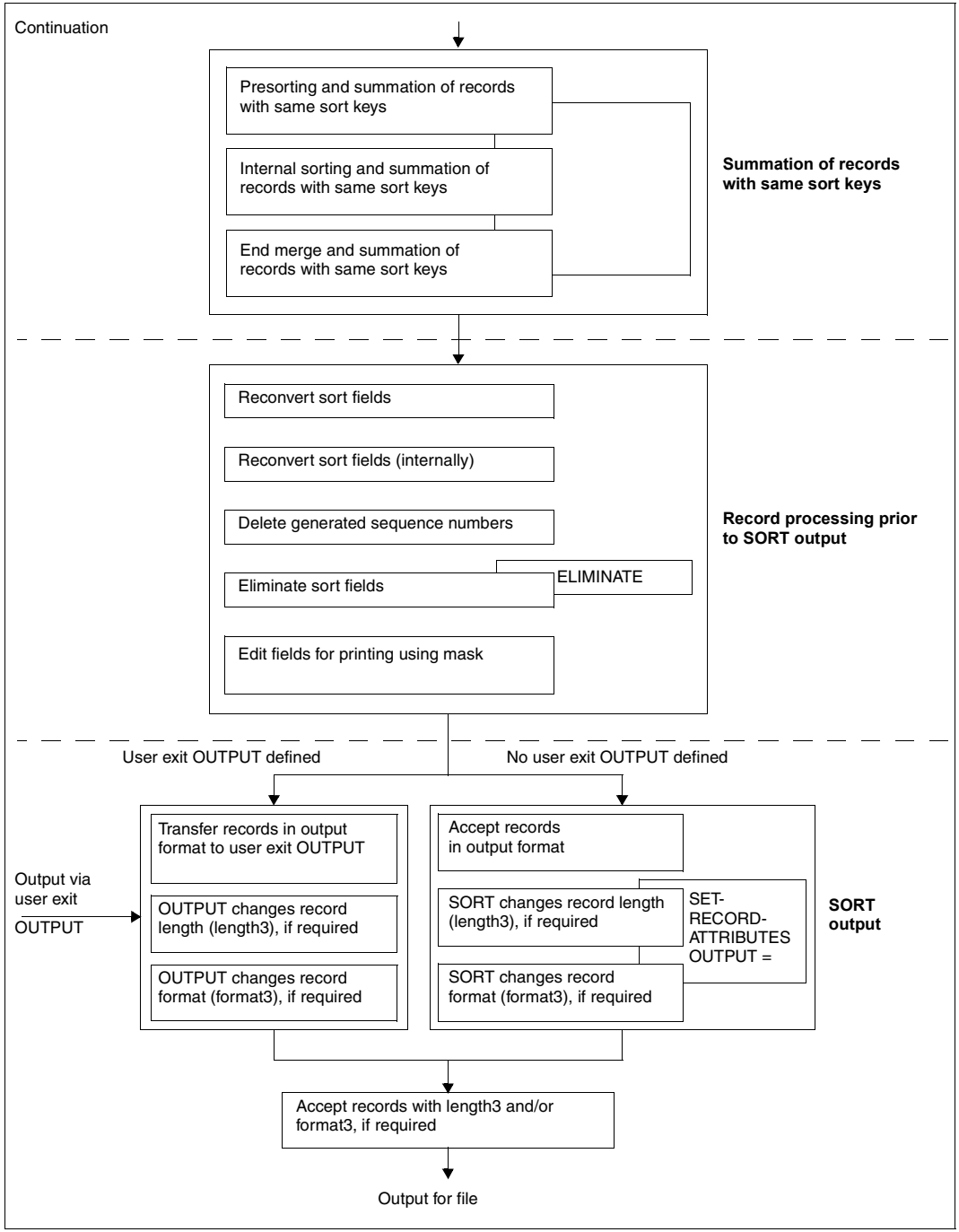


Figure 6: Record processing and modification in SORT (part 2 of 2)

Record selection during input

The user can include or exclude input records from the sort/merge run as follows:

- SORT-RECORDS statement, INPUT-RANGE operand
  Specifying the INPUT-RANGE operand for a sort run determines how many records will be skipped at the beginning of the input file and how many will subsequently be sorted (see the SORT-RECORDS statement, page 175).
- SELECT-INPUT-RECORDS statement
  Those records in the input file that meet the condition set in the SELECT-INPUT-RECORDS statement are included in the sort/merge (see page 156).
- User exit INPUT
  A user routine for the user exit INPUT (ASSIGN-EXITS statement) can be used to transfer, insert or exclude records during the sort/merge run (see chapter “SORT user exits” on page 233).

Record format and record length modification based on specifications in the SET-RECORD-ATTRIBUTES statement

The format and length of the input file records (current status) cannot be modified by SORT. Modifications are only possible when switching from

- SORT input to internal SORT processing (internal record format and internal record length)
- internal SORT processing to SORT output (output record format and length).

These changes to the record format and record length are defined by means of the SET-RECORD-ATTRIBUTES statement (see the SET-RECORD-ATTRIBUTES statement, page 160). The following points apply:

- Current status
  The format and length of the input file records are determined primarily by the ADD-FILE-LINK command or the catalog entry (RECORD-FORMAT, RECORD-SIZE) for the input file. If these specifications are not available (e.g. inputs via the user exit INPUT), the format and length of the input records must be defined by means of a SET-RECORD-ATTRIBUTES statement.

If a SET-RECORD-ATTRIBUTES statement is specified, then

- the format entry *VARIABLE() or *FIXED() in the INPUT operand determines the record format of the input records, and
- the length specified in the MAXIMUM-RECORD-SIZE or RECORD-SIZE operand determines the length of the input records.

If a SET-RECORD-ATTRIBUTES statement has been specified even though there is an existing catalog entry, and the specifications made in the statement do not tally with the attributes of the input file, SORT takes over the input file attributes and issues a warning message.

- Variable-length input records
  With variable-length records, SORT uses an input record length equal to the **maximum value** specified in the RECORD-SIZE entries of all the input files and the entry in the MAXIMUM-RECORD-SIZE operand of the SET-RECORD-ATTRIBUTES statement. If these specifications are not made, the value specified for BUFFER-LENGTH is used as the input record length.
- File attributes of the output file
  If the ADD-FILE-LINK command or catalog entry already includes record format and length specifications for the output file, for fixed-length records these values must match the values that are specified in the SET-RECORD-ATTRIBUTES statement or that have been calculated by SORT. If the values do not match, an error message is issued and SORT terminates with an error condition. For variable-length records, the maximum record length determined by SORT is entered in the catalog entry for the output file.
- Record format/length modification using the SET-RECORD-ATTRIBUTES statement
  Internal modifications to the record format are defined using the operand INTERNAL=*VARIABLE() or *FIXED() of the SET-RECORD-ATTRIBUTES statement. Modifications to the record format of the output records are defined using the operand OUTPUT=*VARIABLE() or *FIXED() of the SET-RECORD-ATTRIBUTES statement.
- Record format/length modifications via defined user exits
  If the user exits INPUT and/or OUTPUT are defined, SORT does **not** carry out any modification to the record format or record length. The routines linked to the user exits are responsible for this.
  - User exit INPUT is responsible for the internal record format and internal record length.
  - User exit OUTPUT is responsible for the record format and length of the output records.

- Record format/length modifications via undefined user exits

  If no INPUT and/or OUTPUT user exits are defined, SORT performs the record format and length modifications, as follows:

  – Record format modification
  For a change of format from fixed to variable record length, SORT recalculates the record length and prefixes a (4-byte) record length field to the converted record. For a change of format from variable to fixed record length, SORT removes the (4-byte) record length field from the beginning of the record. The converted records are padded out to the maximum record length (internal or output records) according to the values specified in the FILLER operand of the SET-RECORD-ATTRIBUTES statement.

  – Record length modification
  Record length modifications effected by SORT result in the following:
  Fixed-length records are shortened by truncation or lengthened by padding according to the value specified in the FILLER operand of the SET-RECORD-ATTRIBUTES statement. Variable-length records are shortened (truncated and record length field corrected) only if they exceed the new length. However, if variable-length records do not exceed the maximum record length, the existing record length is left unchanged. The number of truncated records is indicated by message SRT1054.

Record length modification based on entries in the SORT-RECORDS statement

- ELIMINATE operand
  If the ELIMINATE operand is specified in the sort field description, SORT shortens the record by the length of this sort field.
- SORT-TYPE operand
  In selection sorts (SORT-TYPE=*COMPOUND-RECORD) and tag sorts (SORT-TYPE=*TAG-COMPOUND/*TAG-TRAILER/*TAG-HEADER), the record length is modified. The new record length of the selection record is the sum of the lengths of the sort fields, constant fields, remainder fields and address field.

Record length modification based on entries in the SUM-RECORDS statement

A field extension can be specified in the field description of the SUM-RECORDS statement. The record is then lengthened by the specified number of bytes.

Modification of contents

Changes in content can be produced as a result of:

- Record selection specifications in the SORT-RECORDS statement.
- Code conversion for sort fields in PHYSICAL-TRANSLATE format.
- Record processing via the INPUT and/or OUTPUT user exits.
- ELIMINATE operand entry if the sort field description of the SORT-RECORDS or MERGE-RECORDS statement.
- Padding of records according to the FILLER operand value in the SET-RECORD-ATTRIBUTES statement in the case of record format and record length modifications.
- Editing by means of print masks.

Summation of records

The SUM-RECORDS statement can be used to compact records having the same sort keys and identical sort field contents, i.e. the specified sum fields are added together.
This compaction of records extends over the entire sort/merge operation, i.e. through the presorting phase, internal merging and end merge (see section “Sum fields” on page 58).

2.5 SORT in an XS environment (31-bit addressing)

SORT is able to execute in 24-bit or 31-bit addressing mode. The addressing mode is selected when SORT is started with the START-SORT command or invoked as a subroutine by a main program. SORT itself does not effect any switchover in the hardware addressing mode.

The following points should be noted with regard to the addressing mode:

– in 24-bit addressing mode

  User routines invoked by SORT may be used in their old (unconverted) form (equivalent to AMODE/RMODE=24, PARMOD=24). But they may also be in a wholly or partially converted state (equivalent to AMODE/RMODE=ANY, PARMOD=31). At the relevant PARMOD interface of the user routine EXLST-FOR-INPUT/EXLST-FOR-OUTPUT, SORT adapts to the specified EXLST macro and generates the associated FCBs with a suitable PARMOD.
  The functions in SORT may be used without constraints.

– in 31-bit addressing mode

  The user exits INPUT, OUTPUT and EXTERNAL-COMPARE must be used in combination with PARAMETER-MODE=ANY.

  For all SORT interfaces to the user exits, 31-bit addressing applies.

  For user exits EXLST-FOR-INPUT/EXLST-FOR-OUTPUT, the EXLST macros must be assembled using PARMOD=31.

  FCB reference tables must not be used in 31-bit addressing mode.

If the conventions for SORT-XS conversion are not observed, SORT or the Dynamic BinderLoader issue error messages.

2.6 Using extended character sets in SORT

Computer systems (hosts) and data display terminals each operate with one **character set**, i.e. a set of letters, digits and characters used to form words and other basic components of a language.

By expanding the character set, country-specific characters, such as umlauts (German) and accents (French), can also be offered by a specific character set.

A **coded character set (CCS)** is the unique representation of the characters in a character set in binary form. The content of a coded character set and its rules, such as sorting order and conversion guidelines, are defined by international standards.

Example: In the coded character set EBCDIC.DF.03 (German reference version), the character “ä” is represented by the byte X’FB’, and in EBCDIC.DF.04-1 by X’43’.

Every coded character set (also simply called “code”) is referred to by its unique name (**coded character set name, CCSN**).

Example: The code EBCDIC.DF.03 (international reference version) is referred to as “EDF03IRV”.

The appendix at the end of the “XHCS” manual ([11]) provides a list of all existing codes.

Extended codes expand the current

7-bit codes EBCDIC.DF.03 for hosts and
ISO646 for data display terminals
with approx. 90 characters used

by

8-bit codes EBCDIC.DF.04-x for hosts and
ISO8859-x for data display terminals
with approx. 190 characters used.

The specification of an 8-bit code ISO8859-x for instruciton files is not supported.

Requirements

The software product **XHCS** (e**X**tended **H**ost **C**ode **S**upport) is required for generating expanded codes in the host and for transmitting data between the host and data display terminals. A detailed description of the principles and functions of XHCS, a list of code tables and the names of standard codes are provided in the “XHCS” manual [11].

For inputting and outputting extended character sets on data display terminals, 8-bit terminals are required as hardware.
You can use the software product VTSU to test data display terminals for 8-bit capability.

SORT requires and uses XHCS functions:

– if the sort fields contain the EXTENDED-CHARACTER or TRANSLATE-CHARACTER format or
– if the statements contain character constants and the CCSN of the data records is different from the CCSN of the statements.

If SORT requires XHCS functions and XHCS is not available, the sort run is aborted with message SRT1257.

2.6.1 SORT-specific applications of extended character sets

Data records can be sorted using extended codes. The procedures and requirements for doing so are described on the following pages.

2.6.1.1 Sorting with extended codes

SORT offers the EXTENDED-CHARACTER and TRANSLATE-CHARACTER formats for sorting with extended codes. With these formats you can sort data records in the sorting order of the codes in question.
You can enter these formats for the sort fields in the SORT statements ADD-SYMBOLIC-NAMES, MERGE-RECORDS and SORT-RECORDS and in the SORT-FILE command.

The example on page 320 shows how both formats can be used.

The restrictions that apply to the overlapping of sort fields are shown in the table “Overlapping of sort fields” on page 42. Sort fields are 1-256 bytes in length.

SORT tries to obtain the CCSN for sorting the data records by evaluating the following sources:

– catalog entry of the input file
– SET-RECORD-ATTRIBUTES statement of the sort run
(no input file available)
– if no input file is available and no CCSN is provided by the SET-RECORD-ATTRIBUTES statement, SORT uses the code EDF03IRV.

If the CCSN of a code is obtained which is not defined in XHCS and if SORT requires XHCS functions, the sort run is aborted with message SRT1258.
In no XHCS functions are needed, the CCSN obtained is accepted as a CCS attribute in the catalog entry of the output file without being tested.

With an ISAM output file, the EXTENDED-CHARACTER and TRANSLATE-CHARACTER formats cannot be used for sort fields which form the ISAM key. The ISAM keys must be sorted in ascending order as specified by the host code, not according to any other code. If an error is detected, the sort run is aborted with message SRT1261.

2.6.1.2 CCSN entry in the SORT files

How SORT handles the CCSN entry depends on the type of file.

– Input files:
If the data records are entered from an input file, SORT uses the CCSN of the input file to sort the data records.

Since SORT does not perform data conversion for adapting different input file codes, when multiple input files are used the input files must all have the same CCSN. Dummy input files behave neutrally. If an error is detected, the sort run is aborted with message SRT1254.

Note: No error message will appear if the different input files “EDF03IRV” and “no CCSN defined” are used. In this case, the CCSN of the data records is undefined.

If the SET-RECORD-ATTRIBUTES statement contains an input file (or files) and a CCSN entry, the CCSN of the input file(s) is used for sorting, the CCSN entry in the SET-RECORD-ATTRIBUTES statement is ignored and message SRT1256 is displayed.

– Output file:
SORT issues the CCSN of the output file; a user entry for the output file is ignored. The CCSN of the output file is specified in the same way as the CCSN of the data records.

If an input file exists, the CCSN of this file will be used.
If several input files exist, the CCSN of the first file is used.
If the first file is a dummy file, the CCSN of the first non-dummy file is used.
All input files dummy do not result in a CCSN for the output.

– Statement file:
SORT evaluates the CCSN of the statement file when reading the commands (see section “Converting character constants into the code of the data records” on page 84).

– Auxiliary, work and checkpoint files:
SORT does not issue a CCS attribute for these files; any user entry is ignored.

2.6.1.3 Explicit CCSN entry for data records and output file

If no input file is available, i.e. if the data records are input via the user exit INPUT or the access method SORTZM, the name of the CCS of the data records can be entered via the statement SET-RECORD-ATTRIBUTES, operand CODED-CHARACTER-SET=<name 1..8>.

The CCSN is accepted for the output file. If no input file is available and no CCSN is provided via the SET-RECORD-ATTRIBUTES statement, SORT uses the EDF03IRV code.

2.6.1.4 Converting character constants into the code of the data records

Constants can be defined as character strings in the following SORT statements (e.g. SET-RECORD-ATTRIBUTES FILLER='#'). These are converted by SORT from the CCS of the constants to the CCS of the data records before these constants are integrated in the output record or compared to the fields of the input record.
If the data records have a Unicode-CCSN, the conversion is rejected with SRT1260.

Constants can be entered as character strings in the following commands:

ADD-SYMBOLIC-NAMES	CONSTANTS=list-poss(256):<name(VALUE=<c-string>)>
MODIFY-CODE	SEQUENCES=list-poss(256):<c-string>
SELECT-INPUT-RECORDS	CONDITION=<text>
SET-RECORD-ATTRIBUTES	FILLER=<c-string>
SET-SORT-OPTIONS	IGNORE-CHARACTER=<c-string 1..1 >
SORT-RECORDS	FIELDS=*CONSTANT-EXPLICIT(CONSTANT=<c-string>)
SORT-RECORDS	FIELDS=*FIELD-EXPLICIT(PRINT-MASK=<c-string>)
SUM-RECORDS	FIELDS=*FIELD-EXPLICIT(PRINT-MASK=<c-string>).

SORT obtains the CCSN of the character constants when the statements (with character constants) are input via SYSDTA.

If SORT is called as a subroutine and the statements (with character constants) are transferred via a program interface, the CCSN of the character constants in the statements can be defined by the SORT statement SET-SORT-OPTIONS, STATEMENT-CCSN operand.

If no CCSN entry is available in the SET-SORT-OPTIONS statement, SORT uses the EDF03IRV code.

2.6.1.5 Conditions for mask fields

If SORT is called as a subroutine and the statements are transferred via a program interface, the control characters “#” and “^” of the print masks must always contain the code X’76’ or X’6A’ in order to be recognized as control characters. They are not converted according to the CCS, but rather are represented as X’20’ or X’21’ for the Assembler instruction ED.

2.6.1.6 Using extended character sets in user exits

The CCSN of the data records is transferred to the user at the user exits INPUT, OUTPUT and EXTERNAL-COMPARE (31-bit interface). The CCSN is only for the user’s information; comparing sort fields with EXTERNAL-COMPARE can only be done effectively with this CCSN.

2.7 Use of Unicode character sets with SORT

Unicode is a standardized alphanumeric character set and includes all known text characters in the world in a single character set. SORT currently supports the Unicode variant UTF-16 (UCS Transformation Format 16 Bit) in which each character is represented by two bytes. In UTF-16 the first 256 of the maximum of 65,536 characters correspond to the ISO Latin-1 character set (ISO 8859-1).

The Unicode standard defines a linguistic sort algorithm. Each Unicode character is assigned a collation element. The sequence in which the Unicode characters are sorted is defined with the aid of these collation elements. The collation elements are defined by means of a table supplied by XHCS (Unicode Default Collation Table). This table contains a priority for the character at various levels. SORT recognizes three levels, and these are displayed in the table below. The individual characters are always compared from left to right. The first difference determines the result of the comparison.

In BS2000 you can supply values for the collation element via XHCS, see also the “XHCS” manual [10].

Comparison level	Description	Example
Level 1	Base character	a < b role < roles < rule
Level 2	Diacritics	A < Å role < rôle < roles
Level 3	Uppercase/lowercase	a < A role < Role < rôle

Level 1 : Each base character (a,b,c, etc.) is assigned a permanent priority in the Unicode Default Collation Table. The following characters or diacritics have no influence on the sort sequence of the characters.

Level 2 : The base character has a diacritic. A diacritic is an additional character (e.g. accent, slash, dot, cedilla, tilde) which is paced in, above or below a character to define how it is pronounced or stressed. At level 1, a base character with a diacritic has the same priority as the associated base character without a diacritic. At level 2, a character with a diacritic has a higher priority than the same character without a diacritic. If the sort key is otherwise identical, the sort sequence is defined using this diacritic (see the example for level 2: role < rôle).

Level 3: The sort sequence is defined by the distinction between upper- and lowercase letters. Uppercase letters have a higher priority than lowercase letters. Level 3 is taken into account only if level 1 and level 2 are identical for the entire sort key (see the example for level 3: role < Role).

When two Unicode sort fields follow directly one after the other, the sort sequence can differ depending on whether the sort fields are specified as one or two fields in the SORT-RECORDS or MERGE-RECORDS statement. At level 2 and level 3, the sort sequence is only defined within a sort field.

A collation element can be assigned a pointer to a second collation element. For example, if a base character has a diacritic and the two characters are encoded as one character.
If a pointer refers to a second collation element with faulty content (e.g. pointer points out of the table), the sort run is aborted with SRT1089.

2.7.1 Normalization

In the case of a base character with a diacritic (level 2), the encoding of a character can vary in Unicode. Consequently several encodings can exist for one character. The character “Ä”, for example, can also be written as a string consisting of “A” and “°”. These characters are treated as equal when the comparison takes place. The normalization functions DECOMPOSE and COMPOSE are provided to unify a file. Normalization assigns a uniform format to identical characters with different encoding. The basis for normalization is provided by the XHCS normalization table, which is derived from the Unicode collation table, see also the “XHCS” manual [10].

The normalization function is executed using the XHCS macro:
NLSCNV ACTION = COMPOSE or
NLSCNV ACTION = DECOMPOSE.

COMPOSE combines a base character with the associated diacritics to form a single character.

DECOMPOSE splits a composite character into the base character and the associated diacritics. The sequence of the linked diacritics is strictly defined here.

If neither of the normalization methods is available, consequent errors can occur in the sort sequence.

2.7.2 Characters with special processing

The IGNORE-UNICODE-BLANK operand in the SET-SORT-OPTIONS statement enables particular characters in a text to be taken into account or ignored. When characters are ignored, the sort key is contracted and, if necessary, padded with the Unicode fill character at the end. As a result, the sort sequence is changed. The characters which can be ignored are blank, slash, hyphen and variable collation elements.

The example in the table below illustrates how the sort sequence changes:

Character	**Taken into account**	**Ignored**
Blank	at home at school Atlantic	at home Atlantic at school
Slash, e.g. $\frac{1}{4}$	Three characters: 1 (one), - (slash), 4 (four)	Two characters: 14

The following characters consisting of more than one base character (not diacritics) are treated in a special manner:

œ	(hex: 0153)	is equivalent to:	oe	(hex: 006F 0065)
Œ	(hex: 0152)	is equivalent to:	OE	(hex: 004F 0045)
№	(hex: 2116)	is equivalent to:	No	(hex: 004E 0366)
ß	(hex: 00DF)	is equivalent to:	ss	(hex: 0073 0073)

2.7.3 Characters which are not supported

Not all two-byte encryptions belong to a supported Unicode character. All characters which are not supported are ignored. These characters also include the two-byte characters which are outside the supported range. Currently the range from X'0000' through X'2FFF' is supported. If characters are not supported, the sort key is contracted and, if necessary, padded with the Unicode fill character at the end.

The characters which are not supported also include the NIL character (X'0000'). If a search key comprises only NIL characters, only Unicode fill characters are used for the comparison. The Unicode fill character is specified in the SET-RECORD-ATTRIBUTES statement.

If the sort table supplied by XHCS changes (e.g. through the addition of new characters), the sort sequence may also change. Characters which were previously ignored are then evaluated after they have been included in the Unicode table.

3 Files of the sort/merge program SORT

The SORT sort/merge program operates with files which can be set up either by SORT or by the user, and with files which **must** be set up by the user. Dummy files can be processed. Following are the various types of file used by SORT (see figure 7):

– sort input files
– merge input files
– output file
– work files (not when using a merge application)
– auxiliary files (not when using a merge application)
– checkpoint file
– object module library SORTMODS
– statement files

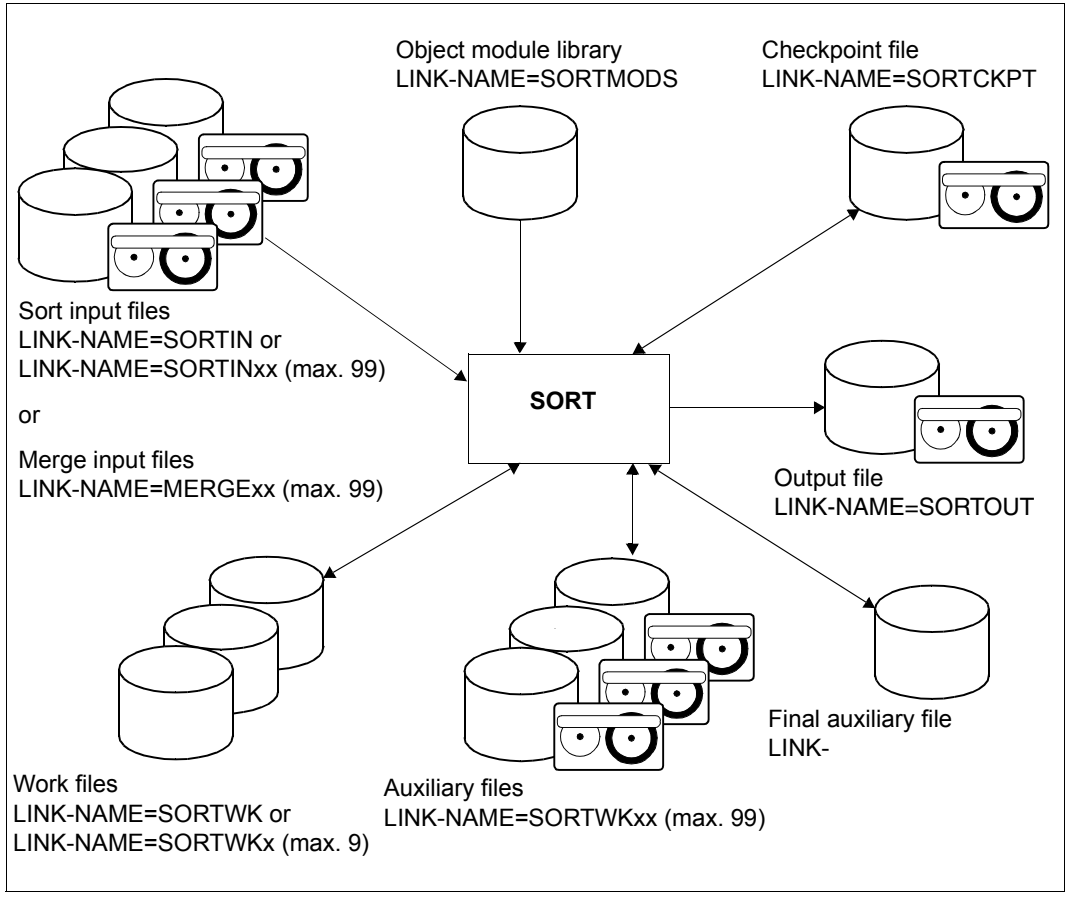


Figure 7: Files used by SORT

The necessary files are assigned either by means of the ASSIGN-FILES statement or via the file link name given in a ADD-FILE-LINK command or FILE macro. If the ASSIGN-FILES statement is used, SORT issues internal ADD-FILE-LINK commands with the appropriate file link names (SORTIN or SORTINxx, MERGExx, SORTOUT, SORTWK or SORTWKx, SORTWKxx, SORTCKPT and SORTMODS). Any TFT entries already in existence for these file link names will be overwritten.

If the same files are assigned both via the ASSIGN-FILES statement and via the ADD-FILE-LINK command, the files assigned via the command will be ignored. If the LINK-PREFIX-CHANGE operand is specified in the SET-SORT-OPTIONS statement, the “SORT” or “MERGE” prefix in the file link names used will be replaced by the specified prefix.

The IGNORE-INOUT-FILE operand of the SET-SORT-OPTIONS statement has no effect on files that have been assigned via the ASSIGN-FILES statement.

Notes

– If the user sets up the input, output, work and auxiliary files with SET-FILE-LINK, entries can be made for using fast storage media (PERFORMANCE operand). If SORT sets up these files, the value PERFORMANCE=*STD is used, i.e. the default storage medium is used.

– Input and output files can be in DIV (Data-in-Virtual) format (PAM file with BLOCK-CONTROL-INFO=*NO). SORT processes these files with the UPAM access method.

– All SORT tape files must be set up on separate tapes. The only exception permitted is when sort runs take place in which the sort input files and the sort output files are on the same tape.

Handling temporary files set up by SORT

Files created by SORT, such as auxiliary files and work files, are automatically deleted at the end of a sort operation if the sort run completed normally or terminated in error without a RESTART capability. If a run terminates abnormally with the possibility of a RESTART, SORT does not delete these files. If the RESTART is not activated, the user is responsible for deleting the temporary files.

Assigning and creating the SORT files

File type	Assigned via		Created	
	/ADD-FILE-LINK	//ASSIGN-FILES	on basis of operand entries	by SORT as necessary
Sort input file Sort input files	SORTIN SORTINxx	INPUT-FILES= INPUT-FILES=(...)	-	-
Merge input files	MERGExx	INPUT-FILES=(...)	-	-
Sort/merge output file	SORTOUT	OUTPUT-FILE=	-	-
Sort work file Sort work files	SORTWK SORTWKx	WORK-FILES= WORK-FILES=(...)	-	1 disk file
Sort auxiliary files	SORTWKxx	AUXILIARY-FILES=(...)	1 - 99	1 disk file in reserve
Checkpoint file	SORTCKPT	CHECKPOINT-FILE=	1 file	1 disk file
Object module library	SORTMODS	MODULE-LIBRARY=	-	-
Statement files	-	STATEMENT-FILES=(...)	-	-

Key:

x stands for values from 1 to 9

xx stands for values from 01 to 99

\- not allowed

Notes

– Specifying the operand LINK-PREFIX-CHANGE=<name 1..4> in the SET-SORT-OPTIONS statement causes the “SORT” or “MERGE” part of the file link name to be replaced by the specified prefix <name 1..4>. In the following file descriptions this applies to all specified file link names which include “SORT” and “MERGE” as part of the name.

– The file series x and xx must be numbered consecutively, without gaps. If there is a gap in the numbering sequence, only the files up to the gap will be processed.

– SORTIN and SORTWK may only be specified in conjunction with **one** input or work file. The file link names SORTIN and SORTINxx, or SORTWK and SORTWKx, must not be combined.

3.1 Input files

3.1.1 Input files for sort runs

SORT can sort up to 99 input files into one output file. The link to the sort/merge program is set up by means of the INPUT-FILES operand of the ASSIGN-FILES statement with the internally assigned file link name SORTIN or SORTINxx.

File link names for input files

The file link names must be specified in ADD-FILE-LINK commands or FILE macros, or are generated by SORT as a result of an ASSIGN-FILES statement.

- One input file (single-file sort)

```
/ADD-FILE-LINK LINK-NAME=SORTIN,FILE-NAME=filename,... €
```

or

```
//ASSIGN-FILES INPUT-FILES=filename,... €
```

- Multiple input files (multi-file sort)
  In order to sort a number of input files into one output file, SORT requires each of the input files to be assigned either via a ADD-FILE-LINK command or by the ASSIGN-FILES statement.

  ADD-FILE-LINK statement

  ADD-FILE-LINK LINK-NAME=SORTINxx,....

  The input files must be specified consecutively in ascending numerical order, with 01, 02, .., 99 being inserted for “xx”, according to the number of files present. For example, if there are 4 input files the file link names must be specified as follows:

```
/ADD-FILE-LINK LINK-NAME=SORTIN01,FILE-NAME=filename1 ...     1st input file
/ADD-FILE-LINK LINK-NAME=SORTIN02,FILE-NAME=filename2 ...     2nd input file
/ADD-FILE-LINK LINK-NAME=SORTIN03,FILE-NAME=filename3 ...     3rd input file
/ADD-FILE-LINK LINK-NAME=SORTIN04,FILE-NAME=filename4 ...     4th input file
```

  ASSIGN-FILES statement

```
//ASSIGN-FILES -
//       INPUT-FILES=(filename1, filename2, filename3, filename4)
```

Note

If neither an ASSIGN-FILES statement nor ADD-FILE-LINK commands or FILE macros have been specified for assigning input files, the sort input must have been defined via the user exit INPUT. If no input is defined, SORT issues an error message.

Input file = output file

In tag sorts, the input file must not be used as the output file. This applies also to those cases in which file attributes (record format, record length) are different for input and output.

File attributes of multiple input files

Multiple input files for a sort run must all have the same record format (RECORD-FORMAT=*FIXED or RECORD-FORMAT=*VARIABLE). Input files having fixed record format must additionally have the same record length. File type (ACCESS-METHOD) and block size (BUFFER-LENGTH) may be different for a sort involving multiple input files. When using the EXTENDED-CHARACTER and TRANSLATE-CHARACTER formats, the input files must all have the same CCSN since SORT does not perform data conversion for adapting different input file codes. If an error is detected, the sort run is aborted with message SRT1254.
If one input file is a POSIX file, all input files must be POSIX files.

With cataloged files, SORT checks the file attributes at the very start of the sort run. With foreign files (files for which no system catalog entry exists yet), SORT does not check the file attributes until immediately before the file is opened. This is important to remember when the input consists of a number of files.

Access rights for POSIX input files

POSIX input files are only sorted if the read access right is available. Access rights for POSIX input files are not changed by SORT.

Opening of input files

SORT opens input files with OPEN=INPUT by default. With BTAM and SAM tape files, the user can change the OPEN mode in the ADD-FILE-LINK command to OPEN-MODE= *REVERSE so that the tape input files will be read in reverse.

SORT attempts to open locked files at 1-second intervals (max. 100 tries). If still not successful after 100 attempts, SORT issues an error message. The effect of this is different depending on whether there is only one input file or multiple input files:

- One input file
  In a single-file sort, SORT terminates the run if the input file cannot be opened.
- Multiple input files
  SORT's response if one of the input files cannot be opened can be defined by means of the INPUT-OPEN-ERROR operand of the SET-SORT-OPTIONS statement.
  Possible options are:
  - *CONTINUE-NEXT-FILE(TERMINATE=*NORMAL)
    SORT reports the error, skips the file concerned and continues processing the other files.
  - *FINISH-INPUT
    SORT reports the error, terminates the input, and sorts the records read in up to that point.
  - *TERMINATE-ABNORMAL
    SORT reports the error and terminates abnormally.
  - *CONTINUE-NEXT-FILE (TERMINATE=*ABNORMAL)
    SORT reports the error, skips the file concerned, continues processing the other files and terminates as follows:
    a) in a standalone sort run, with TERM UNIT=STEP, MODE=ABNORMAL.
    b) in a sort subroutine run, with storage of the return code X'FF' the low-order byte of register 15. The first two bytes of register 15 contain the message number (e.g. open error SRT1035 produces the contents X'103500FF' in register 15).

Note

Other DMS errors cause the SORT run to terminate with an error.

Close processing of input files after termination of the sort run

The input file(s) is (are) closed by SORT (CLOSE) after being read in, and the associated file link name is released (RELEASE). SORT controls the release of the input files (or devices) on the following basis:

– type of input device (tape/disk)

– device assignment prior to sort run (PREMOUNT-LIST=0 in the ADD-FILE-LINK command)

– location of tape files (multi-file/multi-volume sets).

Using this information, SORT selects the optimal RELEASE mode for the input files. This ensures the earliest possible release for the devices. With tape files, this also optimizes tape unloading.

The KEEP-INPUT-TAPES operand of the SET-SORT-OPTIONS statement can be used to define whether input tapes are to be rewound and unloaded after they have been read, or just rewound. With multi-volume files, it should be noted, only the final tape of the multi-volume set is not unloaded in this case.

Note

In the case of multi-volume input files for which no catalog entry yet exists, if not all tapes are specified in the ADD-FILE-LINK command, SORT requests the missing tapes to be provided. If the request is refused by the operator, end-of-file is assumed and the sort run continued. Records read up to that point are sorted. If a tape request is refused by the operator in the case of a cataloged multi-volume input file, the sort run is aborted.

– For tape input files belonging to multi-file/multi-volume sets, the position of the files on the tape(s) should correspond to the order of processing so as to avoid unnecessary positioning and unloading operations
(1st file = SORTIN01, 2nd file = SORTIN02).

– Where the input is a mix of tape and disk files, the input files should be arranged according to device groups, i.e. first all tape files, then all disk files (or vice versa). By issuing a LOCK-FILE-LINK command the user can prevent the ADD-FILE-LINK assignment from being released. LOCK-FILE-LINK remains effective until an UNLOCK-FILE-LINK command is issued.

  With tape files, all tape units are released at the end of the sort run.

3.1.2 Input files for merge runs

SORT can merge up to 99 input files into one output file. Before the merge run, all the input files must be presorted according to the same sort criteria. The file link names MERGE01, MERGE02, ... , MERGExx set up the link to the sort/merge program. The input files must be specified consecutively in ascending numerical order, with 01, 02, .., 99 being inserted for “xx”, according to the number of files present. These file link names are assigned internally when the ASSIGN-FILES command is used, but they can also be specified explicitly in a ADD-FILE-LINK command or FILE macro.

```
/START-SORT
.
.
//ASSIGN-FILES INPUT-FILES=(list of input files)
//MERGE-RECORDS FIELDS=...
.
//END €
```

When the file link names are used, the associated ADD-FILE-LINK commands for, say, 4 input files are as follows:

```
/ADD-FILE-LINK LINK-NAME=MERGE01,FILE-NAME=filename1 ...    1st input file
/ADD-FILE-LINK LINK-NAME=MERGE02,FILE-NAME=filename2 ...    2nd input file
/ADD-FILE-LINK LINK-NAME=MERGE03,FILE-NAME=filename3 ...    3rd input file
/ADD-FILE-LINK LINK-NAME=MERGE04,FILE-NAME=filename4 ...    4th input file
```

Note

If the input for the merge run is defined via the user exit INPUT, then the entire input must be read in via this user exit.

File attributes of merge input files

All merge input files must have the same record format (RECORD-FORMAT=*VARIABLE or RECORD-FORMAT=*FIXED). Merge input files having fixed record format must also have the same record length. With variable-length records, the record structure must be the same in those parts of the records to which the sort field description refers.
When using the EXTENDED-CHARACTER and TRANSLATE-CHARACTER formats, the input files must all have the same CCSN since SORT does not perform data conversion for adapting different input file codes. If an error is detected, the sort run is aborted with message SRT1254.
If one of the merge input files is a POSIX file, all merge input files have to be POSIX files and must have the read access right.

Opening of merge input files

Input files for a merge run are opened with OPEN=INPUT by default. BTAM files and SAM tape files can also be read in reverse. In this case they are opened with OPEN-MODE= *REVERSE.
If an error occurs during opening of an input file, SORT message SRT1035 is issued. Other DMS errors cause SORT to issue message SRT1036. Following this, the merge run is aborted.

Close processing of merge input files

On termination of the merge run SORT closes the input files (CLOSE) and releases the associated file link names (RELEASE). The user can stop the ADD-FILE-LINK assignment from being released by issuing a LOCK-FILE-LINK command. LOCK-FILE-LINK remains in force until an UNLOCK-FILE-LINK command is issued.
With tape files, all tape units are released at the end of the merge run.

3.1.3 PAM key elimination for input files

PAM key elimination only affects input files if they are PAM files.

For files with BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK, the input record begins after the control field (start of block + 12).

For tape files with BLOCK-CONTROL-INFO=*NO, each block (padded with X'00' to the maximum block length, if necessary) is interpreted as an input record.

For files with BLOCK-CONTROL-INFO=*PAMKEY/*WITHIN-DATA-BLOCK, the check to determine whether a block being read belongs to the input file and is therefore to be processed further (interblock gap check) is carried out by comparing the coded file ID in the PAM key / control field with the catalog entry. For PAM files with BLOCK-CONTROL-INFO=*NO all the blocks are transferred.

3.2 Output file for sort/merge runs

Only one output file may be assigned, irrespective of the number of input files. Sufficient memory space has to be allocated to the output file with the SPACE operand of the CREATE-FILE command. To avoid having to make secondary space allocations, the user should specify the number of PAM pages to be read from the input files as the primary allocation. Selection criteria and record length modifications can be taken into account when determining the necessary primary allocation.
The file link name SORTOUT sets up the link to the sort/merge program. It applies equally to sort and merge runs and is either assigned internally through the ASSIGN-FILES command or specified in the ADD-FILE-LINK command or FILE macro.

Note

- If neither a ADD-FILE-LINK command or FILE macro with LINK-NAME= SORTOUT nor an ASSIGN-FILES statement has been specified, the sort output must be defined via the user exit OUTPUT. If this is also not the case, SORT issues an error message.
- If SORT cannot open the specified output file, SORT message SRT1035 is issued and the run is terminated abnormally.

File attributes of the output file

ACCESS-METHOD (FILE-STRUCTURE)

Output files can be PAM, BTAM, SAM or ISAM files. If no access method has been specified for the output file, SORT assumes the access method of the input file, or of the first input file if there is more than one. If this is not possible (e.g. input via the INPUT user exit), SORT sets ACCESS-METHOD=*SAM by default.

RECORD-FORMAT

If no format attribute is specified for the output file, SORT takes the value from the OUTPUT operand of the SET-RECORD-ATTRIBUTES statement as the default. If this is not possible, SORT uses the value set for the input file.

Exception

With the selection of sort and tag sort (see page 27 and page 29) SORT will by default specify the output files using RECORD-FORMAT=*FIXED.

RECORD-SIZE

If no record length specification has been entered for the output file, SORT uses the value specified for the first (or only) input file. If this is not possible, SORT takes the value from the OUTPUT operand of the SET-RECORD-ATTRIBUTES statement or uses the length that it has calculated.
If RECORD-SIZE=0 is defined for the first (or only) input file, this value is not taken over for the output file. If, however, RECORD-SIZE=0 is actually desired for the output file in this case, SORT must be forced to take over the value with an appropriate ADD-FILE-LINK command.

BUFFER-LENGTH

If no block size is defined for the output file, SORT uses the same block size as the input file, or as the first input file if there is more than one. If this is smaller than the output record length, or if there is no input file (i.e. the records to be sorted were passed via the INPUT user exit), then the block size for the output file is set equal to the output record length, rounded up to a multiple of STD(1) blocks (2048 bytes, K/NK2 disks) or STD(2) blocks (4096 bytes, NK4 disks). SORT takes the output record length from the SET-RECORD-ATTRIBUTES statement or calculates it separately.
If the block size is defined for the output file, SORT checks whether this is equal to or greater than the output record length. If the block size of the output file is less than the output record length, the run is aborted due to an error. With NK4 disks, the block size must be a multiple of STD(2), otherwise the run is aborted with error message SRT1252.

CODED-CHARACTER-SET

SORT issues the CCSN of the output file; any entry made by the user for the output file is ignored. The CCSN of the output file is specified in the same way as the CCSN of the data records. If “EDF03IRV” is found, no CCS attribute is defined for the output file (see section “Sorting with extended codes” on page 82).

BLOCK-CONTROL-INFO (BLKCTRL)

If the data format of the output file is not defined, the format is taken from the input file (from the first input file if there are multiple input files). If necessary, SORT corrects the data format on the basis of ACCESS-METHOD, block size and class 2 option BLKCTRL.
If the data format of the output file is defined, SORT tries to load the output file. If an error is detected, the sort run is aborted with message SRT1253.

Special characteristics of ISAM output files

- ISAM key
  For ISAM output files without defined ISAM keys, SORT uses the most significant sort field (field with sequence number 1) as the ISAM key. Thus, the most significant sort field and other sort fields used for the ISAM key may only be sorted in ascending order. If a value is explicitly specified for KEY-POSITION, this must tally with the beginning of the most significant sort field.
- Length of the ISAM key
  For ISAM output files, SORT takes the length of the most significant sort field as the key length. If KEY-LENGTH is already defined for the output file (ADD-FILE-LINK command/FILE macro or catalog entry), the key length may also extend to the secondary sort fields. These fields must be contiguous and follow immediately after the most significant sort field. Overlapping of the sort fields is not allowed. The permitted value for KEY-LENGTH is derived from the sum of the lengths of the most significant sort field and the immediately following sort fields. With variable-length records, the entire ISAM key must lie within the fixed part of the record.
- Data formats for ISAM output files
  The most significant sort field of the input file may have the data format BINARY or CHARACTER only. All other formats are *not* allowed. This is because SORT uses this field as the ISAM key for the output file. Fields having BINARY data format must begin on a byte boundary.
- Flagged ISAM files
  SORT assumes a default value of “0” for LOGICAL-FLAG-LENGTH and VALUE-FLAG-LENGTH. However, if values for LOGICAL-FLAG-LENGTH and VALUE-FLAG-LENGTH are specified in the CREATE-FILE command/FILE macro, the sum of KEY-LENGTH + VALUE-FLAG-LENGTH + LOGICAL-FLAG-LENGTH must not be greater than 255. SORT checks for this condition and aborts the run if there is an error.
- Identical sort keys
  If there are records with the same sort key, the order of output is undefined. This also applies if KEEP-EQUAL-SEQUENCES=*YES has been specified.

- Duplicate ISAM keys
  SORT uses the default setting DUPLICATE-KEY=*YES, i.e. the output file may contain records with the same ISAM keys. If this is the case, SORT issues the warning SRT1070 and proceeds with the sort/merge run as normal.
  If DUPLICATE-KEY=*NO is selected in the ADD-FILE-LINK command and duplicate ISAM keys are detected during sorting, SORT aborts the run with error message SRT1036.

  No secondary keys can be created when using duplicate ISAM keys (see the CREATE-ALTERNATE-INDEX command in the “Commands” [1] manual).

  To avoid records with identical ISAM keys from appearing in the output file, the length of the sort field must be increased, or in the case of a defined ISAM key, the length of the key must be increased.
- DMS sets the percentage of free buffer length (PADDING-FACTOR) to 15% by default. This means that the (max.) record length must not be greater than the buffer size BUFFER-LENGTH - PADDING-FACTOR. Otherwise DMS reports an error.

Opening of the output file

The open mode for output files may vary depending on the access method, as follows:

– PAM files: with OUTIN

– BTAM files: with OUTPUT or OUTIN

– SAM or ISAM files: with OUTPUT or EXTEND

If the open mode is not specified, SORT opens PAM files with OPEN=OUTIN by default. BTAM, SAM and ISAM files are opened with OPEN=OUTPUT by default.

Note

If SORT cannot open the specified output file, SORT message SRT1035 is issued and the run is terminated abnormally.

Close processing of the output file

On termination, SORT closes the output file (CLOSE). The file link name is not released.

3.2.1 PAM key elimination for the output file

SORT determines the file attribute BLOCK-CONTROL-INFO for output files as follows:

1. If the user has specified a ADD-FILE-LINK command with the BLOCK-CONTROL-INFO operand, this value applies.
2. If a value for BLOCK-CONTROL-INFO for the file is already entered in the catalog, this value applies.
3. If neither of these two conditions are met, SORT will attempt to take the value for BLOCK-CONTORL-INFO from the input file. However, this is only possible if the output medium permits it, for example, it is not possible to create a file with PAMKEY on a NONKEY disk.
4. If neither of the above is the case, the following default rule applies:

   For tape files, BLOCK-CONTROL-INFO=*DATA (for performance reasons) applies.

   For disk files, BLOCK-CONTROL-INFO depends on ACCESS-METHOD and the pre-format (SM pubsets) or the class 2 option BLKCTRL according to the following table (see also the manual “Introductory Guide to DMS” [2]):

ACCESS-METHOD	Class 2 option BLKCTRL / Pre-format		
	PAMKEY	NONKEY / NK2	NONKEY / NK4
*SAM	*PAMKEY	*DATA	*DATA
*ISAM	*PAMKEY	*DATA2K	*DATA4K
*PAM	*PAMKEY	*NO	*NO

SAM output file

With SAM output files, the connection between block length and maximum record length is as follows, depending on the value specified by BLOCK-CONTROL-INFO and the record format:

BLOCK-CONTROL-INFO	Record format	Max. record length
*PAMKEY	*FIXED	BUFFER-LENGTH
	*VARIABLE	BUFFER-LENGTH-4
*WITHIN-DATA-BLOCK	*FIXED	BUFFER-LENGTH-16
	*VARIABLE	BUFFER-LENGTH-16
*NO *)	*FIXED	BUFFER-LENGTH
	*VARIABLE	BUFFER-LENGTH-4

*) For tape files only

PAM output file

With BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK, the output record is entered after the control field (start of block + 12).
For a tape file with BLOCK-CONTROL-INFO=*NO, each individual output record (padded with X'00' if necessary) is written to the tape in the block length.

3.2.2 POSIX output file

Access rights of an existing POSIX output file are not changed by SORT. A POSIX file which is created by SORT is given read and write access rights for the user.

3.3 Work files

For sort runs in which the amount of data to be sorted is greater than the available CORE memory, SORT requires a work file on disk for buffering and internal merging. If the sort is performed using multi-tasking, the user has to set up at least 2, and not more than 9 such work files on disk. Work files must be created as PAM files.

With simple merge runs, no work files are required.

Creating work files

The user can create work files using the ASSIGN-FILES statement or via a CREATE-FILE-/ADD-FILE-LINK command or FILE macro. Work files can also be created by SORT itself.
When this is done, the block size calculated for the work file is always rounded up to a multiple of STD(2) (any block size entered by the user is ignored).
The coded character set is of no relevance to work files. Entries made by the user are ignored.

When setting up work files with the ADD-FILE-LINK command or FILE macro, the user must specify one or more file link names. For one work file the file link name is SORTWK, for multiple files SORTWKx, where “x” stands for a digit from 1 to 9, assigned consecutively in ascending numerical order. Work files must be present during the sort/merge run and may not be deleted by the user until after the run.
If the file assignment is made via the ASSIGN-FILES statement, the file link names are assigned by SORT.

SORT calculates the primary and secondary allocations for work files on the basis of one of the following entries, in the order of precedence given:

– Size of the input files (for disk files)
– DISK-SPACE specification (ASSIGN-RESOURCES statement)
  Eight times the internal block length is taken as the minimum size.
– RECORDS-PER-CYCLE specification (SORT-RECORDS statement)
– ESTIMATED-RECORDS specifications (SORT-RECORDS statement) divided by the number of auxiliary files
– NUMBER-OF-RECORDS specification of the INPUT-RANGE operand (SORT-RECORDS statement) divided by the number of auxiliary files
– MEMORY-SIZE specification * 16.

The primary and secondary allocations of user-defined work files are increased if they are less than the values calculated by SORT. Their values can be output via the user exit PLANNING. SORT makes no correction to the values specified by the user if a value of zero was specified for the secondary allocation.

A work file that is created by SORT is cataloged under the file link name SORTWK and the file name

```
SORTWORK.tsn.yymmdd.hhmmss
```

Key:

tsn	Task sequence number (TSN) for sort run
yy	Year
mm	Month
dd	Day
hhmmss	Six-digit time-of-day entry

If the operand LINK-PREFIX-CHANGE=<name 1..4> is given in the SET-SORT-OPTIONS statement, the “SORTWORK” part of the name is replaced by “<name 1..4>WORK”.

Allocation of storage space for work files

The size of work files (in PAM pages) required by SORT can be calculated as follows:

$$\text{File size} = \frac{1.1 * \text{PAM pages of sort data}}{(\text{number of auxiliary files} + 1)}$$

Notes

– If a work file is set up with a secondary allocation = 0, then a factor of 1.2 should be substituted for 1.1 in the above formula in order to provide an adequate margin for contingencies.

  With very large files, (for example, greater than 32 GB), the factor of 1.2 should be selected as well, since the secondary allocation (maximum of 32767 PAM pages) is negligible compared to the file size.

– For a user working with the PLANNING user exit and specifying the action DIALOG, SORT calculates the size of file required and outputs its estimate via SORT message SRT1031.

Sorting time and throughput improvements can be achieved by the user by

– allocating sufficient storage space or specifying accurate values for the ESTIMATED-RECORDS or RECORDS-PER-CYCLE operands in order to avoid frequent secondary allocations
– setting up the work files on separate private volumes
– not setting up the work files on the volumes used for the input/output files.

Close processing of work files

– Work files set up by SORT are closed, released and deleted at the end of the sort run if this terminates normally. If the sort run ends abnormally and at least one checkpoint has been written, SORT does not release or delete these files, as otherwise no RESTART-PROGRAM would be possible. If no RESTART-PROGRAM is initiated, the user is responsible for deleting the file. If a RESTART-PROGRAM is performed and the run terminates normally, SORT signs off the file, releases it, and deletes it.
– Work files set up by the user are closed at the end of the sort run. The file link name is not released, however. The specification DELETE-WORK-FILES=*YES for the SET-SORT-OPTIONS statement allows the user to specify that SORT is also to delete the work files that are created by the user or the requesting program. By default SORT does not delete these files.

Notes

Work files with the file link name SORTWKx (’x’ stands for a value from 1-9) can only be used in multi-task sorting (see page 271). Different SORTWKx cannot be used to supply parallel sort runs with different work files when the SORT access method SORTZM is used (see page 220).

3.3.1 PAM key elimination for work files

Work files on tape will be created as BLOCK-CONTROL-INFO=*NO, even if *WITHIN-DATA-BLOCK has been specified by the user.

3.4 Auxiliary files

Auxiliary files are required by SORT whenever large amounts of data are to be processed in a cycle sort, i.e. the data is divided into subsets for separate sorting. This is the case with cycle and multi-task sorts. SORT requires one auxiliary file for each subset except the last, which remains in the work file. This produces the following formula:

Number of auxiliary files = Number of cycles - 1

Auxiliary files may be disk or tape files (exception: disk files only for multi-task sorts). SORT writes and reads these files sequentially (SAM). It may therefore be possible to speed up data throughput by setting up the files on separate volumes. Up to 99 auxiliary files are permitted.

With simple merge runs no auxiliary files are required.

Creating auxiliary files

The user can create auxiliary files on disk using the ASSIGN-FILES statement or CREATE-FILE-/ADD-FILE-LINK command or FILE macro. Auxiliary files on tape must be assigned via the ADD-FILE-LINK command. The coded character set is of no relevance to auxiliary files. Entries made by the user are ignored.

The auxiliary files are assigned to SORT via the file link name SORTWKxx, where “xx” must be numbered consecutively from 01 through 99 (max.) in strict ascending sequence. The user should always set up auxiliary files explicitly when there are large amounts of data to be sorted and checkpoints to be written, or when a multi-task sort run is to be performed.

If the user has made specifications that allow SORT to determine the number of cycles (for example, ESTIMATED-RECORDS and RECORDS-PER-CYCLE in the SORT-RECORDS statement), SORT is able to create the required number of auxiliary files itself.

If this information is missing, SORT creates 8 auxiliary files as required. If the user has already created some of these auxiliary files, then SORT will finish creating the rest. If there are not enough auxiliary files, or an overflow occurs for a work file, SORT will create a single, additional final disk auxiliary file.

Creation of this final auxiliary file is carried out regardless of whether the other auxiliary files have been created by SORT or by the user. However, if the number of files has already reached 99, then the final auxiliary file will not be created. The final auxiliary file can also accomodate exactly a cycle. All remaining records are left in the work file and must be processed there.

The final auxiliary file occupies PUBLIC disk space by default.

The user can prevent this (in some circumstances unwanted) use of PUBLIC disk space by doing the following:

– The final auxiliary file can be set up by the user themselves. This is done by assigning the file link name SORTWKEX.
– This file may be located on a private disk.
– The space for this file may be limited by a small primary allocation and the specification of 0 for the secondary allocation. In this case, an excessive number of records leads to SORT being aborted. This process ensures that work and auxiliary files are not able to occupy all of the PUBLIC disk space.

If the space available in the final auxiliary file is used up, the sort run is aborted with the message SRT1060.

Auxiliary files are cataloged under the file link name SORTWKxx. SORT generates the file name as follows:

```
SORTWKxx.tsn.yymmdd.hhmmss
```

Key:

xx	File link name sequence number
tsn	Task sequence number (TSN) for sort run
yy	Year
mm	Month
dd	Day
hhmmss	Six-digit time-of-day entry

If the operand LINK-PREFIX-CHANGE=<name 1..4> is given in the SET-SORT-OPTIONS statement, the “SORTWKxx” part of the name is replaced by “<name 1..4>WKxx”.

Tape auxiliary files set up by SORT

Tape files can be assigned via ASSIGN-FILES statements only if a catalog entry already exists. If the user has already set up auxiliary files on tape using the ADD-FILE-LINK command/FILE macro, SORT increases their number to that specified in the TAPE-UNITS operand of the ASSIGN-RESOURCES statement.

Note

With auxiliary files on tape, the RECORDS-PER-CYCLE record set should not exceed the capacity of the shortest tape (or shortest tape series, as applicable).

Disk auxiliary files set up by SORT

SORT sets up auxiliary files on disk using the space occupied by the work file as the value for the primary allocation.

In the case of auxiliary files set up by the user, the primary and secondary allocations are increased if they are less than the values calculated by SORT. SORT makes no correction to the values specified by the user if a value of zero was specified for the secondary allocation.

Close processing of auxiliary files

– Auxiliary files set up by SORT are signed off, released and deleted at the end of the sort run when this terminates normally. If the sort run terminates with an error but checkpoints have already been written, SORT does not release or delete the files, as otherwise no RESTART-PROGRAM would be possible. A user choosing not to initiate a RESTART-PROGRAM becomes responsible for deleting the files. If a RESTART-PROGRAM is performed and the run terminates normally, SORT closes the files, releases them, and deletes them.

– Auxiliary files set up by the user are closed at the end of the sort run, but the file link name is not released. Tape files are rewound but not unloaded.

  By specifying DELETE-WORK-FILES=*YES in the SET-SORT-OPTIONS statement the user can invoke SORT to delete the auxiliary files that are set up by SORT or the calling program. By default SORT does not delete those files.

Note

Sorting time and throughput improvements can be achieved by the user when using auxiliary files on disk by

– allocating sufficient storage space or specifying accurate values for the ESTIMATED-RECORDS or RECORDS-PER-CYCLE operands in order to avoid frequent secondary allocations

– setting up the auxiliary files on separate private volumes, thereby enabling SORT to process the files sequentially without having to delay processing for multiple files due to the time taken to reposition disk arms

– not setting up the auxiliary files on the volumes used for the input/output files.

3.4.1 PAM key elimination for auxiliary files

Auxiliary files are SAM files on disk or tape. They will be created on disk by SORT using BLOCK-CONTROL-INFO=*PAMKEY/*WITHIN-DATA-BLOCK, depending on the class 2 option BLKCTRL = PAMKEY/NONKEY. With BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK it is important to remember that the maximum length of records that can be processed by SORT is reduced by 8 bytes to 32751 bytes.

Auxiliary files on tape will be created as BLOCK-CONTROL-INFO=*NO, even if *WITHIN-DATA-BLOCK has been specified by the user.

3.5 Checkpoint file

SORT requires a checkpoint file to be available for writing checkpoints. This file can be on disk (not NK4 disks) or on tape, and can be created by the user or by SORT. If an error is detected in writing a checkpoint, message SRT1042 is displayed and processing is resumed.

A checkpoint enables an aborted sort/merge run to be restarted by means of the RESTART-PROGRAM command.

– The user can create a checkpoint file using an ASSIGN-FILES statement or CREATE-FILE-/ADD-FILE-LINK command or FILE macro with the file link name SORTCKPT. The coded character set is of no relevance to checkpoint files. Entries made by the user are ignored.
By specifying OPEN-MODE=*INOUT in the ADD-FILE-LINK command or OPEN=INOUT in the FILE macro it is possible to continue using an existing checkpoint file. Disk files should be set up with an adequate space allocation. The minimum size for the primary allocation is estimated as follows:

((MEMORY-SIZE value * 4) + 80)

The optimum value for the primary allocation is calculated as follows:

((MEMORY-SIZE value * 4) + 80) * number of checkpoints

The secondary allocation should correspond to one checkpoint, i.e.:

((MEMORY-SIZE value * 4) + 80)

Unless the checkpoint file is to be on a private volume, there is no need for the user to create such a file, since SORT automatically sets up a checkpoint file on a public volume.

– If the user does not specify a checkpoint file, SORT sets up a disk file with the file link name SORTCKPT and assigns it the file name

```
SORTCKPT.tsn.yymmdd.hhmmss
```

Key:

tsn	Task sequence number (TSN) for sort run
yy	Year
mm	Month
dd	Day
hhmmss	Six-digit time of day entry

If the operand LINK-PREFIX-CHANGE=<name 1..4> is given in the SET-SORT-OPTIONS statement, the “SORTCKPT” part of the name is replaced by “<name 1..4>CKPT”.

Close processing of checkpoint files

– Checkpoint files set up by the user are closed at the end of the sort run, but the file link name is not released. Tapes are rewound but not unloaded.

– Checkpoint files set up by SORT are closed, released and deleted if the sort/merge run terminates normally. The same applies if the sort/merge terminates abnormally and no checkpoints have been written. If a run terminates with an error and checkpoints have been taken, SORT does not release or delete the checkpoint file, as this is required for a RESTART-PROGRAM. A user choosing not to perform a RESTART-PROGRAM is responsible for deleting the checkpoint file.

Note

When processing POSIX files as the input files, it is not possible to set up checkpoints.

3.5.1 PAM key elimination for checkpoint files

Checkpoint files are PAM files which SORT sets up using BLOCK-CONTROL-INFO=*NO.

Only BLOCK-CONTROL-INFO=*PAMKEY/*NO may be specified for checkpoint files. Specifying BLKCTRL=*WITHIN-DATA-BLOCK causes return code X’48’ to be generated during checkpoint writing and the run is terminated abnormally.

3.6 Object module library SORTMODS

An additional object module library can be defined with the file link name SORTMODS for the purpose of supporting user-defined routines that use SORT user exits. Priority will then be given to this object module library when user routines are loaded. The SORTMODS object module library can also be assigned by means of the ASSIGN-FILES statement.

The SET-SORT-OPTIONS statement can be used to replace the “SORTMODS” part of the name with “<name 1..4>MODS”.

3.7 Statement files

At any point during statement input, statements grouped into so-called “statement files” can be inserted into the input. Statement files are subject to all the rules and conventions applicable to BS2000 procedure files.

Up to 10 statement files can be assigned using the ASSIGN-FILES statement (STATEMENT-FILES operand). They are processed immediately after the ASSIGN-FILES statement in the order in which they occur. A requirement is, however, that the ASSIGN-FILES statement is read from SYSCMD.

Multiple ASSIGN-FILES statements may be specified. In this case statements originating from statement files of previous ASSIGN-FILES statements remain valid until overwritten by subsequent identically-named statements. Statement files cannot be nested, i.e. other statement file assignments within a statement file are ignored.

Each statement file must be concluded with an END statement. The END statement causes SYSDTA to be assigned to the next statement file or reassigned to SYSCMD.

If an error occurs during the assignment of SYSDTA to a statement file (e.g. statement file is found to be a PAM file), the run is aborted and an error message is issued.

If a syntax error occurs in a statement file and is not corrected (e.g. PROCEDURE-DIALOG=*NO), the remaining statement files will be processed and then the run will be aborted.

The CCSN of the statement files is evaluated when the statements are read in. Character sequences in the form of constants in the SORT statements are converted by SORT to the CCS of the data records before these constants are inserted in the output record or compared with fields in the input record.

Statement files with an ISO-CCSN are rejected with SRT1149 and statement files with Unicode-CCSN are rejected with SRT1150.

3.8 Close processing for SORT files

The table below provides an overview of the close processing for SORT files following normal termination of the sort/merge run:

Function of file(s)	File link name	File set up by	File status following the sort/merge run				
			The sort/merge program has			Files set up on magnetic tape: Tapes are	
			closed the file(s)	deleted the file(s)	released the file link name	rewound	unloaded
Sort input file(s)	SORTIN SORTINxx	user	yes	no	yes	yes	yes [1] no [2]
Merge input files	MERGExx	user	yes	no	yes	yes	yes [1] no [2]
Output file	SORTOUT	user	yes	no	no	yes	no
Work file(s)	SORTWK SORTWKx	user	yes	no	no	-	-
		SORT	yes	yes	yes	-	-
		SORT	yes	yes	yes	yes	no
Checkpoint file	SORTCKPT	user	yes	no	no	yes	no
		SORT	yes	yes	yes	no	no
Object module library	SORTMODS	user	-	-	no	-	-

[1] If KEEP-INPUT-TAPE=*NO is set in the SET-SORT-OPTIONS statement
[2] If KEEP-INPUT-TAPE=*YES is set in the SET-SORT-OPTIONS statement

Following normal termination of the sort/merge run, SORT closes all the sort files (CLOSE). The file link names of the input files, and of the work, auxiliary and checkpoint files set up by SORT, are automatically released (RELEASE). After normal completion of the sort/merge run the files set up by SORT are deleted (ERASE).

Input files on tape are rewound and unloaded if the KEEP-INPUT-TAPE operand was set to NO (UNLOAD). Output files and auxiliary files on tape, together with checkpoint files set up by the user, are rewound but not unloaded.

The table below provides an overview of the close processing for SORT files following abnormal termination of the sort/merge run:

Function of file(s)	File link name	File set up by	File status following the sort/merge run				
			The sort/merge program has			Files set up on magnetic tape: Tapes are	
			closed the file(s)	deleted the file(s)	released the file link name	rewound	unloaded
Sort input file(s)	SORTIN SORTINxx	user	yes	no	yes	yes	yes [1] no [2]
Merge input files	MERGExx	user	yes	no	yes	yes	yes [1] no [2]
Output file	SORTOUT	user	yes	no	no	yes	no
Work file(s)	SORTWK SORTWKx	user	yes	no	no	-	-
		SORT	yes	yes [*]	yes [*]	-	-
		SORT	yes	yes [*]	yes [*]	yes	no
Checkpoint file	SORTCKPT	user	yes	no	no	yes	no
		SORT	yes	yes [*]	yes [*]	no	no
Object module library	SORTMODS	user	-	-	no	-	-

[*] Only if no checkpoints were written.
[1] If KEEP-INPUT-TAPE=*NO is set in the SET-SORT-OPTIONS statement
[2] If KEEP-INPUT-TAPE=*YES is set in the SET-SORT-OPTIONS statement

Following abnormal termination of a sort/merge run, SORT closes all the files (CLOSE). The file link names of the work, auxiliary and checkpoint files set up by SORT are released (RELEASE) if no checkpoints have been written. In this case these files are also automatically deleted.

Input files on tape are rewound and unloaded if the KEEP-INPUT-TAPE operand was set to *NO (UNLOAD). Output files and auxiliary files on tape, together with checkpoint files set up by the user, are rewound but not unloaded.

3.9 Processing POSIX files with SORT

In order to be able to use the file processing functions to handle POSIX files in BS2000, the POSIX subsystem must be activated.

3.9.1 POSIX in BS2000

In view of the increasing networking of different computer systems and the distributed processing performed across these networks, computer systems and their interfaces nowadays need to be standardized and open. These interfaces must comply with the POSIX/XPG4 standards. The operating system BS2000 supports the POSIX/XPG4 standards with the software product “POSIX”.

POSIX (**P**ortable **O**pen **S**ystem **I**nterface for UNI**X**) or XPG4 (**X**/Open **P**ortability **G**uide Version **4**) denotes a range of UNIX-based standards. The name POSIX refers both to these standards and to the software product.

Through the software product POSIX, BS2000 becomes an open system. Applications which comply with the standard can be ported between BS2000 and other systems which support POSIX interfaces, especially UNIX.

The POSIX file system is a file system in BS2000 with the structure of a UNIX file system (UFS). It is hierarchically structured and consists of files (POSIX files) and directories. POSIX users can create and edit POSIX files, and can access remote UNIX file systems from within the POSIX file system. By the same token, the local POSIX file system can be accessed from a remote UNIX system.

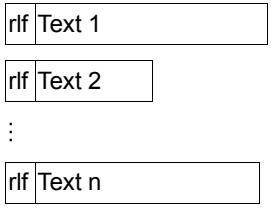
POSIX can be accessed by BS2000 users with the appropriate authorization. From a UNIX system too (via rlogin or emulation), POSIX can be accessed on a BS2000 system. The access control is handled completely by BS2000.

For further information on POSIX in BS2000, see the manuals “POSIX Basics for Users and System Administrators” [14] and “POSIX Commands” [15].

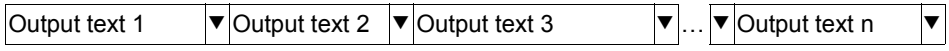
3.9.2 Sorting POSIX files with SORT

Within POSIX files, data is in text format, which cannot be processed directly by SORT. In this format, the individual records are delimited by end-of-record identifiers, which are represented by the ▼ symbol in the following graphics.

Before it is processed by the sort routine, this data is converted by SORT into variable-length records, each prefixed by a record length field (rlf).



After the sort process, SORT converts the sorted output file back into the text format if it is to be stored in the POSIX file system.



The internal use of variable-length records causes the position of the user data in the record to be displaced by the record length field, but this does not normally have any consequence for users of POSIX files. With records from POSIX files, SORT calculates the field positions by default relative to the beginning of the user data.

However, if the user wants to access the internal record length field, e.g. in order to sort the records according to their length, the IGNORE-LENGTH-FIELD operand is available in the SET-SORT-OPTIONS statement (see page 166) and in the SORT-FILE command (see page 194).
Specifying IGNORE-LENGTH-FIELD=*NO causes the positions within the record to be calculated from the start of the record, both with variable-length records in BS2000 files and with records in POSIX files. The user data thus begins at position 5 in the record.

The encoding of the end-of-record identifier is determined by the CODE operand in the ASSIGN-FILES statement (see page 139) and in the SORT-FILE command (see page 194). If CODE=*EBCDIC is specified, the end-of-record identifier is encoded as X’15’, if CODE=*ASCII is specified, as X’0A’.

If POSIX files are used as output files, you must make sure that the output records do not contain characters which might be interpreted as end-of-record identifiers. More specifically:

– No constant fields containing end-of-record identifiers may be specified in the SORT-RECORDS statement or in the SORT-FILE command.

– The records of a BS2000 input file must not contain end-of-record identifiers if the output file is to be a POSIX file.

– The sort type “tag sort” must not be used because it cannot be guaranteed that the address fields do not contain characters which could be interpreted as end-of-record identifiers.

Note

Using the “sort” command, which can be called in a POSIX shell, is not the same as calling the product SORT.

3.10 SORT and ACS

If the subsystem ACS (alias catalog service) is used, the following effects should be noted:

1. If a TFT entry created by the user exists, it is impossible to say whether or not the file name was replaced. Consequently, if the file name taken from this entry is used (e.g. with SHOW-FILE-ATTRIBUTES), ACS is interrupted (HOLD-ALIAS-SUBSTITUTION).
2. If SORT itself creates a TFT entry with the FILE macro, ACS remains in the same state as when SORT is started, i.e. file names can be replaced. There is therefore no difference with regard to file names between the TFT entries of the user and those of SORT.
3. If a user specifies a file name in an ASSIGN-FILES statement and a TFT entry already exists, he or she must consider whether the file name has been replaced or not. The file name in the statement will in any case be used without replacements for the file name comparison.
4. If the user changes the state of ACS between creation of the TFT entries and the start of SORT (e.g. through delayed loading, interruption, etc.), it may be unclear which file names are actually used in the end.

3.11 Working with files larger than 32 GB

Disk storage capacities are continuously increasing in size, and the increasing volume of data resources that must be maintained online have been taken into consideration for BS2000/OSD-BC V5.0. Previously available disk and file sizes have been increased to include volumes of 32 GB.

As of BS2000/OSD-BC V5.0:

– the maximum capacity of a single disk is approximately 4 terabytes (2.147.483.647 PAM pages)

– the maximum file size is also approximately 4 terabytes (2.147.483.647 PAM pages)

SORT V8.0 supports a file size of more than 32 GB for input, output, work and auxiliary files.

Due to the numerous possible constellations when sorting, it is possible that only one, several or all participating files exceed the 32 GB limit. In principle, SORT permits file sizes of more than 32 GB for the files used. However, the user must usually make sure that the files greater than 32 GB are created on a pubset that is designed to handle files of this type and that there is sufficient storage space available to it.

Additional information about files larger than 32 GB can be found in the “Introductory Guide to DMS” manual [2].

3.11.1 Creating files larger than 32 GB

Files where the size is permitted to exceed 32 GB should be created before a sort run using CREATE-FILE/ADD-FILE-LINK. This guarantees that a file of sufficient size is made available on a suitable pubset.

The creation of output, work and auxiliary files may, however, be left to SORT. For this purpose, the names of the files that are to be created must be entered in the ASSIGN-FILES statement along with the catalog ID (cat-id). These catalog IDs must refer to a pubset that is suitable for files ≥ 32 GB. The size of the file can only be specified explicitly for the work file. This is done using the statement

```
//ASSIGN-RESOURCES ...,DISK-SPACE=<integer 1..2147483647>...
```

You do not need to specify the file name for work and auxiliary files, as long as the default pubset permits files ≥ 32 GB. In this case, SORT creates the files on the default pubset, as required, under a name that it generates itself.

Forcing the use of files greater than 32 GB

If the system is to check before a sort run whether there is sufficient space available for the individual files, the relevant files must be made available with the appropriate SPACE specifications. Here, the value of the primary allocation is particularly important, since the value of the secondary allocation is not permitted to exceed 32767 PAM pages. This also applies to automatically calculated values for the secondary allocation, if the specification made using DISK-SPACE of the ASSIGN-RESOURCES statement specifies a larger value for the primary allocation.

When using files ≥ 32 GB, you must pay more attention to ensuring sufficient primary allocation than for smaller files because the storage space that is made available with secondary allocation is a smaller proportion of the overall volume.

3.11.2 Tag sort

In a tag sort the following restrictions apply for input files ≥ 32 GB, according to file type:

– SAM files

  When using the variants *TAG-HEADER and *TAG-TRAILER, four-byte long retrieval address fields are formed (three bytes for the number of the data block and one for the relative record number within the block). When using input files ≥ 32 GB it may be that the number of data blocks cannot be represented using just three bytes. The sort run will be rejected with the message SRT1264.

  With the variant *TAG-COMPOUND there are no restrictions.

– PAM files

  All retrieval addresses are four bytes in length. With input files ≥ 32 GB all variants of address list sorting are rejected.

– ISAM files

  There are no restrictions for input files ≥ 32 GB compared with files < 32 GB.

3.11.3 Recommendations when working with large files

As a result of the longer run times required when sorting large files, it may be sensible to set checkpoints. This naturally assumes that you will be working with either the cycle sort or multi-tasking sort method. The increased time expenditure of these methods is rewarded with a greater level of safety. Sort runs of this magnitude may take 24 hours or longer.

In order to achieve longer sequences in the presort phase (and thus shorter sort times) it is recommended that a larger MEMORY-SIZE value is entered in the ASSIGN-RESOURCES statement (e.g. 10000 or above). The optimum value here will depend on the following factors:

– Size of the real memory of the computer being used,
– Maximum value for the virtual memory of the ID,
– other users that are working in parallel to SORT.

Insufficient real memory and work being carried out in parallel may lead to increased paging. This then reduces the benefit of using longer sequences. When carrying out very long sort runs, you should try to ensure that the computer is available exclusively for SORT.

If an excessively large value is specified for MEMORY-SIZE, the run time of the SORT may be increased considerably because the pages of the virtual memory are continuously being swapped in and out.

– The size of the real memory (in MB) can be determined using
`/SHOW-SYSTEM-INFORMATION INFORMATION=MEMORY-SIZE`

– The maximum size of the virtual memory of the ID (in MB) can be determined using
`/SHOW-USER-ATTRIBUTES` (ADDRESS-SPACE-LIMIT field)

  The value for the maximum size of the virtual memory can be increased by systems support if necessary.

The MEMORY-SIZE value should be set at 70% of the smaller of the two values. MEMORY-SIZE specifies the amount of the main memory to be used in CORE pages (4096 Byte).

If the real memory is extremely small, the value for MEMORY-SIZE may exceed the size of the real memory. The size of the virtual memory may not be exceeded.

Example

Size of real memory	120 MByte
Size of virtual memory of the ID	128 MByte
Value of MEMORY-SIZE in ASSIGN-RESOURCES	20480 (or 80 MByte)

The value of MEMORY-SIZE in ASSIGN-RESOURCES is, in certain circumstances, restricted for SORT by the CORE-MAXIMUM limit value of the system. In this case, this value can be increased by specifying CORE-MAXIMUM in the statement MODIFY-SORT-DEFAULTS.

The value for CORE-MAXIMUM must be at least 1/16 of the value of MEMORY-SIZE, otherwise the MEMORY-SIZE will be decreased accordingly.

A sufficiently large value for MEMORY-SIZE is then available if there is only a single presort run and a single end merge run, but no intermediate merge run (recognizable by MIN-MSG-WEIGHT=*ALL).

In order that large MEMORY sizes can actually be used by SORT, SORT must run in the upper address space. You can ensure that this happens by specifying PROG-MODE=*ANY in the START-SORT command, or by starting SORT using the SORT-FILE command.

4 SORT statements

4.1 Input sources

When SORT is used as an independent program or as a subroutine invoked at level 0, statements can be entered at a display terminal or from a procedure or ENTER file. When SORT is used as a subroutine at level 1, the statements must be made available in virtual memory.

Each sort/merge run requires either a SORT-RECORDS or MERGE-RECORDS statement to be entered. Each definition of a sort/merge run has to be concluded with an END statement.

With the exception of the END statement and the ADD-SYMBOLIC-NAME statement, all statements in a sort/merge run may be specified in any order. The END statement must always be the last statement specified and may only occur once in any run. The ADD-SYMBOLIC-NAME statement must always come ahead of the statement in which the assigned symbolic name is used. In interactive mode, multiple statements of a single type are permitted. In procedure, batch and level 1 modes, if more than one statement of a single type occurs the SORT run will be aborted.

SORT statements are similar in format to variable-length records. The length specification is stored in the first halfword of the record length field that is prefixed to the record proper. SORT statements are read in via the system file SYSDTA. Statements entered directly at the terminal have the length specification added by the system.

When SORT is invoked as a subroutine, the SORT statements can also be passed directly to it in main memory.

4.2 SDF syntax representation

Details on the SDF metasyntax and data types and on entering commands and statements can be found in the „SDF Dialog Interface“ [7] and "Commands" [1] manuals.

4.3 Error handling

Syntax errors are handled by SDF itself. Semantic errors within a statement (e.g. conflicting operands) are dealt with as follows:

– in interactive and procedure mode with the SDF option PROCEDURE-DIALOGUE=*YES, a correction dialog is initiated.

– in procedure mode with the SDF option PROCEDURE-DIALOGUE=*NO and when running SORT as a subroutine at level 1, a branch is made to the next STEP or END statement.

– statement files are handled like procedure files.

Correction dialog sequence

Messages indicating the type of error are followed by a request issued at the terminal to correct the errored operands. This process is repeated until the statement is free of error.

4.4 Overview of the SORT statements

Operation	Scope	Old ISP statement
ADD-SYMBOLIC-NAMES	Assigns symbolic names to sort, sum, match and remainder fields, as well as to constants and masks.	
ASSIGN-EXITS	Enables user routines to be connected at defined SORT exits.	MODS
ASSIGN-FILES	Assigns input, output, work, auxiliary, checkpoint and statement files.	
ASSIGN-RESOURCES	Specifies the amount of main memory and disk storage space required, as well as the number of tape units.	ALLOC
END	End of statement input and start of sort/merge run	END
MERGE-RECORDS	Statement defining a merge run; describes the sort fields and their most significant attributes.	MERGE
MODIFY-CODE	Defines and modifies the code sorting sequence for MODIFY-CODE sort fields.	NEWCOL
MODIFY-SORT-DEFAULTS	Modifies default values.	
SELECT-INPUT-RECORDS	Enables records of the input file(s) to be selected using a logical expression.	OMIT, INCLUDE
SET-RECORD-ATTRIBUTES	Describes the input, internal and output records in terms of format, length and filler character (if any).	RECORD
SET-SORT-OPTIONS	Defines options for message output, link name handling, error handling, etc.	OPTION
SHOW-SORT-DEFAULTS	Displays default values.	
SORT-RECORDS	Statement defining a sort run; describes the sort fields and their most significant attributes.	SORT
SUM-RECORDS	Records with identical sort keys can be compacted into one record and the sum fields added together.	SUM

The SDF standard statements may be entered additionally. They are not described in the present manual. A description may be found in the manual “SDF Dialog Interface” [7].

ADD-SYMBOLIC-NAMES

ADD-SYMBOLIC-NAMES assigns symbolic names to

– sort fields
– sum fields
– match fields
– remainder fields
– constants
– print masks.

Symbolic names can be used in the following statements:

– MERGE-RECORDS
– SELECT-INPUT-RECORDS
– SORT-RECORDS
– SUM-RECORDS

Up to 255 symbolic names can be defined.

ADD-SYMBOLIC-**NAM**ES

FIELDS = ***NO** / list-poss(255): <name 1..20>(...)

<name 1..20>(...)

POSITION = <integer 1..32759>(...)

<integer 1..32759>(...)

BIT-POSITION = **0** / <integer 0..7>

,**LEN**GTH = <integer 0..32759>(...)

<integer 0..32759>(...)

NUMBER-**OF-BIT**S = **0** / <integer 0..7>

,**FOR**MAT = ***CHAR**ACTER / ***NO** / ***BI**NARY / ***FIX**ED-POINT / ***FLOAT**ING-POINT /
***PACK**ED-**DEC**IMAL / ***ZONED-DEC**IMAL / ***EBCDIC-DIN** /
***EBCDIC-INT**ERNATIONAL / ***PHY**SICAL-**TRANS**LATE / ***VIRT**UAL-**TRANSL**ATE /
***MOD**IFY-**CODE** / ***EBCDIC-ISO-EBCDIC** / ***EXT**ENDED-**CHAR**ACTER /
***TRAN**SLATE-**CHAR**ACTER / ***UNICODE-CHAR**ACTER

,**CONST**ANTS = ***NO** / list-poss(255): <name 1..20>(...)

<name 1..20>(...)

VALUE = <integer -2147483639..2147483639> / <c-string 1..256 with-low> /
<x-string 1..512>

,**PRINT-MASK**S = ***NO** / list-poss(255): <name 1..20>(...)

<name 1..20>(...)

FORM = <c-string 1..254 with-low>

FIELDS =
Up to 255 symbolic names may be defined for sort, sum, match and remainder fields in respect of position, length and format for the statements SORT-RECORDS, MERGE-RECORDS, SUM-RECORDS and SELECT-INPUT-RECORDS.

FIELDS = *NO
No symbolic names are defined for fields.

FIELDS = list-poss(255): <name 1..20(...)>
Symbolic name for the field.

POSITION = <integer 1..32759(...)>
Position of the field relative to start of record. The specified position must be within the permitted bounds. These are indicated in the description of the POSITION operand of the statement for which the field is defined.

BIT-POSITION = 0 / <integer_0..7>
Position of the binary field relative to start of field. BIT-POSITION may be specified only for sort fields in BINARY format.

LENGTH = <integer 0..32759(...)>
Length of the field. The specified length must be within the permitted bounds. These are indicated in the description of the LENGTH operand of the statement for which the field is defined.

NUMBER-OF-BITS = 0 / <integer 0..7>
Length in bits, specified in addition to the length in bytes. NUMBER-OF-BITS may be specified only for sort fields in BINARY format.

FORMAT = *CHARACTER / *NO / *BINARY / *FIXED-POINT / *FLOATING-POINT / *PACKED-DECIMAL / *ZONED-DECIMAL / *EBCDIC-DIN / *EBCDIC-INTERNATIONAL/ *PHYSICAL-TRANSLATE / *VIRTUAL-TRANSLATE / *MODIFY-CODE / *EBCDIC-ISO-EBCDIC / *EXTENDED-CHARACTER / *TRANSLATE-CHARACTER / *UNICODE-CHARACTER
Format of the sort field (for attributes see page 38). The format specified must be valid for the statement for which the field is defined. The entry FORMAT=*NO is only effective when used in conjunction with remainder fields (see SORT-RECORDS statement, page 182).

CONSTANTS =
Symbolic names of the constants for the SORT-RECORDS and SELECT-INPUT-RECORDS statements. The combined length of all specified constants and print masks must not be greater than 5000 bytes.

CONSTANTS = *NO
No symbolic names are defined for constants.

CONSTANTS = list-poss(255): <name 1..20(...)>
Symbolic names of the constants.

VALUE =
Value of the constant. Permitted values are indicated in the description of the statement for which the constant is defined.

VALUE = <integer-2147483639..2147483639>
Decimal constant. Converted by SORT into a 4-byte fixed-point number.

VALUE = <c-string 1..256 with-low>
Character constant.

VALUE = <x-string 1..512>
Hexadecimal constant.

PRINT-MASKS =
Assigns symbolic names to print masks for the SORT-RECORDS and SUM-RECORDS statements. The combined length of all specified constants and print masks must not be greater than 5000 bytes, including the extra 1-byte length field required for each mask.

PRINT-MASKS = *NO
No symbolic names are defined for print masks.

PRINT-MASKS = list-poss(255): <name 1..20(...)>
Name(s) of the print mask(s).

FORM = <c-string 1..254 with-low>
Form of the print mask. The following characters are possible:

– a freely selectable filler character as the first character of the mask
– the control characters ’#’ (number sign) and ’^’ (circumflex)
– characters to be inserted (not equal to the control characters)

ASSIGN-EXITS

ASSIGN-EXITS links the user exits provided by SORT with the associated user routines.

SORT uses modules that are loaded from libraries. The libraries first have to be assigned in one of the following ways:

- `//ASSIGN-FILES MODULE-LIBRARY=libraryname`
- `/ADD-FILE-LINK LINK-NAME=SORTMODS`
- `/ADD-FILE-LINK LINK-NAME=<BLSLIB00..BLSLIB99>`
- `/SET-TASKLIB LIBRARY=libraryname`

SORT looks for the user routines using the following procedure:

- If a library is assigned using //ASSIGN-FILES MODULE-LIBRARY=... (see page 139) or with the file link name SORTMODS, SORT attempts to load the routine from this library.

  If SORT is unable to locate the requested user routine in this library, it will then look in the libraries with the file link names BLSLIB00 - BLSLIB99 in ascending order (00, 01, .., 99). The numbers do not have to be contiguous.

- If no library is assigned with the file link name SORTMODS or using //ASSIGN-FILES MODULE-LIBRARY=..., then the system looks for a library made available using the SET-TASKLIB command.

ASSIGN-**EXITS**

PLANNING = ***NO** / ***DIALOG** / ***TERM**INATE-**ABNORM**AL

,**INPUT** = ***NO** / ***MOD**ULE(...)

***MOD**ULE(...)

NAME = <name 1..8>

,**PAR**AMETER-**MODE** = **24** / ***ANY**

,**OUTPUT** = ***NO** / ***MOD**ULE(...)

***MOD**ULE(...)

NAME = <name 1..8>

,**PAR**AMETER-**MODE** = **24** / ***ANY**

,**EXLST-F**OR-**IN**PUT = ***NO** / ***MOD**ULE(...)

***MOD**ULE(...)

NAME = <name 1..8>

,**EXLST-F**OR-**OUT**PUT = ***NO** / ***MOD**ULE(...)

***MOD**ULE(...)

NAME = <name 1..8>

,**WORK-FIL-OV**ERFLOW = ***NO** / ***DIALOG** / ***FIN**ISH-**INPUT** / ***TERM**INATE-**ABNORM**AL / ***MOD**ULE(...)

***MOD**ULE(...)

NAME = <name 1..8>

,**INT**ERFACE-VERSION = **1** / <integer 1..2>

,**PHY**SICAL-**TRANS**LATE = ***NO** / ***MOD**ULE(...)

***MOD**ULE(...)

NAME = <name 1..8>

,**VIRT**UAL-**TRANSL**ATE = ***NO** / ***MOD**ULE(...)

***MOD**ULE(...)

NAME = <name 1..8>

,**EXT**ERNAL-**COMP**ARE = ***NO** / ***MOD**ULE(...)

***MOD**ULE(...)

NAME = <name 1..8>

,**PAR**AMETER-**MODE** = **24** / ***ANY**

,**TRAN**SLATE-**CHAR**ACTER = ***NO** / ***MOD**ULE(...)

***MOD**ULE(...)

NAME = <name 1..8>

PLANNING =
Activated after the planning phase has been completed and the sort strategy selected.

PLANNING = *NO
The exit is not activated.

PLANNING = *DIALOG
SORT outputs the estimated size of the work file (in PAM pages) and the number of requested CORE pages. After this, the following actions are allowed:

CONTINUE	Processing is continued.
START	The SORT run is restarted. Only the modified statements need to be reentered. START is permitted only for independent SORT runs and when SORT is invoked as a subroutine via level 0.
TERMINATE	The SORT run is terminated.

In procedure or batch mode, if no action is specified, processing resumes with CONTINUE.

PLANNING = *TERMINATE-ABNORMAL
The SORT run is terminated.

INPUT =
Activated when SORT accepts a record from the input source. The record can be checked, modified or deleted. It is also possible to insert new records. The user routine can also be responsible for the entire input.

INPUT = *NO
The exit is not activated.

INPUT = MODULE(...)
Specifies a user module to be linked in and invoked by SORT.

NAME = <name 1..8>
Name of the user module.

PARAMETER-MODE =
Defines the interface between SORT and the user module and the addressing mode in which the user module executes. For a user module which can run only in 24-bit addressing mode, PARAMETER-MODE=24 must be set and the corresponding interface used.

PARAMETER-MODE = 24
The user module can run in 24-bit addressing mode only.

PARAMETER-MODE = *ANY
The user module can run in 24-bit or 31-bit addressing mode.

OUTPUT =
Activated before SORT writes to the output file. The record can be checked, modified or deleted. It is also possible to insert new records. The user routine can also be responsible for the entire output.

OUTPUT = *NO
The exit is not activated.

OUTPUT = *MODULE(...)
Specifies a user module to be linked in and invoked by SORT.

NAME = <name 1..8>
Name of the user module.

PARAMETER-MODE =
Defines the interface between SORT and the user module and the addressing mode in which the user module executes. For a user module which can run only in 24-bit addressing mode, PARAMETER-MODE=24 must be set and the corresponding interface used.

PARAMETER-MODE = 24
The user module can run in 24-bit addressing mode only.

PARAMETER-MODE = *ANY
The user module can run in 24-bit or 31-bit addressing mode.

EXLST-FOR-INPUT =
Specifies EXLST exits for the input file(s).

EXLST-FOR-INPUT = *NO
The exit is not activated.

EXLST-FOR-INPUT = *MODULE(...)
Specifies a user module to be linked in by SORT.

NAME = <name 1..8>
Name of the user module.

EXLST-FOR-OUTPUT =
Specifies EXLST exits for the output file.

EXLST-FOR-OUTPUT = *NO
The exit is not activated.

EXLST-FOR-OUTPUT = *MODULE(...)
Specifies a user module to be linked in by SORT.

NAME = <name 1..8>
Name of the user module.

WORK-FILE-OVERFLOW =
Activated if the overflow of a disk work file with a secondary allocation = 0 is imminent, SORT cannot remove the bottleneck and no auxiliary file is available for a further cycle.

WORK-FILE-OVERFLOW = *NO
The exit is not activated.

WORK-FILE-OVERFLOW = *DIALOG
The number of records accepted by SORT up to this point is displayed and one of the following actions is expected:

CONTINUE	SORT attempts to perform the SORT run with fewer reserves.
FINISH	SORT terminates record input and processes the records already accepted.
TERMINATE	SORT terminates the run.

WORK-FILE-OVERFLOW = *FINISH-INPUT
Record input is terminated and SORT processes the records read thus far.

WORK-FILE-OVERFLOW = *TERMINATE-ABNORMAL
The sort/merge run is terminated.

WORK-FILE-OVERFLOW = *MODULE(...)
Specifies a user module to be linked in and invoked by SORT.

NAME = <name 1..8>
Name of the user module.

INTERFACE-VERSION =
Version of the interface that SORT is to use when calling the exit module (see section “WORK-FILE-OVERFLOW” on page 249).

INTERFACE-VERSION = 1
SORT uses the old interface with 4-byte record counters. Up to 2.14.483.639 records can be processed.

INTERFACE-VERSION = 2
SORT uses the new interface with 8-byte record counters. Up to 9.223.372.036.854.775.807 - 8 records can be processed.

PHYSICAL-TRANSLATE =
Connects a user routine consisting of two code tables (of 256 characters each). The code tables determine the sorting sequence for fields having the format PHYSICAL-TRANSLATE. The first table is used to convert the sort fields after input, the second to reconvert them prior to output (see chapter “SORT user exits” on page 233).

PHYSICAL-TRANSLATE = *NO
The exit is not activated.

PHYSICAL-TRANSLATE = *MODULE(...)
Specifies a user module to be linked in by SORT.

NAME = <name 1..8>
Name of the user module.

VIRTUAL-TRANSLATE =
Connects a user routine consisting of one code table (of 256 characters). The code table determines the sorting sequence for fields having the format VIRTUAL-TRANSLATE. Before the comparisons are made, the sort field is converted into an auxiliary field. The sort field itself is not modified (see chapter “SORT user exits” on page 233).

VIRTUAL-TRANSLATE = *NO
The exit is not activated.

VIRTUAL-TRANSLATE = *MODULE(...)
Specifies a user module to be linked in by SORT.

NAME = <name 1..8>
Name of the user module.

EXTERNAL-COMPARE =
This exit is activated for each record comparison for fields defined with the sorting order EXTERNAL-COMPARE. The order can then be determined by the user module.

EXTERNAL-COMPARE = *NO
The exit is not activated.

EXTERNAL-COMPARE = *MODULE(...)
Specifies a user module to be linked in by SORT.

NAME = <name 1..8>
Name of the user module.

PARAMETER-MODE =
Defines the interface between SORT and the user module and the addressing mode in which the user module executes. For a user module which can run only in 24-bit addressing mode, PARAMETER-MODE=24 must be set and the corresponding interface used.

PARAMETER-MODE = 24
The user module can run in 24-bit addressing mode only.

PARAMETER-MODE = *ANY
The user module can run in 24-bit or 31-bit addressing mode.

TRANSLATE-CHARACTER =
Connects a user routine consisting of two code tables (of 256 characters each). The code table determines the sorting sequence for fields having the format TRANSLATE-CHARACTER when the CCSN of the input file matches the name of the specified module (see the chapter “SORT user exits” on page 233).

TRANSLATE-CHARACTER = *NO
The exit is not activated.

TRANSLATE-CHARACTER = *MODULE(...)
Specifies a user module to be linked in by SORT.

NAME = <name 1..8>
Name of the user module.

ASSIGN-FILES

ASSIGN-FILES assigns input, output, work, auxiliary, checkpoint and statement files, as well as a library containing user modules. This statement is used in combination with the START-SORT command.

ASSIGN-**FIL**ES

INPUT-FILES = ***LINK** / list-poss(99): <filename 1..54> / <posix-pathname 1..1023>

,**OUTPUT-FIL**E = ***LINK** / <filename 1..54> / <posix-pathname 1..1023>

,**WORK-FIL**ES = ***STD** / list-poss(9): <filename 1..54>

,**AUXIL**IARY-**FIL**ES = ***STD** / list-poss(99): <filename 1..54>

,**CHECK**POINT-**FIL**E = ***STD** / <filename 1..54>

,**MOD**ULE-**LIB**RARY = ***STD** / <filename 1..54>

,**STAT**EMENT-**FIL**ES = ***NONE** / list-poss(10): <filename 1..54>

,**CODE** = ***E**BCDIC / ***A**SCII

INPUT-FILES =
Assignment of input file(s) (max. 99).

INPUT-FILES = *LINK
The input files are assigned via ADD-FILE-LINK commands (file link name SORTIN, SORTINxx or MERGExx).
By default the file link name is released again when SORT terminates. If the LOCK-FILE-LINK command is specified before SORT is invoked, the file link name is retained.

INPUT-FILES = list-poss(99): <filename 1..54> / <posix-pathname 1..1023>
Name(s) of the input file(s). Input files assigned using a ADD-FILE-LINK command or a preceding ASSIGN-FILES statement are ignored.
POSIX file names must be specified in single quotes to distinguish them from BS2000 file names.
POSIX input files and BS2000 input files must not be used simultaneously in a sort run. If one input file is a POSIX file, all other input files must also be POSIX files.

Restriction:
The sum of the number and lengths of all specified POSIX input file names must not be greater than 5100. If any specifications exceed this, an error message is issued and SORT is aborted.

OUTPUT-FILE =
Output file assignment.

OUTPUT-FILE = *LINK
The output file is assigned by means of a ADD-FILE-LINK command (file link name SORTOUT).

OUTPUT-FILE = <filename 1..54> / <posix-pathname 1..1023>
Name of the output file. An output file assigned via a ADD-FILE-LINK command or a preceding ASSIGN-FILES statement is ignored. Different file attributes than those taken from the input file or specified by SORT can only be assigned via a ADD-FILE-LINK command.
The file link name SORTOUT is used, and is retained after the sort run is completed.
A POSIX file name must be specified in single quotes to distinguish it from a BS2000 file name.

WORK-FILES =
Assignment of the work file(s) (max. 9).

WORK-FILES = *STD
The work files are assigned via a ADD-FILE-LINK command (file link name SORTWK or SORTWKx) or are created by SORT (file name: SORTWKx.tsn.yymmdd.hhmmss).

WORK-FILES = list-poss(9): <filename 1..54>
Name(s) of the work file(s). Work files assigned via the ADD-FILE-LINK command or a preceding ASSIGN-FILES statement are ignored. The files and file link names are retained after the end of the sort run.

AUXILIARY-FILES =
Assignment of the auxiliary file(s) (max. 99).

AUXILIARY-FILES = *STD
The auxiliary files are assigned via ADD-FILE-LINK commands (file link name SORTWKxx) or are created by SORT (file name SORTWKxx.tsn.yymmdd.hhmmss).

AUXILIARY-FILES = list-poss(99): <filename 1..54>
Name(s) of the auxiliary file(s). Auxiliary files assigned via ADD-FILE-LINK commands or a preceding ASSIGN-FILES statement are ignored. The files and file link name are retained after the end of the sort run.

CHECKPOINT-FILE =
Assignment of a checkpoint file.

CHECKPOINT-FILE = *STD
The checkpoint file is assigned via a ADD-FILE-LINK command (file link name SORTCKPT) or is created by SORT (file name: SORTCKPT.tsn.yymmdd.hhmmss).

CHECKPOINT-FILE = <filename 1..54>
Name of the checkpoint file. A checkpoint file assigned via a ADD-FILE-LINK command or a preceding ASSIGN-FILES statement is ignored. The file and file link name are retained after the end of the sort run.

MODULE-LIBRARY =
Assignment of an object module library containing user modules. SORT loads user modules from this library.

MODULE-LIBRARY = *STD
The object module library is assigned via a ADD-FILE-LINK command (file link name SORTMODS). The file link name is retained after the end of the sort run.

MODULE-LIBRARY = <filename 1..54>
Name of the object module library. A library assigned via a ADD-FILE-LINK command or a preceding ASSIGN-FILES statement is ignored.

STATEMENT-FILES =
Assignment of one or more files (max. 10) containing SORT statements (see section “Statement files” on page 114).

STATEMENT-FILES = *NONE
No statement files are specified.

STATEMENT-FILES = list-poss(10): <filename 1..54>
Name(s) of the statement file(s).

CODE =
Specification of the code of POSIX files. This applies to both the input file and the output file and merely defines the encoding of the end-of-record identifier. In particular, this operand has **no** effect on the sort sequence. For BS2000 files the operand has no significance and is ignored.

CODE = *EBCDIC
The end-of-record identifier is encoded according to EBCDIC and has the value X’15’.

CODE = *ASCII
The end-of-record identifier is encoded according to ASCII and has the value X’0A’.

ASSIGN-RESOURCES

ASSIGN-RESOURCES defines the size of the main memory and disk storage to be used by SORT, as well as the number of tape units for auxiliary files. The specifications made in ASSIGN-RESOURCES take precedence over the values determined by SORT.

ASSIGN-**RES**OURCES
MEMORY-**SIZE** = ***STD** / ***MIN** / ***SMALL** / ***LARGE** / ***MAX** / <integer 1..500000> ,**DISK-SPACE** = ***BY-CALC**ULATION / <integer 1..2147483647> ,**TAPE-UNITS** = ***NONE** / <integer 1..99>

MEMORY-SIZE =
Specifies the size of virtual memory for the presorting areas and the internal input/output buffers. If this operand is not specified, SORT calculates a value from other explicit or implicit specifications for the sort/merge run. If no data is available on which to base an estimate, SORT uses the value which can be defined as the default value with the CORE-DEFAULT operand of the MODIFY-SORT-DEFAULTS statement. The upper and lower limits can also be defined with the MODIFY-SORT-DEFAULTS statement.

MEMORY-SIZE = *STD
The value is calculated by SORT from the specifications for the sort/merge run. If SORT cannot calculate this value because the necessary specifications are missing, the preset default value is assumed. This default can be defined specific to the ID or to the system by means of the CORE-DEFAULT operand of the MODIFY-SORT-DEFAULTS statement (factory presetting = 40).

MEMORY-SIZE = *MIN
Minimum value. Corresponds to the value that can be defined ID- or system-specifically with the CORE-MINIMUM operand of the MODIFY-SORT-DEFAULTS statement (factory presetting: SORT = 24). If this CORE value is too small (e.g. record length = 32759), it is increased by SORT.

MEMORY-SIZE = *SMALL
Lower value, corresponds to the larger of the following values:
– value calculated by SORT minus 33% or
– specification in the CORE-MINIMUM operand of the MODIFY-SORT-DEFAULTS statement.

MEMORY-SIZE = *LARGE
Higher value, corresponds to the smaller of the following values:
– value calculated by SORT plus 33% or
– specification in the CORE-MAXIMUM operand of the MODIFY-SORT-DEFAULTS statement.

MEMORY-SIZE = *MAX
Maximum value. Value which can be defined ID- or system-specifically with the CORE-LIMIT operand of the MODIFY-SORT-DEFAULTS statement (factory presetting = 96).

MEMORY-SIZE = <integer 1..500000>
Size of the desired memory area in CORE pages (4K). If a value greater than 32767 is specified, this value is converted to megabytes (division by 256), with the result that the actual CORE value can be up to 255 CORE pages less than the specified value.

DISK-SPACE =
Size, in PAM pages, of a disk work file to be set up by SORT.

DISK-SPACE = *BY-CALCULATION
The size of the work file is calculated by SORT from the entries for ESTIMATED-RECORDS and RECORDS-PER-CYCLE (in the SORT-RECORDS statement). If this is not possible, SORT uses the value for MEMORY-SIZE multiplied by 16.

DISK-SPACE = <integer 1..2147483647>
Size of the work file in PAM pages. If the user has set up one or more work files independently, this value is ignored.

TAPE-UNITS =
Specifies auxiliary tape units to be used for cycle output during cycle sorting. If the values given for ESTIMATED-RECORDS and RECORDS-PER-CYCLE (in the SORT-RECORDS statement) result in the number of cycles being greater than the number of auxiliary files assigned via file link names, SORT sets up additional auxiliary files.

TAPE-UNITS = *NONE
No tape units are made available for auxiliary files.

TAPE-UNITS = <integer 1..99>
Number of tape units available for auxiliary files.

END

Function: Completes the entry of a SORT statement and initiates the sort/merge run. The END statement is always the last statement entered and may only be used once.

END

This statement has no operands.

MERGE-RECORDS

MERGE-RECORDS defines sort fields and their most significant attributes (position, length,...) for a merge run. A merge run can take place only if all of the input files are already sorted according to the specified sort keys.

A merge run using MERGE-RECORDS should be performed when files have already been sorted according to a common criterion, since then there is no compute-intensive sorting phase.

MERGE-**REC**ORDS

FIELDS = ***COMPL**ETE-**REC**ORD / list-poss(64): ***FIE**LD-**EXPL**ICIT(...) / ***FIE**LD-**SYM**BOLIC(...)

***FIE**LD-**EXPL**ICIT(...)

POSITION = <integer 1..32759>(...)

<integer 1..32759>(...)

BIT-POSITION = **0** / <integer 0..7>

,**LEN**GTH = <integer 0..32759>(...)

<integer 0..32759>(...)

NUMBER-**OF**-**BIT**S = **0** / <integer 0..7>

,**FOR**MAT = ***CHAR**ACTER / ***BI**NARY / ***FIX**ED-POINT / ***FLOAT**ING-POINT /
***PACK**ED-**DEC**IMAL / ***ZONED-DEC**IMAL / ***EBCDIC-DIN** /
***EBCDIC-INT**ERNATIONAL / ***PHY**SICAL-**TRANS**LATE / ***VIRT**UAL-**TRANSL**ATE /
***MOD**IFY-**CODE** / ***EBCDIC-ISO-EBCDIC** / ***EXT**ENDED-**CHAR**ACTER /
***TRAN**SLATE-**CHAR**ACTER / ***UNICODE-CHAR**ACTER

,**SORT**ING-**ORD**ER = ***ASC**ENDING / ***DES**CENDING / ***EXT**ERNAL-**COMP**ARE

,**PRIO**RITY = ***CUR**RENT-**NUMB**ER / <integer 1..64>

,**ELIM**INATE = ***NO** / ***Y**ES

,**TWO-DIGIT-YEAR** = ***NO** / ***Y**ES

***FIE**LD-**SYM**BOLIC(...)

SORT-FIELD-**NAM**E = list-poss(64): <name 1..20>

,**SORT**ING-**ORD**ER = ***ASC**ENDING / ***DES**CENDING / ***EXT**ERNAL-**COMP**ARE

,**PRIO**RITY = ***CUR**RENT-**NUMB**ER / <integer 1..64>

,**ELIM**INATE = ***NO** / ***Y**ES

,**TWO-DIGIT-YEAR** = ***NO** / ***Y**ES

,**REC**ORDS-**PER-CYC**LE = ***NO** / <integer 1..2147483639>

,**CHECK**POINT = ***NO** / ***Y**ES

FIELDS =
Specifies the sort fields that determine the merge sequence. At least one sort field must be specified; up to 64 may be specified.

FIELDS = *COMPLETE-RECORD
With fixed record format, the whole record is used as a sort field. With variable record format, the record length field is not included in the record comparison. For the purposes of record comparison, the variable part of the record is padded out to the maximum record length with the filler character defined in the FILLER operand of the SET-RECORD-ATTRIBUTES statement (default value X'00').

FIELDS = *FIELD-EXPLICIT(...)
Specifies sort fields in terms of position, length and format.

POSITION = <integer 1..32759(...)>
Position of the sort field relative to start of record. In fixed-length records the first data field occupies position 1. In records with variable record format, the position of the first data field depends on the file type and the IGNORE-LENGTH-FIELD operand of the SET-SORT-OPTIONS statement. This dependency is shown in the following table:

IGNORE-LENGTH-FIELD=	*STD	*YES	*NO
BS2000 file	5	1	5
POSIX file	1	1	5

For all formats except CHARACTER, EBCDIC-DIN, EBCDIC-INTERNATIONAL, EXTENDED-CHARACTER and TRANSLATE-CHARACTER the position specifications for sort fields must be in the range 1 to 4096.

BIT-POSITION = 0 / <integer 0..7>
Position specified in bits, in addition to the byte position entry. Specifying BIT-POSITION is allowed only for sort fields in BINARY format.

LENGTH = <integer 0..32759(...)>
Length of the sort field. The length must be within the limits allowed for the format. Specifying BIT-POSITION and NUMBER-OF-BITS increases the length of the sort field by 1 byte (BIT-POSITION + NUMBER-OF-BITS $\leq$ 7) or by 2 bytes (BIT-POSITION + NUMBER-OF-BITS > 7). This must be taken into account when specifying the maximum length. With variable-length records sort fields in CHARACTER, EBCDIC-DIN and EBCDIC-INTERNATIONAL format are allowed to extend into the variable part of the record.

NUMBER-OF-BITS = 0 / <integer 0..7>
Length specified in bits, in addition to the length in bytes. Specifying NUMBER-OF-BITS is allowed only for sort fields in BINARY format.

FORMAT = *CHARACTER / *BINARY / *FIXED-POINT / *FLOATING-POINT / *PACKED-DECIMAL / *ZONED-DECIMAL / *EBCDIC-DIN / *EBCDIC-INTERNATIONAL / *PHYSICAL-TRANSLATE / *VIRTUAL-TRANSLATE / *MODIFY-CODE / *EBCDIC-ISO-EBCDIC / *EXTENDED-CHARACTER / *TRANSLATE-CHARACTER / *UNICODE-CHARACTER
Format of the sort field (for attributes, see page 38).

SORTING-ORDER =
Order in which SORT is to arrange the records.

SORTING-ORDER =*ASCENDING
Ascending sorting order.

SORTING-ORDER = *DESCENDING
Descending sorting order. This option is not allowed for ISAM output files.

SORTING-ORDER = *EXTERNAL-COMPARE
The order for this sort field is defined by the user via user exit EXTERNAL-COMPARE. This sort field must not be more than 255 bytes long.

PRIORITY =
Priority (weight) of the sort field. If several sort fields are specified, those with a priority of 1 are compared first. If they are identical, the sort fields with a priority of 2 are compared. This process is repeated until the sort fields compare unequal or until those with the highest priority have been compared.

PRIORITY = *CURRENT-NUMBER
The priority is determined by the order in which the sort fields are specified (the first sort field has a priority of 1).

PRIORITY = <integer 1..64>
Specifies the priority number. If a priority number is specified for one sort field, it must also be specified for all other sort fields. All the priority numbers must form a consecutive ascending sequence beginning with 1.

ELIMINATE =
Specifies whether the sort field is to be eliminated (not included in the output).

ELIMINATE = *NO
The sort field is not to be eliminated.

ELIMINATE =* YES
The sort field is to be eliminated. It serves solely to determine the sorting order. This option is not allowed for binary fields with bit specifications; if specified, it will be ignored and SORT will issue a warning. When the OUTPUT user exit is used (with PARAMETER-MODE = 24), identical records are not displayed.

TWO-DIGIT-YEAR =
Determines whether the sort field contains a two-digit year number.

TWO-DIGIT-YEAR = *NO
The sort field does not contain a two-digit year number.

TWO-DIGIT-YEAR = *YES
The sort field contains a two-digit year number. This statement is legal only in combination with FORMAT=*PACKED-DECIMAL or FORMAT=*ZONED-DECIMAL. In this case, the sorting sequence depends on the CENTURY-WINDOW-SHIFT operand in the SET-SORT-OPTIONS statement.

The following sort field formats are possible:

FORMAT=	Sort field format
*PACKED-DECIMAL	X'0yyv' and X'0jjmmddv'
*ZONED-DECIMAL	X'Fyvy'

With: y = one digit of the two-digit year specification, mm = month, dd = day,
s = sign (must be positive)

FIELDS = *FIELD-SYMBOLIC(...)
Specifies sort fields using symbolic names which must previously have been defined as fields by means of the ADD-SYMBOLIC-NAMES statement.

SORT-FIELD-NAME = list-poss(64): <name 1..20>
Name of a sort field (max. 64 fields). If a list of symbolic names is given, the SORTING-ORDER and ELIMINATE entries apply to all names; the value given for PRIORITY is used for the first name specified, and a value incremented by 1 each time is entered for the remaining names.

SORTING-ORDER =
Order in which SORT is to arrange the records.

SORTING-ORDER = *ASCENDING
Ascending sorting order.

SORTING-ORDER = *DESCENDING
Descending sorting order. This option is not allowed for ISAM output files.

SORTING-ORDER = *EXTERNAL-COMPARE
The order for this sort field is defined by the user via user exit EXTERNAL-COMPARE. This sort field must not be more than 255 bytes long.

PRIORITY =
Priority (weight) of the sort field. If several sort fields are specified, those with a priority of 1 are compared first. If they are identical, the sort fields with a priority of 2 are compared. This process is repeated until the sort fields compare unequal or until those with the highest priority have been compared.

PRIORITY = *CURRENT-NUMBER
The priority is determined by the order in which the sort fields are specified (the first sort field has a priority of 1).

PRIORITY = <integer 1..64>
Specifies the priority number. If a priority number is specified for one sort field, it must also be specified for all other sort fields. All the priority numbers must form a consecutive ascending sequence beginning with 1.

ELIMINATE =
Specifies whether the sort field is to be eliminated (not included in the output).

ELIMINATE = *NO
The sort field is not to be eliminated.

ELIMINATE = *YES
The sort field is to be eliminated. It serves solely to determine the sorting order. This option is not allowed for binary fields with bit specifications; if specified, it will be ignored and SORT will issue a warning. If the OUTPUT user exit is used (with PARAMETER-MODE=24), identical records are not displayed.

TWO-DIGIT-YEAR =
Determines whether the sort field contains a two-digit year number.

TWO-DIGIT-YEAR = *NO
The sort field does not contain a two-digit year number.

TWO-DIGIT-YEAR = *YES
The sort field contains a two-digit year number. This statement is legal only in combination with FORMAT=*PACKED-DECIMAL or FORMAT=*ZONED-DECIMAL. In this case, the sorting sequence depends on the CENTURY-WINDOW-SHIFT operand in the SET-SORT-OPTIONS statement.

The following sort field formats are possible:

FORMAT=	Sort field format
*PACKED-DECIMAL	X’0yyv’ and X’0jjmmddv’
*ZONED-DECIMAL	X’Fyvy’

With: y = one digit of the two-digit year specification, mm = month, dd = day,
s = sign (must be positive)

RECORDS-PER-CYCLE =
Number of records per checkpoint cycle. The value specified determines the number of records after which SORT is to write a checkpoint. Specifying a value for this operand only makes sense if CHECKPOINT=*YES is also given.

RECORDS-PER-CYCLE = *NO
No checkpoints are written.

RECORDS-PER-CYCLE = <integer 1..2147483639>
Number of records after which a checkpoint is to be written.

CHECKPOINT =
Controls the output of checkpoints.

CHECKPOINT = *NO
SORT writes no checkpoints.

CHECKPOINT = *YES
SORT writes a checkpoint after the number of records specified in the RECORDS-PER-CYCLE operand.

MODIFY-CODE

MODIFY-CODE changes the sorting order determined by the EBCDIC code into a user-defined sequence. This statement is required if a field has been defined using FORMAT=*MODIFY-CODE.

MODIFY-**CODE**
SEQUENCES = list-poss(256): <c-string 2..2 with-low> / <x-string 3..4>

SEQUENCES =
Specifies all characters that are to be sorted in a different sequence than that given by the EBCDIC code. A list of character pairs of the form ’cq’ (character) or X’cq’ (hexadecimal) is used to cause each occurrence of the character ’q’ to be moved in the sorting order to a position immediately after the character ’c’. Up to 256 character pairs can be specified.

SEQUENCES = list-poss(256): <c-string 2..2 with-low>
Specifies a character pair in character representation. Example: MODIFY-CODE SEQUENCES = (’aA’,’bB’). This causes ’A’ to be placed immediately after ’a’ in the sorting order, and ’B’ immediately after ’b’.

SEQUENCES = list-poss(256): <x-string 3..4>
Specifies a character pair in hexadecimal representation. Example: MODIFY-CODE SEQUENCES = (X’81C1’). This causes X’C1’ (=’A’) to be placed immediately after X’81’ (=’a’) in the sorting order.

If a character is to be moved to a position in the sorting order after a character that has already been converted, then the new position applies, not the original EBCDIC position.

If a character (e.g. X’F8’) is to be moved to the first position in the sorting order, then 2 character pairs have to be specified:

MODIFY-CODE SEQUENCES = (X’00F8’,X’F800’)

The first character pair causes X’F8’ to be placed immediately after X’00’, and the second causes X’00’ to be placed immediately after X’F8’, thereby moving X’F8’ to the beginning of the sorting order.

MODIFY-SORT-DEFAULTS

MODIFY-SORT-DEFAULTS defines or modifies default values for SORT parameters. The SORT parameters can be preset specific to the system or to the ID. The validity of the defaults can be restricted to the current SORT run or extended to cover all subsequent SORT runs.

MODIFY-**SORT-DEF**AULTS
MIN-MSG-WEIGHT = ***UNCHA**NGED / ***STD** / ***ALL** / ***NORM**AL / ***CRIT**ICAL / ***NONE** ,**CORE-MIN**IMUM = ***UNCHA**NGED / ***STD** / <integer 24..32767> ,**CORE-MAX**IMUM = ***UNCHA**NGED / ***STD** / <integer 24..32767> ,**CORE-LIM**IT = ***UNCHA**NGED / ***STD** / <integer 24..32767> ,**CORE-DEF**AULT = ***UNCHA**NGED / ***STD** / <integer 24..32767> ,**SUBTASK-JOB-CLASS** = ***UNCHA**NGED / ***STD** / <name 1..8> ,**SAVE-DEF**AULTS = ***NO** / ***Y**ES

MIN-MSG-WEIGHT =
Default for the minimum message weight to be output.
This specifies the priority level as of which messages are to be output. Messages are then output if their priority is greater than or equal to the specified value.

MIN-MSG-WEIGHT = *UNCHANGED
The value remains unchanged (see note on page 155).

MIN-MSG-WEIGHT = *STD
The default value is used (*NORMAL).

MIN-MSG-WEIGHT = *NORMAL
With autonomous sort/merge runs, messages with a priority of 2 or above are output. With SORT as a subroutine, messages with a priority of 3 or above are output.

MIN-MSG-WEIGHT = *ALL
All messages are output (as of priority 0).

MIN-MSG-WEIGHT = *CRITICAL
Messages with a priority of 3 or above are output.

MIN-MSG-WEIGHT = *NONE
Messages with a priority of 7 or above are output (only messages concerning internal errors).

Note
If a SET-SORT-OPTIONS statement exists in the sort run, the specification or the default value as set for MIN-MSG-WEIGHT in the SET-SORT-OPTIONS statement will take priority.

CORE-MINIMUM =
Lower limit for the CORE value calculation by SORT.

CORE-MINIMUM = *UNCHANGED
The value remains unchanged (see note on page 155).

CORE-MINIMUM = *STD
The default value is used (24).

CORE-MINIMUM = <integer 24...32767>
The specified value is used.

CORE-MAXIMUM =
Maximum value for the intensively used virtual memory. If a value greater than 400 is specified in the ASSIGN-RESOURCES statement, only a sixteenth of this memory is used intensively. If the intensively used memory exceeds the value specified here, the value specified in the ASSIGN-RESOURCES statement is reduced.

CORE-MAXIMUM = *UNCHANGED
The value remains unchanged (see note on page 155).

CORE-MAXIMUM = *STD
The default value is used (4096).

CORE-MAXIMUM = <integer 24...32767>
The specified value is used.

Note
If the specified value does not exceed the minimum value specified using the CORE-MINIMUM operand, then the maximum value is the same as the minimum value.

CORE-LIMIT =
Upper limit for the CORE value calculation by SORT.

CORE-LIMIT = *UNCHANGED
The value remains unchanged (see note on page 155).

CORE-LIMIT = *STD
The default value is used (96).

CORE-LIMIT = <integer 24...32767>
The specified value is used.

Note
The specified value must not be lower than the value defined by the CORE-MINIMUM operand or higher than the value defined by the CORE-MAXIMUM operand. If either of these situations is the case, the limit value used is the value defined in the corresponding operand.

CORE-DEFAULT =
CORE value if any specifications are missing.

CORE-DEFAULT = *UNCHANGED
The value remains unchanged (see note on page 155).

CORE-DEFAULT = *STD
The default value is used (40).

CORE-DEFAULT = <integer 24...32767>
The specified value is used.

Note
The specified value must not be lower than the value defined by the CORE-MINIMUM operand or higher than the value defined by the CORE-MAXIMUM operand. If either of these situations is the case, the limit value used is the value defined in the corresponding operand.

SUBTASK-JOB-CLASS =
Job class to be used for the start of a subtask.

SUBTASK-JOB-CLASS = *UNCHANGED
The value remains unchanged (see note on page 155).

SUBTASK-JOB-CLASS = *STD
The default value is used (*STD).

SUBTASK-JOB-CLASS = <name 1...8>
The specified job class is used.

SAVE-DEFAULTS =
Specifies the period of validity for the modified defaults.

SAVE-DEFAULTS = *NO
Defaults changed for this SORT run only.

SAVE-DEFAULTS = *YES
Defaults changed for all subsequent SORT runs.
The result is stored in the parameter file.
If the user ID under which the command is entered matches the one under which the central parameter file is installed, the new defaults apply to the whole system. Otherwise, they apply only to the ID.

Note

The operand value *UNCHANGED is interpreted as follows:
If in a previous MODIFY-SORT-DEFAULTS statement of the same SORT run a value other than *UNCHANGED was specified, this value applies. Otherwise, the value is taken from the ID-specific parameter file or, if this does not yet exist, from the central parameter file.
When the first MODIFY-SORT-DEFAULTS statement is executed, the ID-specific parameter file is stored under an ID with SAVE-DEFAULTS=*YES, and is modified at each subsequent execution of this statement.
The central parameter file is created when the product SORT is installed.
System-specific modification of the default value is only possible from the ID under which SORT is installed, because the central parameter file is stored there too.

You can view the current defaults using the SHOW-SORT-DEFAULTS statement.

SELECT-INPUT-RECORDS

SELECT-INPUT-RECORDS enables the user to select those records from the input files that are to be sorted by SORT. Selection is on the basis of a logical expression (logically combined comparisons).

SELECT-**INPUT-REC**ORDS
CONDITION = <text 0..1800 with-low>

CONDITION = <text 0..1800 with-low>
The CONDITION operand must specify the condition under which an input record is included in the sort operation. The condition may be formed from one or more comparison relations joined by the logical operators AND or OR. Eight levels of nesting are possible using parentheses. Up to 64 relations may be specified.

Comparison relations are specified in the following form:

[NOT] (rel1, { AND / OR } ,rel2 ...)

If **NOT** is specified, all records that do **not** meet the condition are included in the sorting operation.

The **relations** rel1,rel2... have the following format:

rel = field1, rop, { field2 / constant }

The **match fields** field1,field2 have the following format:

field = { (start, length, format) / symbolic name }

Start:
The start of the match field must be specified here. The value may only be given in bytes, even for binary data format. With fixed record format, the first data byte occupies position 1; in records with variable record format, this position depends on the file type and the IGNORE-LENGTH-FIELD operand of the SET-SORT-OPTIONS statement. The position of the first data field in these cases is shown in the following table:

IGNORE-LENGTH-FIELD=	*STD	*YES	*NO
BS2000 file	5	1	5
POSIX file	1	1	5

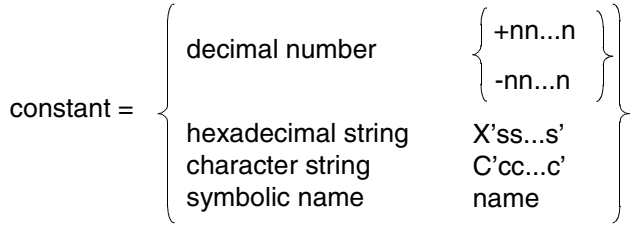
Length:
The length of the match field must be specified here (in bytes). The maximum permitted lengths are dependent on the data format. See the following table.

Format:
The following formats and lengths are permitted for match fields:

Format	**Format description**	**Length in bytes**
BI	Binary	1 - 256
CH	Character	1 - 256
FI	Fixed-point	1 - 256
PD	Packed decimal	1 - 16
ZD	Zoned decimal	1 - 16

Symbolic name:
This name must previously have been defined as a field in terms of position, length and format by means of the ADD-SYMBOLIC-NAMES statement. If the name has not already been so defined, SORT issues an error message. A symbolic name may be up to 20 characters long.

Constants which can be specified in place of the second match field field2 take the following form:



The combined length of all constants must not be greater than 4000 bytes.

Decimal constants must be specified with an arithmetic sign. If the first match field has FIXED-POINT format, the constant must not have a value greater than +2147483647 or less than -2147483648.

Hexadecimal and **character constants** must not be more than 256 bytes long.

Symbolic name:
This name must previously have been defined as a constant with the ADD-SYMBOLIC-NAMES statement. If the name has not already been so defined, SORT issues an error message. A symbolic name may be up to 20 characters long. The rules regarding maximum length, matching of constants and format compatibility are the same as for direct specification of the constant value.

Relational operator rop:
The following relational operators are permitted:

Relational operator	Replacement character	Meaning
EQ	=	equal to
LT	<	less than
GT	>	greater than
NE	<>	not equal to
GE	>=	greater than or equal to
LE	<=	less than or equal to

The relations can be combined using AND or OR. If desired, parentheses can be used to reverse the normal precedence of AND over OR.

Permissible combinations of match fields and match constants:

The format of the first (or only) match field must be compatible with that of the second or with the type of the match constant. The combinations allowed are shown in the table below.

1st match field	2nd match field					Match constant		
	BI	CH	FI	PD	ZD	Decimal number	Hexa-decimal string	Character string
BI	+	+	-	-	-	-	+	+
CH	+	+	-	-	-	-	+	+
FI	-	-	+	-	-	+	-	-
PD	-	-	-	+	+	+	-	-
ZD	-	-	-	+	+	+	-	-

Key:

+ This format combination is permitted.

- This format combination is not permitted and causes SORT to issue an error message.

When comparing match fields of different lengths, SORT adjusts the shorter field to the same length as the longer one. The following table provides an overview of how the shorter field can be padded:

Format	Data is stored	Padding on the	Filler character
BI	left-justified	right	Zero
CH	left-justified	right	X' 40' (space)
FI	right-justified	left	Sign
PD	right-justified	left	Zero
ZD	right-justified	left	X' F0' (zero)

When a match field is compared with a constant, the constant is adjusted to the length of the match field. Numerical constants are lengthened or shortened to or from the left, character constants to or from the right. If this adjustment causes the match constant to be truncated by a significant amount, i.e. an amount not equal to the corresponding padding, SORT issues a warning message.

SET-RECORD-ATTRIBUTES

SET-RECORD-ATTRIBUTES defines the format for input and output records, and also for internal records. The user must specify this statement if SORT cannot obtain this information from other statements or cannot calculate it (e.g. no input file if records are read solely via the INPUT user exit) or if record lengths or formats are to be changed.

```
SET-RECORD-ATTRIBUTES

 INPUT = *STD / *VARIABLE(...) / *FIXED(...)
    *VARIABLE(...)
        | MAXIMUM-RECORD-SIZE = *STD / <integer 1..32759>
    *FIXED(...)
        | RECORD-SIZE = *STD / <integer 1..32759>
,INTERNAL = *BY-CALCULATION / *VARIABLE(...) / *FIXED(...)
    *VARIABLE(...)
        | MAXIMUM-RECORD-SIZE = *BY-CALCULATION / <integer 1..32759>
        | ,MINIMUM-RECORD-SIZE = *BY-CALCULATION / <integer 1..32759>
        | ,AVERAGE-RECORD-SIZE = *BY-CALCULATION / <integer 1..32759>
    *FIXED(...)
        | RECORD-SIZE = *BY-CALCULATION / <integer 1..32759>
,OUTPUT = *STD / *VARIABLE(...) / *FIXED(...)
    *VARIABLE(...)
        | MAXIMUM-RECORD-SIZE = *STD / <integer 1..32759>
    *FIXED(...)
        | RECORD-SIZE = *STD / <integer 1..32759>
,FILLER = *NIL / <c-string 1..1 with-low> / <x-string 1..2>
,CODED-CHARACTER-SET = *NOT-SPECIFIED / <name 1..8>
,UNICODE-FILLER = *BLANK / <x-string 4..4>
```

INPUT =
Describes the input records in terms of record format and length.

INPUT = *STD
SORT attempts to obtain the record format and length from a preceding ADD-FILE-LINK command or from the catalog entry. If the record format cannot be ascertained, the format of the internal record is used. If that is not possible, fixed record format is assumed. If the record length cannot be ascertained, the maximum possible record length, derived from the BUFFER-LENGTH specification, is used as a replacement value. If that is not possible, 2048 is assumed.

INPUT = *VARIABLE(...)
The input records have variable record format. If the specified value does not match the value taken from a preceding ADD-FILE-LINK command or the catalog entry, SORT issues a warning and ignores the specified value.

MAXIMUM-RECORD-SIZE =
Maximum length of the input records.

MAXIMUM-RECORD-SIZE = *STD
The record length is taken from a preceding ADD-FILE-LINK command or from the catalog entry. If that is not possible, the maximum possible record length, derived from the BUFFER-LENGTH specification, is used as a replacement value. If that is not possible, 2048 is assumed.

MAXIMUM-RECORD-SIZE = <integer 1..32759>
Specifies the maximum input record length. SORT uses the maximum value from the record lengths of all the input files and the length specified here. If the specified value is less than the maximum value from the record lengths of all the input files, SORT issues a warning.

INPUT = *FIXED(...)
The input records have fixed record format. If the specified value does not match the value taken from a preceding ADD-FILE-LINK command or the catalog entry, SORT issues a warning and ignores the specified value.

RECORD-SIZE =
Length of the input records.

RECORD-SIZE = *STD
SORT attempts to obtain the record length from a previously specified ADD-FILE-LINK command or from the catalog entry. If the record length cannot be ascertained, the BUFFER-LENGTH is used. If that is not possible, 2048 is assumed.

RECORD-SIZE = <integer 1..32759>
Length of the input records. If the specified value does not match the value taken from a preceding ADD-FILE-LINK command or from the catalog entry, SORT issues a warning and ignores the specified value.

INTERNAL =
Describes the internal records. This only needs to be specified if the record format or length is to be changed compared with the input records in the INPUT user exit or by SORT. If the INPUT user exit is specified, it is responsible for making the changes; otherwise the record format or length is adjusted by SORT.

INTERNAL = *BY-CALCULATION
The record format and length are the same as for the input records. If the input record format is not specified, fixed record format is taken as the replacement value.

INTERNAL = *VARIABLE(...)
The internal records are to have variable record format. If the input records are in fixed format and the adjustment is to be carried out by SORT, a record length field (4 bytes) is prefixed to the internal records.

MAXIMUM-RECORD-SIZE =
Maximum length of the internal records.

MAXIMUM-RECORD-SIZE = *BY-CALCULATION
With input records in variable record format, the maximum internal record length is equal to the maximum input record length. With input records in fixed record format, the maximum internal record length is equal to the maximum input record length + 4.

MAXIMUM-RECORD-SIZE = <integer 1..32759>
Specifies the maximum length of the internal records. Fixed-length input records are truncated if their length is greater than the internal length minus 4; variable-length input records are truncated if they are longer than the specified length.

MINIMUM-RECORD-SIZE =
Minimum length of the internal records. Used for capacity calculations; should be specified only if known.

MINIMUM-RECORD-SIZE = *BY-CALCULATION
Calculated by SORT.

MINIMUM-RECORD-SIZE = <integer 1..32759>
Specifies the minimum record length.

AVERAGE-RECORD-SIZE =
Average length of the internal records. Used for capacity calculations; should be specified only if known.

AVERAGE-RECORD-SIZE = *BY-CALCULATION
Calculated by SORT.

AVERAGE-RECORD-SIZE = <integer 1..32759>
Specifies the average record length.

INTERNAL = *FIXED(...)
The internal records are to have fixed record format. If the input records have variable record format and the adjustment is carried out by SORT, the record length field (4 bytes) of the input records is truncated.

RECORD-SIZE =
Length of the internal records.

RECORD-SIZE = *BY-CALCULATION
With input records in fixed record format, the internal record length is equal to the input record length. With input records in variable record format, the internal record length is equal to the maximum input record length minus 4.

RECORD-SIZE = <integer 1..32759>
Specifies the internal record length. If the adjustment is carried out by SORT, the input records are truncated if longer than the specified length or padded with the character specified in the FILLER operand if shorter.

OUTPUT =
Describes the output records. This only needs to be specified if the record format or length is to be changed compared with the internal records in the OUTPUT user exit or by SORT. If the OUTPUT user exit is specified, it is responsible for making the changes; otherwise the record format or length is adjusted by SORT. If the specified record format does not match the RECORD-FORMAT operand in a ADD-FILE-LINK command or the format given in the catalog entry for the output file, the sort run is abnormally terminated. If the RECORD-FORMAT entry is missing, a suitable value is supplied by SORT.

OUTPUT = *STD
The record format and length are as for the internal records, taking into account the eliminated sort fields and the print masks.

OUTPUT = *VARIABLE(...)
The output records are to have variable record format. If the internal records have fixed record format and the adjustment is carried out by SORT, a record length field (4 bytes) is prefixed to the internal records.

MAXIMUM-RECORD-SIZE =
Maximum record length of the output records. Inserted for RECORD-SIZE in the catalog entry for the output file.

MAXIMUM-RECORD-SIZE = *STD
With internal records in variable record format, the maximum output record length is equal to the maximum internal record length. With internal records in fixed record format, the maximum output record length is equal to the internal record length + 4.

MAXIMUM-RECORD-SIZE = <integer 1..32759>
Specifies the maximum output record length. Fixed-length internal records are truncated if their length is greater than the specified length minus 4; variable-length internal records are truncated if they are longer than the specified length.

OUTPUT = *FIXED(...)
The output records are to have fixed record format. If the internal records are in variable record format and the adjustment is to be carried out by SORT, the record length field (4 bytes) of the internal records is truncated.

RECORD-SIZE =
Length of the output records. If the specified length does not match the RECORD-SIZE operand of a preceding ADD-FILE-LINK command or the value in the catalog entry for the output file, the sort run is abnormally terminated.

RECORD-SIZE = *STD
With internal records in fixed record format, the output record length is equal to the internal record length. With internal records in variable record format, the output record length is equal to the maximum internal record length minus 4. Sort fields that are to be eliminated and print masks are additionally taken into account.

RECORD-SIZE = <integer 1..32759>
Specifies the output record length. If the adjustment is to be carried out by SORT, the internal records are truncated if they are longer than the specified length; if shorter, they are padded out to the specified length with the character specified in the FILLER operand.

FILLER =
Specifies a 1-byte filler character to be used for padding records in the event of a change of format (V → F) or record length (internal record length > input record length or output record length > internal record length).

FILLER = *NIL
NUL (X'00') is used as the filler character.

FILLER = <c-string 1..1 with-low>
Character constant (1 byte).

FILLER = <x-string 2..2>
Hexadecimal constant (1 byte).

CODED-CHARACTER-SET =
Specifies the name of the coded character set for sorting the data records. This operand is only evaluated by SORT when the data records are input via the INPUT user exit or via the SORT access method. If an input file is available, this operand is ignored and the warning SRT1256 appears.

CODED-CHARACTER-SET = *NOT-SPECIFIED
No coded character set is specified. EDF03IRV code is used.

CODED-CHARACTER-SET = <name 1..8>
Specifies the name of the coded character set.

UNICODE-FILLER =
Specifies a character which is two bytes long which, when necessary, can be used to fill a sort field with the UNICODE-CHARACTER format. This is required, for example, then the field is incomplete because of variable records, or when it is shortened as a result of IGNORE-UNICODE-BLANK.

UNICODE-FILLER = *BLANK
X'0020' (Unicode blank) is used.

UNICODE-FILLER = <x-string 4..4>
The character specified is used.

SET-SORT-OPTIONS

SET-SORT-OPTIONS allows the user to define the following options for a sort/merge run:

– influence of the record length field on the position calculation for control fields

– message output

– sort sequence check

– variations on predefined file link names

– error response action

– sort and merge run optimization

– file and tape handling

– diagnostics output

– whether blanks or other variable collation elements are taken into account when the sort sequence is defined.

SET-SORT-OPTIONS

MIN-MSG-WEIGHT = ***NORM**AL / ***ALL** / ***CRIT**ICAL / ***NONE**

,**SEQ**UENCE-**CHECK** = ***Y**ES / ***NO**

,**LINK-PREFIX-CHANGE** = ***NO** / <name 1..4>

,**IGN**ORE-**INOUT-FIL**E = ***NO** / ***Y**ES

,**INPUT-OP**EN-**ERR**OR = ***CONT**INUE-**NEXT-FILE**(...) / ***FIN**ISH-**INPUT** / ***TERM**INATE-**ABNORM**AL

***CONT**INUE-**NEXT-FILE**(...)

TERMINATION = ***ABNORM**AL / ***NORM**AL

,**OPTIM**IZATION = ***RUN-TIME** / ***CPU-TIME** / ***VIRT**UAL-**MEM**ORY

,**DEST**ROY-**WORK-FIL**ES = ***NO** / ***Y**ES

,**KEEP-INPUT-TAP**ES = ***NO** / ***Y**ES

,**DUMP** = ***Y**ES / ***NO**

,**STAT**EMENT-**CCSN** = ***NOT-SPECIFIED** / <name 1..8>

,**IGN**ORE-**LEN**GTH-**FIE**LD = ***STD** / ***Y**ES / ***NO**

,**DEL**ETE-**WORK-FIL**ES = ***NO** / ***Y**ES

,**CEN**TURY-**WIN**DOW-**SHI**FT = **50**(...) / <integer 0..99> (...)

<integer 0..99>(...)

WINDOW-**BOUND**ARY = ***SUB**TRACT-**FROM-CUR**RENT-**YEAR** / ***ADD-TO-CUR**RENT-**YEAR**

,**IGN**ORE-**UNI**CODE-**BL**ANK = ***NO** / ***Y**ES

,**IGNORE-CHARACTER** = ***NONE** / < list-poss(4):<c-string 1..1 with low> / <x-string 1..2>

MIN-MSG-WEIGHT =
Specifies the message output priority. A message is output if its priority is greater than or equal to the specified message level.

MIN-MSG-WEIGHT = *NORMAL
In independent sort/merge runs, messages are output starting at priority 2 or the predefined priority. When SORT is run as a subroutine, messages with priority 3 or higher are output.

MIN-MSG-WEIGHT = *ALL
All messages are output (starting at priority 0).

MIN-MSG-WEIGHT = *CRITICAL
Messages with priority 3 or higher are output.

MIN-MSG-WEIGHT = *NONE
Messages with priority 7 or higher are output (messages concerning internal errors only).

SEQUENCE-CHECK =
This operand controls the check on ascending or descending sorting sequence prior to the final output. If the SUM-RECORDS statement is used, the sorting sequence is checked anyway, so that SEQUENCE-CHECK=*NO would be ineffective in this case. The sequence check is carried out ahead of any OUTPUT user exit specified. Thus, the records inserted or changed by means of this user exit can disrupt the sorting sequence. The SEQUENCE-CHECK operand has precedence over the corresponding flag byte in the OUTPUT user exit (only with PARAMETER-MODE=24).

SEQUENCE-CHECK = *YES
SORT checks the ascending or descending sorting sequence prior to the final output.

SEQUENCE-CHECK = *NO
The sorting sequence is not checked.

LINK-PREFIX-CHANGE =
Defines alternative prefixes for the following file link names: SORTIN / SORTINxx / MERGExx / SORTOUT / SORTWK / SORTWKx / SORTWKxx / SORTCKPT / SORTMODS. The “SORT” or “MERGE” character string in these file link names is replaced by the specified string (1 to 4 characters). This enables multiple SORT runs to be performed concurrently from a single main program. The alternative prefix is also used for generation of the names of the files set up by SORT (work, auxiliary and checkpoint files).

LINK-PREFIX-CHANGE = *NO
The prefix is not changed.

LINK-PREFIX-CHANGE = <name 1..4>
Specifies a prefix.

IGNORE-INOUT-FILE =
This operand enables the user to specify whether SORT is to ignore the file link names SORTIN / SORTINxx for the SORT input file and SORTOUT for the SORT output file. It should be used when input and output are to take place via the INPUT and OUTPUT user exits.

IGNORE-INOUT-FILE = *NO
SORT takes the file link names into account.

IGNORE-INOUT-FILE = *YES
The file link names are not taken into account.

INPUT-OPEN-ERROR =
This operand enables the user to define how SORT is to behave when there are multiple input files and one of these input files cannot be opened.

INPUT-OPEN-ERROR = *CONTINUE-NEXT-FILE(...)
SORT reports the error, skips the file concerned and processes the remaining files.

TERMINATION =

TERMINATION = *ABNORMAL
The sort/merge run is to terminate abnormally in the event of an error. In an independent sort/merge run, SORT terminates with TERM UNIT=STEP,MODE=ABNORMAL. When SORT is run as a subroutine, it stores X'FF' in the low-order byte of register 15 and the message number in the first two bytes (e.g. following open error SRT1035 register 15 will contain X'103500FF').

TERMINATION = *NORMAL
The sort/merge run is to terminate normally.

INPUT-OPEN-ERROR = *FINISH-INPUT
SORT reports the error, terminates the input phase and sorts the records read up to that point.

INPUT-OPEN-ERROR = *TERMINATE-ABNORMAL
SORT reports the error and terminates abnormally.

OPTIMIZATION =
Optimizes the sort/merge run.

OPTIMIZATION = *RUN-TIME
Runtime optimization is requested.

OPTIMIZATION = *CPU-TIME
CPU-time optimization is requested. There is currently no difference between *RUN-TIME and *CPU-TIME.

OPTIMIZATION = *VIRTUAL-MEMORY
Memory optimization is requested (e.g. by releasing load modules that are no longer needed). When SORT is run as a subroutine, all dynamically loaded modules are unloaded again at the end of the sort run (SRTXGEN, SRTXKERN and SRTXKRN1).

DESTROY-WORK-FILES =
Controls how files set up by SORT are deleted.

DESTROY-WORK-FILES = *NO
The files set up by SORT are logically deleted, i.e. only the catalog entry is deleted.

DESTROY-WORK-FILES = *YES
The files set up by SORT are logically and physically deleted, i.e. the catalog entry is deleted and the file overwritten with binary zeros. This can be used for data protection.

KEEP-INPUT-TAPES =
Controls how input tapes are unloaded.

KEEP-INPUT-TAPES = *NO
Input tapes are rewound and unloaded in order to make the tape unit available for other files.

KEEP-INPUT-TAPES = *YES
After being read, the input tapes are rewound but not unloaded; this means they can be used by another program. With multi-volume files, however, only the last tape is not unloaded.

DUMP =
Controls the output of memory contents and errored data to SYSOUT and SYSLST when internal SORT errors or application errors occur.

DUMP = *YES
Diagnostics are output.

DUMP = *NO
Diagnostics are suppressed. This value should only be used when there are compelling data protection reasons for it, because of the risk that SORT or application errors cannot be diagnosed.

STATEMENT-CCSN =
Specifies the coded character set of the character constants in the statements. This value is valid for the constants of all statements of the program run. SORT only evaluates this operand when SORT is called as a subroutine.

STATEMENT-CCSN = *NOT-SPECIFIED
No coded character set is specified. EDF03IRV code is used.

STATEMENT-CCSN = <name 1..8>
Specifies the name of the coded character set.

IGNORE-LENGTH-FIELD =
Specifies whether, in the case of variable record lengths, the record length field must be taken into account in position specifications for control fields.

The specification applies to all control fields in the current sort run. It does not affect position specifications in fixed-length records.

IGNORE-LENGTH-FIELD = *STD
With BS2000 files, *STD has the same effect as *NO, i.e. the record length field must be taken into account in position specifications. The behavior is therefore the same as in the previous versions.

With POSIX files, *STD has the same effect as *YES. The record length field created internally by SORT is therefore ignored in the position calculation.

IGNORE-LENGTH-FIELD = *YES
The record length field is ignored in the position calculation. The first data field of a record begins at position 1.

IGNORE-LENGTH-FIELD = *NO
The record length field is taken into account in the position calculation. The first data field of a record begins at position 5.

DELETE-WORK-FILES =
Indicates whether work and auxiliary files (SORTWK, SORTWKx, SORTWKxx) set up by the user are deleted by SORT at the end of the sort run.

DELETE-WORK-FILES = *NO
The auxiliary and work files are deleted at the end of the sort run only if they were created by SORT.

DELETE-WORK-FILES = *YES
At the end of the sort run, the auxiliary and work files are deleted even if they were set up by the user.

Note

If SORT aborts during a run that uses checkpoint/restart, the work and auxiliary files are maintained regardless of what has been specified for the DELETE-WORK-FILES operand, in order to enable a restart.

CENTURY-WINDOW-SHIFT = 50(...) / <integer 0..99>(...)
Determines how SORT interprets and sorts fields that contain two-digit years (FORMAT= *PACKED-DECIMAL or *ZONED-DECIMAL and TWO-DIGIT-YEAR=*YES). The CENTURY-WINDOW-SHIFT operand defines a century window within which sorting is executed (ascending or descending order). Depending on what is specified for the WINDOW-BOUNDARY operand, this century window is determined by either its start or its end date.

WINDOW-BOUNDARY =
Specifies the way the start and the end of the century window is calculated.

WINDOW-BOUNDARY = *SUBTRACT-FROM-CURRENT-YEAR
The **start** of the century window is computed as follows:

<century window start> = <current year> - <integer in century-window-shift>.

Since the century window is 100 years long, its **end** is 99 years later, i.e.:

<century window end> = <century window start> + 99.

WINDOW-BOUNDARY = *ADD-TO-CURRENT-YEAR
The **end** of the century window is computed as follows:

<century window end> = <current year> + <integer in century-window-shift>.

Since the century window is 100 years long, its **start** is 99 years earlier, i.e.:

<century window start> = <century window end> - 99.

Example:

The current year is 2002, and the SET-SORT-OPTIONS statement contains CENTURY-WINDOW-SHIFT=55. The century window therefore ranges from 1947 (=2002-55) through 2046 (=1947+99).

For the sort process, this means:
The two-digit year numbers 00 through 46 are interpreted as 2000 through 2046 and are therefore larger than the years 47 through 99 since these are interpreted as 1947 through 1999.

In this case, the sort sequence (with ascending sort) looks as follows:

44, 45, ..., 98, 99, 00, 01, ..., 42, 43.

IGNORE-UNICODE-BLANK =
This operand defines whether blanks or other characters represented by variable collation elements are taken into account when the sort sequence is defined.

IGNORE-UNICODE-BLANK = *NO
All characters are taken into account.

IGNORE-UNICODE-BLANK = *YES
Blanks and other characters represented by variable collation elements are not used to define the sort sequence. The search key is contracted and, if necessary, padded with the Unicode fill character at the end.

IGNORE-CHARACTER =
Specifies the characters that must be ignored by sort. The characters are ignored for the formats *CHARACTER and *TRANSLATE-CHARACTER. Although the characters are ignored when comparing the sort fields, they remain in the record.

IGNORE-CHARACTER = *NONE
No characters are to be ignored.

IGNORE-CHARACTER = list-poss(4): <c-string 1..1 with-low>
Character constant (1 byte).

IGNORE-CHARACTER = list-poss(4): <x-string 1..2>
Hexadecimal constant (1 byte).

Examples:

```
SET-SORT-OPTIONS IGNORE-CHARACTER=C' '
SET-SORT-OPTIONS IGNORE-CHARACTER=X'40'
SET-SORT-OPTIONS IGNORE-CHARACTER=(C' ',C'_')
SET-SORT-OPTIONS IGNORE-CHARACTER=(C' ',X'6D')
SET-SORT-OPTIONS IGNORE-CHARACTER=(C'(',C')',C'<',C'>')
SET-SORT-OPTIONS IGNORE-CHARACTER=*NONE
```

SHOW-SORT-DEFAULTS

SHOW-SORT-DEFAULTS is used to display default values.

The currently valid defaults for the following SORT parameters are output:

– minimum message weight that causes messages to be output (MIN-MSG-WEIGHT)

– maximum value for intensively used memory (CORE-MAXIMUM)

– lower limit for the CORE value calculation (CORE-MINIMUM)

– upper limit for the CORE value calculation (CORE-LIMIT)

– default for the CORE value (CORE-DEFAULT)

– job class for subtasks (SUBTASK-JOB-CLASS)

SHOW-SORT-DEFAULTS

This statement has no operands.

SORT-RECORDS

SORT-RECORDS defines sort, remainder and constant fields and their most significant attributes (position, length,...) for a sort run. Fields can be defined via symbolic names.

SORT-RECORDS

```
FIELDS = *COMPLETE-RECORD / list-poss(64): *FIELD-EXPLICIT(...) / *FIELD-SYMBOLIC(...) /
         *REMAINDER-EXPLICIT(...) / *REMAINDER-SYMBOLIC(...) / *CONSTANT-EXPLICIT(...) /
         *CONSTANT-SYMBOLIC(...)

  *FIELD-EXPLICIT(...)
      POSITION = <integer 1..32759>(...)
        <integer 1..32759>(...)
          BIT-POSITION = 0 / <integer 0..7>
      ,LENGTH = <integer 0..32759>(...)
        <integer 0..32759>(...)
          NUMBER-OF-BITS = 0 / <integer 0..7>
      ,FORMAT = *CHARACTER / *BINARY / *FIXED-POINT / *FLOATING-POINT /
                *PACKED-DECIMAL / *ZONED-DECIMAL / *EBCDIC-DIN /
                *EBCDIC-INTERNATIONAL / *PHYSICAL-TRANSLATE / *VIRTUAL-TRANSLATE /
                *MODIFY-CODE / *EBCDIC-ISO-EBCDIC / *EXTENDED-CHARACTER /
                *TRANSLATE-CHARACTER / *UNICODE-CHARACTER
      ,SORTING-ORDER = *ASCENDING / *DESCENDING / *EXTERNAL-COMPARE
      ,PRIORITY = *CURRENT-NUMBER / <integer 1..64>
      ,ELIMINATE = *NO / *YES
      ,PRINT-MASK = *NO / <c-string 1..254 with-low>
      ,TWO-DIGIT-YEAR = *NO / *YES
  *FIELD-SYMBOLIC(...)
      SORT-FIELD-NAME = list-poss(64): <name 1..20>
      ,SORTING-ORDER = *ASCENDING / *DESCENDING / *EXTERNAL-COMPARE
      ,PRIORITY = *CURRENT-NUMBER / <integer 1..64>
      ,ELIMINATE = *NO / *YES
      ,PRINT-MASK-NAME = *NO / <name 1..20>
      ,TWO-DIGIT-YEAR = *NO / *YES
```

(part 1 of 2)

```
    *REMAINDER-EXPLICIT(...)
        |  POSITION = <integer 1..32759>
        |  ,LENGTH = <integer 1..32759>
        |  ,FORMAT = *NO / *BINARY / *FIXED-POINT / *PACKED-DECIMAL / *ZONED-DECIMAL
        |  ,PRINT-MASK = *NO / <c-string 1..254 with-low>
    *REMAINDER-SYMBOLIC(...)
        |  FIELD-NAME = list-poss(64): <name 1..20>
        |  ,PRINT-MASK-NAME = *NO / <name 1..20>
    *CONSTANT-EXPLICIT(...)
        |  CONSTANT = <integer -2147483639..2147483639> / <c-string 1..256 with-low> /
        |                 <x-string 1..512>
    *CONSTANT-SYMBOLIC(...)
        |  CONSTANT-NAME = list-poss(64): <name 1..20>
,SORT-TYPE = *COMPLETE-RECORD / *COMPOUND-RECORD / *TAG-COMPOUND / *TAG-HEADER /
             *TAG-TRAILER
,ESTIMATED-RECORDS = *BY-CALCULATION / <integer 1..2147483639> / <alphanum-name 1..19>
,RECORDS-PER-CYCLE = *BY-CALCULATION / <integer 1..2147483639>
,INPUT-RANGE = *ALL / *PARAMETER(...)
    *PARAMETER(...)
        |  FROM-RECORD = 0 / <integer 0..2147483639> / <alphanum-name 1..19>
        |  ,NUMBER-OF-RECORDS = *REST-INPUT / <integer 1..2147483639> / <alphanum-name 1..19>
,CHECKPOINT = *NO / *YES
,KEEP-EQUAL-SEQUENCES = *NO / *YES
```

(part 2 of 2)

FIELDS =
The FIELDS operand is used to define sort, remainder and constant fields. At least 1 field must be specified; up to 64 fields may be specified in a list.

FIELDS = *COMPLETE-RECORD
With fixed record format, the whole record is used as a sort field. With variable record format, the record length field is not included in the record comparison. For the purpose of the comparison, the variable part of the record is padded to the maximum record length with the filler character specified in the FILLER operand of the SET-RECORD-ATTRIBUTES statement. *COMPLETE-RECORD may only be specified for full sorts (SORT-TYPE = *COMPLETE-RECORD).

FIELDS = *FIELD-EXPLICIT(...)
Specifies sort fields in terms of position, length and format.

POSITION = <integer 1..32759(...)>
Position of the sort field relative to start of record. In fixed-length records the first data field occupies position 1. In records with variable record format, the position of the first data field depends on the file type and the IGNORE-LENGTH-FIELD operand of the SET-SORT-OPTIONS statement. This dependency is shown in the following table:

IGNORE-LENGTH-FIELD=	*STD	*YES	*NO
BS2000 file	5	1	5
POSIX file	1	1	5

For all formats except CHARACTER, EBCDIC-DIN, EBCDIC-INTERNATIONAL, EXTENDED-CHARACTER and TRANSLATE-CHARACTER the position specifications for sort fields must be in the range 1 to 4096.

BIT-POSITION = 0 / <integer 0..7>
Position specified in bits in addition to the byte position entry. This entry is only permitted for sort fields in BINARY format when no PRINT-MASK operand has been specified for these.

LENGTH = <integer 0..32759(...)>
Length of the sort field. The length must be within the limits allowed for the format. Specifying BIT-POSITION and NUMBER-OF-BITS increases the length of the sort field by 1 byte (BIT-POSITION + NUMBER-OF-BITS ≤ 7) or by 2 bytes (BIT-POSITION + NUMBER-OF-BITS > 7). This must be taken into account when specifying the maximum length. With variable-length records, sort fields in CHARACTER, EBCDIC-DIN and EBCDIC-INTERNATIONAL format are allowed to extend into the variable part of the record.

NUMBER-OF-BITS = 0 / <integer 0..7>
Length specified in bits, in addition to the length in bytes. Specifying NUMBER-OF-BITS is allowed only for sort fields in BINARY format when the PRINT-MASK operand is not specified.

FORMAT = *CHARACTER / *BINARY / *FIXED-POINT / *FLOATING-POINT / *PACKED-DECIMAL / *ZONED-DECIMAL / *EBCDIC-DIN / *EBCDIC-INTERNATIONAL / *PHYSICAL-TRANSLATE / *VIRTUAL-TRANSLATE / *MODIFY-CODE / *EBCDIC-ISO-EBCDIC / *EXTENDED-CHARACTER / *TRANSLATE-CHARACTER / *UNICODE-CHARACTER
Format of the sort field (for attributes, see page 38).

SORTING-ORDER =
Order in which SORT is to arrange the records.

SORTING-ORDER = *ASCENDING
Ascending sorting order.

SORTING-ORDER = *DESCENDING
Descending sorting order. This option is not permitted for ISAM output files.

SORTING-ORDER = *EXTERNAL-COMPARE
The order for this sort field is defined by the user via the user exit EXTERNAL-COMPARE. This sort field must not be more than 255 bytes long.

PRIORITY =
Priority (weight) of the sort field. If several sort fields are specified, those with a priority of 1 are compared first. If they are identical, the sort fields with a priority of 2 are compared. This process is repeated until the sort fields compare unequal or until those with the highest priority have been compared.

PRIORITY = *CURRENT-NUMBER
The priority is determined by the order in which the sort fields are specified (the first sort field has a priority of 1).

PRIORITY = <integer 1..64>
Specifies the priority number. If a priority number is specified for one sort field, it must also be specified for all other sort fields. All the priority numbers must form a consecutive ascending sequence beginning with 1.

ELIMINATE =
Specifies whether the sort field is to be eliminated (not included in the output).

ELIMINATE = *NO
The sort field is not to be eliminated.

ELIMINATE = *YES
The sort field is to be eliminated. It serves solely to determine the sorting order. This option is not allowed for binary fields with bit position specifications or for mask fields. If specified in such cases, it is ignored and SORT issues a warning. When the OUTPUT user exit is used (with PARAMETER-MODE=24), identical records are not displayed.

PRINT-MASK =
Print mask used to edit the field for printing. Print masks are permitted only for selection sorting and only with the formats BINARY, FIXED-POINT, PACKED-DECIMAL and ZONED-DECIMAL. The combined length of all constants and print masks must not be greater than 4000 bytes. Note here that an additional 1-byte length field must be taken into account for each mask.

PRINT-MASK = *NO
No print mask is specified.

PRINT-MASK = <c-string 1..254>
Format of the print mask. The following characters may be used in a mask:

– a freely selectable filler character as the first character of the mask

– the control characters '#' (number sign) and '^' (circumflex)

– characters to be inserted (not equal to the control characters)

TWO-DIGIT-YEAR =
Determines whether the sort field contains a two-digit year number.

TWO-DIGIT-YEAR = *NO
The sort field does not contain a two-digit year number.

TWO-DIGIT-YEAR = *YES
The sort field contains a two-digit year number. This statement is legal only in combination with FORMAT=*PACKED-DECIMAL or FORMAT=*ZONED-DECIMAL. In this case, the sorting sequence depends on the CENTURY-WINDOW-SHIFT operand in the SET-SORT-OPTIONS statement.

The following sort field formats are possible:

FORMAT=	Sort field format
*PACKED-DECIMAL	X'0yyv' and X'0jjmmddv'
*ZONED-DECIMAL	X'Fyvy'

With: y = one digit of the two-digit year specification, mm = month, dd = day,
s = sign (must be positive)

FIELDS = *FIELD-SYMBOLIC(...)
Specifies sort fields using symbolic names which must previously have been defined as fields by means of the ADD-SYMBOLIC-NAMES statement.

SORT-FIELD-NAME = list-poss(64): <name 1..20>
Name of a sort field (max. 64 fields). If a list of symbolic names is given, the SORTING-ORDER, ELIMINATE and PRINT-MASK-NAME entries apply to all names; the value given for PRIORITY is used for the first name specified, and a value incremented by 1 each time is entered for the remaining names.

SORTING-ORDER =
Order in which SORT is to arrange the records.

SORTING-ORDER = *ASCENDING
Ascending sorting order.

SORTING-ORDER = *DESCENDING
Descending sorting order. This option is not allowed for ISAM output files.

SORTING-ORDER = *EXTERNAL-COMPARE
The order for this sort field is defined via the user exit EXTERNAL-COMPARE. The field must not be more than 255 bytes long.

PRIORITY =
Priority (weight) of the sort field. If several sort fields are specified, those with a priority of 1 are compared first. If they are identical, the sort fields with a priority of 2 are compared. This process is repeated until the sort fields compare unequal or until those with the highest priority have been compared.

PRIORITY = *CURRENT-NUMBER
The priority is determined by the order in which the sort fields are specified (the first sort field has a priority of 1).

PRIORITY = <integer 1..64>
Specifies the priority number. If a priority number is specified for one sort field, it must also be specified for all other sort fields. All the priority numbers must form a consecutive ascending sequence beginning with 1.

ELIMINATE =
Specifies whether the sort field is to be eliminated (not included in the output).

ELIMINATE = *NO
The sort field is not to be eliminated.

ELIMINATE = *YES
The sort field is to be eliminated. It serves solely to determine the sorting order. This option is not allowed for binary fields with bit position specifications or for mask fields. In such cases it is ignored and SORT issues a warning. If the OUTPUT user exit is used (with PARAMETER-MODE=24), identical records are not displayed.

PRINT-MASK-NAME =
Specifies a print mask by means of a symbolic name. Print masks are allowed for selection sorting only, and only for the formats BINARY, FIXED-POINT, PACKED-DECIMAL and ZONED-DECIMAL. The combined length of all constants and print masks must not be more than 4000 bytes. Note that an additional 1-byte length field also has to be taken into account for each mask.

PRINT-MASK-NAME = *NO
No print mask is specified.

PRINT-MASK-NAME = <c-string 1...254>
Format of the print mask. The following characters are permitted in masks:

– a freely selectable filler character as the first character of the mask

– the control characters '#' (number sign) and '^' (circumflex)

– characters to be inserted (not equal to the control characters)

TWO-DIGIT-YEAR =
Determines whether the sort field contains a two-digit year number.

TWO-DIGIT-YEAR = *NO
The sort field does not contain a two-digit year number.

TWO-DIGIT-YEAR = *YES
The sort field contains a two-digit year number. This statement is legal only in combination with FORMAT=*PACKED-DECIMAL or FORMAT=*ZONED-DECIMAL. In this case, the sorting sequence depends on the CENTURY-WINDOW-SHIFT operand in the SET-SORT-OPTIONS statement.

The following sort field formats are possible:

FORMAT=	Sort field format
*PACKED-DECIMAL	X'0yyv' and X'0jjmmddv'
*ZONED-DECIMAL	X'Fyvy'

With: y = one digit of the two-digit year specification, mm = month, dd = day,
s = sign (must be positive)

FIELDS = *REMAINDER-EXPLICIT(...)
Specifies remainder fields in terms of position and length. Remainder fields are used in selection and tag sorting for constructing the output record; they have no effect on the sorting order.

POSITION = <integer 1..32759(...)>
Position of the remainder field relative to start of record. In fixed-length records the first data field occupies position 1. In records with variable record format, the position of the first data field depends on the file type and the IGNORE-LENGTH-FIELD operand of the SET-SORT-OPTIONS statement. This dependency is shown in the following table:

IGNORE-LENGTH-FIELD=	*STD	*YES	*NO
BS2000 file	5	1	5
POSIX file	1	1	5

Remainder fields may start at any position in the record. In selection sorting with variable-length records, a remainder field may also be wholly or partially in the variable part of the record if it is the last field specified. With fixed-length records, the remainder field must be wholly within the record.

LENGTH = <integer 1..32759>
Length of the remainder field. The length of remainder fields is not limited provided they are within the specified record length.

FORMAT = *NO / *BINARY / *FIXED-POINT / *PACKED-DECIMAL / *ZONED-DECIMAL
Format of the remainder field. Specifying a format rather than NO is permitted only in combination with a print mask.

Format	Format description	Length in bytes
BINARY	Binary	1 bit - 256 bytes
FIXED-POINT	Fixed-point	1 - 256
PACKED-DECIMAL	Packed decimal	1 - 16
ZONED-DECIMAL	Zoned decimal	1 - 16
NO		1 - maximum record length

PRINT-MASK =
Print mask used to edit the field for printing. The remainder field must have one of the formats BINARY, FIXED-POINT, PACKED-DECIMAL or ZONED-DECIMAL.
The combined length of all constants and print masks must not be greater than 4000 bytes (note that an additional 1-byte length field must also be taken into account for each mask).

PRINT-MASK = *NO
No print mask is specified.

PRINT-MASK = <c-string 1..254>
Format of the print mask. The following characters are allowed in a print mask:

– a freely selectable filler character as the first character of the mask

– the control characters '#' (number sign) and '^' (circumflex)

– characters to be inserted (not equal to the control characters)

FIELDS = *REMAINDER-SYMBOLIC(...)
Specifies remainder fields using symbolic names which must previously have been defined as fields by means of the ADD-SYMBOLIC-NAMES statement. Remainder fields are used in selection or tag sorting for constructing the output record; they have no effect on the sorting order.

FIELD-NAME = list-poss(64): <name 1..20>
Name of the remainder field (max. 64 fields). If a list of symbolic names is given, the entries for PRINT-MASK-NAME apply to all the names.

PRINT-MASK-NAME =
Specifies a print mask name using a symbolic name which must previously have been defined as a mask by means of the ADD-SYMBOLIC-NAMES statement. The remainder field must have one of the formats BINARY, FIXED-POINT, PACKED-DECIMAL or ZONED-DECIMAL. The combined length of all constants and print masks must not be more than 4000 bytes; this total must also include the additional 1 byte required for a length field for each mask.

PRINT-MASK-NAME = *NO
No print mask is specified.

PRINT-MASK-NAME = <name 1..20>
Name of the print mask.

FIELDS = *CONSTANT-EXPLICIT(...)
Specifies constants used in selection or tag sorting for constructing the output record.

CONSTANT =
Value of the constant. The total length of all constants and print masks must not be more than 4000 bytes; this is including the extra 1 byte required for a length field for each mask.

CONSTANT = <integer-2147483639..2147483639>
Decimal numbers are converted by SORT into fixed-point numbers with a length of 4 bytes.

CONSTANT = <c-string 1..256 with-low>
Character string. A single quote embedded in a character string must be represented by 2 consecutive single quotes.

CONSTANT = <x-string 1..512>
Hexadecimal string.

FIELDS = *CONSTANT-SYMBOLIC(...)
Specifies constants using a symbolic name which must previously have been defined as a constant by means of the ADD-SYMBOLIC-NAMES statement. The constants are used in selection and tag sorting for constructing the output record.

CONSTANT-NAME = list-poss: <name 1..20>
Specifies a symbolic name. The total length of all constant fields and print masks must not be more than 4000 bytes; this is including the extra 1 byte required for a length field for each mask.

SORT-TYPE =
Type of sort by which the records are to be processed.

SORT-TYPE = *COMPLETE-RECORD
Full sort. The entire input record is processed in the sort operation.

SORT-TYPE = *COMPOUND-RECORD
Selection sort. The record to be sorted is composed of those parts of the input record that were specified in the FIELDS operand as sort fields, remainder fields and constants. The order in which these fields are combined to form the output record is determined by the order in which the fields were defined. In selection sorts, fixed-length records are generated as standard, regardless of the input record format.

SORT-TYPE = *TAG-COMPOUND
Tag sort. SORT prefixes the record address - for SAM files an extended record address (six-digit) - to the start of the selection record.

SORT-TYPE = *TAG-HEADER
Tag sort. SORT prefixes the record address to the start of the selection record.

SORT-TYPE = *TAG-TRAILER
Tag sort. SORT appends the record address to the end of the selection record.

ESTIMATED-RECORDS =
Approximate number of records to be processed. This helps SORT to work out the sort strategy and memory requirements with greater precision.

ESTIMATED-RECORDS = *BY-CALCULATION
No specification is given for the approximate number of records.

ESTIMATED-RECORDS = <integer 1..2147483639>
Estimated number of records to be sorted. Records that are to be inserted or omitted using user routines should not be taken into account for this estimate.

ESTIMATED-RECORDS = <alphanum-name 1..19>
Estimated number of records to be sorted exceeding a size of 2.14.483.639.

RECORDS-PER-CYCLE =
Size of a sort cycle. In each cycle, SORT is to presort the specified number of records, merge them internally and write the resulting record sequence to an auxiliary file. If a value has been specified for the ESTIMATED-RECORDS operand, the number of auxiliary files is derived from the ESTIMATED-RECORDS figure divided by the RECORDS-PER-CYCLE value. SORT sets up the calculated number of auxiliary files or makes up the number of existing auxiliary files to the calculated number.

RECORDS-PER-CYCLE = *BY-CALCULATION
No cycle sorting is performed by SORT.

RECORDS-PER-CYCLE = <integer 1..2147483639>
Number of records to be sorted in a cycle.

INPUT-RANGE =
Excludes records at the beginning or end of the input file from the sorting process.

INPUT-RANGE = *ALL
All records are included in the sort.

INPUT-RANGE = *PARAMETER(...)
Specifies a range of records to be included in the sort run.

FROM-RECORD =
Number of records that SORT is to skip, counting from the beginning of the input file.

FROM-RECORD = 0
No records are to be skipped.

FROM-RECORD = <integer 0..2147483639>
Number of records that SORT is to skip.

FROM-RECORD = <alphanum-name 1..19>
Number of records exceeding a size of 2.14.483.639 that SORT is to skip.

NUMBER-OF-RECORDS =
Number of records that SORT is to read.

NUMBER-OF-RECORDS = *REST-INPUT
SORT is to read all records up to end-of-file.

NUMBER-OF-RECORDS = <integer 1..2147483639>
Specifies the number of records.

NUMBER-OF-RECORDS = <alphanum-name 1..19>
Specifies the number of records exceeding a size of 2.14.483.639.

CHECKPOINT =
Controls the output of checkpoints.

CHECKPOINT = *NO
No checkpoints are written by SORT.

CHECKPOINT = *YES
SORT is to write a new checkpoint after every n records, where n is the number of records specified in the RECORDS-PER-CYCLE operand.

KEEP-EQUAL-SEQUENCES =
Controls whether the sequence of records with identical sort keys is to be preserved.

KEEP-EQUAL-SEQUENCES = *NO
The input sequence is not preserved. The order in which records with identical sort keys are transferred to the output is random.

KEEP-EQUAL-SEQUENCES = *YES
The input sequence of records with identical sort keys is preserved.

SUM-RECORDS

SUM-RECORDS is used to define sum fields. This is a means whereby records with identical sort keys are combined into a single record and the sum fields selected with SUM-RECORDS added together. Any summation that would result in an arithmetic overflow is cancelled and a warning is issued by SORT. Records with a unique sort key are not modified. Records with non-addable sum fields are retained intact.

SUM-RECORDS

FIELDS = ***NONE** / list-poss(64): ***FIE**LD-**EXPL**ICIT(...) / ***FIE**LD-**SYM**BOLIC(...)

***FIE**LD-**EXPL**ICIT(...)

POSITION = <integer 1..4096>

,**LEN**GTH = <integer 1..16>

,**FOR**MAT = ***FIX**ED-POINT / ***BI**NARY / ***PACK**ED-**DEC**IMAL / ***ZONED-DEC**IMAL

,**FIE**LD-**EXT**ENSION = **0** / <integer 0..16>

,**PRINT-MASK** = ***NO** / <c-string 1..254 with-low>

***FIE**LD-**SYM**BOLIC(...)

SUM-FIELD-**NAME** = list-poss(64): <name 1..20>

,**FIE**LD-**EXT**ENSION = **0** / <integer 0..16>

,**PRINT-M**ASK-**NAM**E = ***NO** / <name 1..20>

FIELDS =

Up to 64 sum fields can be specified in the FIELDS operand. If there are two records with identical sort keys, their sum fields are added together and one of the two records is eliminated. Sum fields must not overlap other sum fields or sort fields.

FIELDS = *NONE

Out of all the records with identical sort keys, one record is moved to the output and the rest are eliminated. No summation is performed. The record transferred for output is selected entirely at random.

FIELDS = *FIELD-EXPLICIT(...)
Specifies a sum field in terms of position, length and format.

POSITION = <integer 1..4096>
Position of the sum field relative to start of record.
In fixed-length records the first data field occupies position 1. In records with variable record format, the position of the first data field depends on the file type and the IGNORE-LENGTH-FIELD operand of the SET-SORT-OPTIONS statement. This dependency is shown in the following table:

IGNORE-LENGTH-FIELD=	*STD	*YES	*NO
BS2000 file	5	1	5
POSIX file	1	1	5

The record length field must not be used as a sum field. In selection and tag sorting, the specified positions refer to the newly constructed selection record (including any format modifications).

LENGTH = <integer 1..16>
Length of the sum field in bytes. The permissible or maximum length is determined by the format.

FORMAT = *FIXED-POINT / *BINARY / *PACKED-DECIMAL / *ZONED-DECIMAL
Format of the sum field. In ZONED-DECIMAL format, blanks (X'40') are automatically converted to zeros (X'F0'). With positive numbers, the sign zone of the final digit position is additionally set to X'Fx' (where $0 \leq x \leq 9$).

Format	Format description	Length in bytes
BINARY	Binary	2, 4, 8
FIXED-POINT	Fixed-point	2, 4, 8
PACKED-DECIMAL	Packed decimal	1 - 16
ZONED-DECIMAL	Zoned decimal	1 - 16

FIELD-EXTENSION = 0 / <integer 0..16>
Specifies a sum field extension. The sum field is extended (on the left) by the specified number of bytes. By this means it is also possible to combine records in which the addition of the sum fields would lead to an overflow. Padding for the extended sum fields is dependent on the format.
Even for extended sum fields, only the format-specific lengths or maximum lengths are permitted.

Format	Padding on the	Filler
BINARY	left	X'00' (zero)
FIXED-POINT	left	Sign
PACKED-DECIMAL	left	Zero
ZONED-DECIMAL	left	X'F0' (zero)

PRINT-MASK =
Print mask with which the sum field is edited for printing. Print masks are only permitted in selection sorting. The combined length of all the print masks must not be more than 2000 bytes, including the additional one byte required for a length field for each mask.

PRINT-MASK = *NO
No print mask is specified.

PRINT-MASK = <c-string 1..254_with-low>
Format of the print mask. The following characters are permitted in a print mask:

– a freely selectable filler character as the first character of the mask
– the control characters '#' (number sign) for digit selection and '^' (circumflex) for the start of the digits
– characters to be inserted (not equal to the control characters)

FIELDS = *FIELD-SYMBOLIC(...)
Specifies sum fields using symbolic names which must previously have been defined as fields by means of the ADD-SYMBOLIC-NAMES statement.

SUM-FIELD-NAME = list-poss(64): <name 1..20>
Name of the sum field. Up to 64 sum fields can be specified. If a list of symbolic names is used, the entries given for FIELD-EXTENSION and PRINT-MASK-NAME apply to all the names.

FIELD-EXTENSION = 0 / <integer 0..16>
Specifies a sum field extension. The sum field is extended (on the left) by the specified number of bytes. By this means it is also possible to combine records in which the addition of the sum fields would result in an overflow. Padding for the extended sum fields is dependent on the format.

PRINT-MASK-NAME =
Specifies a print mask using a symbolic name which must previously have been defined as a mask by means of the ADD-SYMBOLIC-NAMES statement. A print mask is permitted only in selection sorting.

PRINT-MASK-NAME = *NO
No print mask is specified.

PRINT-MASK-NAME = <name 1..20>
Name of the print mask.

5 Calling SORT

5.1 Calling SORT as a standalone program

SORT can be invoked by means of one of the following statements:

```
/START-SORT
/SORT-FILE
```

In interactive mode the start command is entered directly at the display terminal or is included within a procedure.

In batch mode the command is contained in an ENTER procedure.

In each case the SORT control statements are read from the SYSDTA system file.

START-SORT

Domain: FILE, UTILITIES

The START-SORT command is used to call SORT as a standalone program. For further processing, SORT control statements are necessary.

SORT delivers a command return code on termination (see “Command return codes” on page 202).

START-SORT
VERSION = ***STD** / <product-version 3..8 without-man> ,**CPU-LIM**IT = ***JOB-REST** / <integer 1..32767> ,**MONJV** = ***NONE** / <filename 1..54 without-gen-vers> ,**PROG**RAM-**MODE** = **24** / ***ANY** / ***DBL-DEFAULT**

VERSION =
Product version of SORT to be started (see also “Coexistence” on page 283).

VERSION = *STD
No explicit specification of the product version. In this case, the product version is selected as follows:

1. The version specified by the /SELECT-PRODUCT-VERSION command.
2. The highest SORT version installed with IMON.

VERSION = <product-version 3..8 without-man>
Explicit specification of product version using the format n.n or nn.n (where n stands for a number e.g. 07.9). It is possible to prefix the number with the character V, or to enter it in quotes (or as follows with a preceeding C, e.g. C’V07.9’).

For reasons of compatibility it is possible to specify the release and correction status. However, this specification is not evaluated since coexistence and exchangeability is only permitted for main versions.

CPU-LIMIT =
Maximum CPU time (in seconds) available to the program during its execution. If this time is exceeded, the sort/merge run terminates abnormally (with message SRT1038).

CPU-LIMIT = *JOB-REST
If the operand CPU-LIMIT=*NO is specified in the SET-LOGON-PARAMETERS command, there are no time constraints on the program. If the operand CPU-LIMIT= <integer 1..32767> is specified in the SET-LOGON-PARAMETERS command, the value defined at system generation is used as the time limit for the program.

CPU-LIMIT = <integer 1..32767>
Specifies the CPU time in seconds.

MONJV =
Specifies a job variable. This operand is effective only when the JV software product is installed on the system.

MONJV = *NONE
No job variable is specified.

MONJV = <filename 1..54>
Name of the job variable that is to monitor the program. If no JV of this name is present, it will be created. The system then sets the JV to the appropriate values in the course of program execution:
$R Program running
$T Program terminated
$A Program terminated abnormally
After termination of the SORT run, the JV also contains information on the reason for the termination. For the exact structure of the JV, see section “Error information when SORT is called as a standalone program” on page 398.

PROGRAM-MODE =
This operand determines whether SORT is loaded into the lower or upper (>16 Mb) address space.

PROGRAM-MODE = 24
SORT is loaded into the address space below the 16-Mb boundary. The program executes in 24-bit addressing mode. External references are interpreted as 24-bit addresses.

PROGRAM-MODE = *ANY
SORT can be loaded above or below the 16-Mb boundary.

PROGRAM-MODE = *DBL-DEFAULT
The setting made for the last MODIFY-DBL-DEFAULT command is valid here. At the beginning of the task *ANY is preset.

SORT-FILE

Domain: FILE, UTILITIES

SORT is invoked by the SORT-FILE command. In the command the user also defines the input and output files, sort fields and statement files for the sort run. Consequently, no SORT control statements are requested. Sorting is performed on the basis of the SORT-FILE specifications. If additional statements are required, these must be combined in statement files and assigned by means of the STATEMENT-FILES operand.

Note

This command is processed by a procedure. The operands of the command are passed as a single parameter. For example, the operand

```
FIELDS=FIELD-EXPLICIT(POSITION=100,LENGTH=10,FORMAT=CHARACTER,
                      SORTING-ORDER=ASCENDING)
```

is passed to the procedure as follows:

```
FIELDS=F(100,10,CH,A,,,,*NO)
```

A parameter may be up to 254 characters long. If this limit is exceeded (e.g. by specifying too many sort fields or too many statement files), the following message is issued:

```
SSM2054 SYMBOLIC OPERAND ERROR IN COMMAND. COMMAND IGNORED
```

The command should be simplified or alternatively the START-SORT command should be used.

SORT-FILE is always executed with
PROGRAM-MODE=*ANY,
MONJV=*NONE and
CPU-LIMIT=*JOB-REST.

```
SORT-FILE

 INPUT-FILES = *LINK / list-poss(99): <filename 1..54> / <posix-pathname 1..1023>
,OUTPUT-FILE = *LINK / <filename 1..54> / <posix-pathname 1..1023>
,CODE = *EBCDIC / *ASCII
,FIELDS = *COMPLETE-RECORD / list-poss(64): *FIELD-EXPLICIT(...) / *FIELD-SYMBOLIC(...)
    *FIELD-EXPLICIT(...)
        POSITION = <integer 1..32759>
        ,LENGTH = <integer 1..32759>
        ,FORMAT = *CHARACTER / *FIXED-POINT / *FLOATING-POINT / *PACKED-DECIMAL /
                  *ZONED-DECIMAL / *EBCDIC-DIN / *EBCDIC-INTERNATIONAL /
                  *PHYSICAL-TRANSLATE / *VIRTUAL-TRANSLATE / *MODIFY-CODE /
                  *EBCDIC-ISO-EBCDIC / *EXTENDED-CHARACTER / *TRANSLATE-CHARACTER /
                  *UNICODE-CHARACTER
        ,SORTING-ORDER = *ASCENDING / *DESCENDING
        ,TWO-DIGIT-YEAR = *NO / *YES
    *FIELD-SYMBOLIC(...)
        SORT-FIELD-NAME = list-poss(64): <name 1..20>
        ,SORTING-ORDER = *ASCENDING / *DESCENDING
        ,TWO-DIGIT-YEAR = *NO / *YES
,IGNORE-LENGTH-FIELD = *STD / *YES / *NO
,STATEMENT-FILES = *NONE / list-poss(10): <filename 1..54>
,VERSION = *STD / <product-version 3..8 without-man>
,CENTURY-WINDOW-SHIFT = 50(...) / <integer 0..99> (...)
    <integer 0..99>(...)
        WINDOW-BOUNDARY = *SUBTRACT-FROM-CURRENT-YEAR / *ADD-TO-CURRENT-YEAR
```

INPUT-FILES =
Assigns the input file(s) (max. 99).

INPUT-FILES = *LINK
The input files are assigned via ADD-FILE-LINK commands (file link name SORTIN or SORTINxx).
By default the file link name is released again when SORT terminates. If the LOCK-FILE-LINK command is specified before SORT is invoked, the file link name is retained.

INPUT-FILES = list-poss(99): <filename 1..54> / <posix-pathname 1..1023>
Name(s) of the input file(s). Input files assigned via a ADD-FILE-LINK command are ignored. Up to 99 input files may be specified.
POSIX file names must be specified in single quotes to distinguish them from BS2000 file names.
POSIX input files and BS2000 input files must not be used simultaneously in a sort run. If one input file is a POSIX file, all other input files must also be POSIX files.
Restriction:
The sum of the number and lengths of all specified POSIX input file names must not be greater than 5100. If any specifications exceed this, an error message is issued and SORT is aborted.

OUTPUT-FILE =
Assigns the output file.

OUTPUT-FILE = *LINK
The output file is assigned via a ADD-FILE-LINK command (file link name SORTOUT).

OUTPUT-FILE = <filename 1..54> / <posix-pathname 1..1023>
Name of the output file. An output file that has been assigned via a ADD-FILE-LINK command is ignored. Different file attributes from those taken over from the input file or defined by SORT can be assigned only by means of a ADD-FILE-LINK command.
The file link name SORTOUT is used, and is retained after the sort run is completed.
A POSIX file name must be specified in single quotes to distinguish it from a BS2000 file name.

CODE =
Specification of the code of POSIX files. This applies to both the input file and the output file and merely defines the encoding of the end-of-record identifier. In particular, this operand has **no** effect on the sort sequence. For BS2000 the operand has no significance and is ignored.

CODE = *EBCDIC
The end-of-record identifier is encoded according to EBCDIC and has the value X'15'.

CODE = *ASCII
The end-of-record identifier is encoded according to ASCII and has the value X'0A'.

FIELDS =
Specifies whether sorting is to be based on the complete input record or on individual sort fields. Describes sort fields in terms of position, length, format and sorting order. At least 1 field must be specified, up to 64 fields may be specified either explicitly or using their symbolic names.

FIELDS = *COMPLETE-RECORD
With fixed-length input records, the entire record is taken as a sort field. With variable records, the record length field does not form part of the sort field. When records of different lengths are compared, the shorter record is padded to the length of the longer with the value specified in the FILLER operand of the SET-RECORD-ATTRIBUTES statement (default value: X'00').

FIELDS = *FIELD-EXPLICIT(...)
Specifies sort fields in terms of position, length, format and sorting order.

POSITION = <integer 1..32759>
Position of the sort field relative to start of record.
In fixed-length records the first data field starts at position 1. For records with variable record format, this position depends on the file type and the value of the IGNORE-LENGTH-FIELD operand. The position of the first data field in these cases is shown in the following table:

IGNORE-LENGTH-FIELD=	*STD	*YES	*NO
BS2000 file	5	1	5
POSIX file	1	1	5

For all formats except CHARACTER, EBCDIC-DIN, EBCDIC-INTERNATIONAL, EXTENDED-CHARACTER and TRANSLATE-CHARACTER, the position specifications for sort fields must be in the range 1 through 4096.

LENGTH = <integer 1..32759>
Length of the sort field in bytes. The length must be within the limits allowed for the format.

FORMAT = *CHARACTER / *FIXED-POINT / *FLOATING-POINT / *PACKED-DECIMAL / *ZONED-DECIMAL / *EBCDIC-DIN / *EBCDIC-INTERNATIONAL / *PHYSICAL-TRANSLATE / *VIRTUAL-TRANSLATE / *MODIFY-CODE / *EBCDIC-ISO-EBCDIC / *EXTENDED-CHARACTER / *TRANSLATE-CHARACTER / *UNICODE-CHARACTER
Format of the sort field (for attributes, see section "Sort fields" on page 38).

SORTING-ORDER =
Order in which SORT is to arrange the records.

SORTING-ORDER = *ASCENDING
Ascending sorting order.

SORTING-ORDER = *DESCENDING
Descending sorting order. This option is not permitted for ISAM output files.

TWO-DIGIT-YEAR =
Determines whether the sort field contains a two-digit year number.

TWO-DIGIT-YEAR = *NO
The sort field does not contain a two-digit year number.

TWO-DIGIT-YEAR = *YES
The sort field contains a two-digit year number. This statement is legal only in combination with FORMAT=*PACKED-DECIMAL or FORMAT=*ZONED-DECIMAL. In this case, the sorting sequence depends on the CENTURY-WINDOW-SHIFT operand.

The following sort field formats are possible:

FORMAT=	Sort field format
*PACKED-DECIMAL	X'0yyv' and X'0jjmmddv'
*ZONED-DECIMAL	X'Fyvy'

Where: y = one digit of the two-digit year specification, mm = month, dd = day,
s = sign (must be positive)

FIELDS = *FIELD-SYMBOLIC(...)
Specifies sort fields using symbolic names which must previously have been defined as fields by means of the ADD-SYMBOLIC-NAMES statement (in a statement file).

SORT-FIELD-NAME = list-poss(64): <name 1..20>
Name(s) of the sort field(s). If a list of symbolic names (max. 64) is given, the SORTING-ORDER operand entry applies to all the names.

SORTING-ORDER =
Order in which SORT is to arrange the records.

SORTING-ORDER = *ASCENDING
Ascending sorting order.

SORTING-ORDER = *DESCENDING
Descending sorting order. This option is not permitted for ISAM output files.

TWO-DIGIT-YEAR =
Determines whether the sort field contains a two-digit year number.

TWO-DIGIT-YEAR = *NO
The sort field does not contain a two-digit year number.

TWO-DIGIT-YEAR = *YES
The sort field contains a two-digit year number. This statement is legal only in combination with FORMAT=*PACKED-DECIMAL or FORMAT=*ZONED-DECIMAL. In this case, the sorting sequence depends on the CENTURY-WINDOW-SHIFT operand.

The following sort field formats are possible:

FORMAT=	Sort field format
*PACKED-DECIMAL	X'0yyv' and X'0jjmmddv'
*ZONED-DECIMAL	X'Fyvy'

With: y = one digit of the two-digit year specification, mm = month, dd = day,
s = sign (must be positive)

IGNORE-LENGTH-FIELD =
Specifies whether, in the case of variable record lengths, the record length field must be taken into account in position specifications for control fields.

The specification applies to all control fields in the current sort run. It does not affect position specifications in fixed-length records.

IGNORE-LENGTH-FIELD = *STD
With BS2000 files, *STD has the same effect as *NO, i.e. the record length field must be taken into account in position specifications. The behavior is therefore the same as in the previous versions.
With POSIX files, *STD has the same effect as *YES. The record length field created internally by SORT is therefore ignored in the position calculation.

IGNORE-LENGTH-FIELD = *YES
The record length field is ignored in the position calculation. The first data field of a record begins at position 1.

IGNORE-LENGTH-FIELD = *NO
The record length field is taken into account in the position calculation. The first data field of a record begins at position 5.

STATEMENT-FILES =
Assigns one or more files (max. 10) containing SORT statements. The statement files are processed before the sort statement.

STATEMENT-FILES = *NONE
No statement files are specified.

STATEMENT-FILES = list-poss:(10)<filename 1..54>
Names of the statement files.

VERSION =
Product version of SORT to be started (see also “Coexistence” on page 283).

VERSION = *STD
No explicit specification of the product version. In this case, the product version is selected as follows:

1. The version specified by the /SELECT-PRODUCT-VERSION command.
2. The highest SORT version installed with IMON.

VERSION = <product-version 3..8 without-man>
Explicit specification of product version using the form n.n or nn.n (where n stands for a number e.g. 07.9). It is possible to prefix the number with the character V, or to enter in quotes (or as follows with a preceeding C, e.g. C’V07.9’).

For reasons of compatibility it is possible to specify the release and correction status. However, this specification is not evaluated since coexistence and exchangeability is only permitted for main versions.

CENTURY-WINDOW-SHIFT = 50(...) / <integer 0..99>(...)
Determines how SORT interprets and sorts fields that contain two-digit years (FORMAT= *PACKED-DECIMAL or *ZONED-DECIMAL and TWO-DIGIT-YEAR=*YES). The CENTURY-WINDOW-SHIFT operand defines a century window within which sorting is executed (ascending or descending order). Depending on what is specified for the WINDOW-BOUNDARY operand, this century window is determined by either its start or its end date.

WINDOW-BOUNDARY =
Specifies the way the start and end of the century window is calculated.

WINDOW-BOUNDARY = *SUBTRACT-FROM-CURRENT-YEAR
The **start** of the century window is computed as follows:

<century window start> = <current year> - <integer in century-window-shift>.

Since the century window is 100 years long, its **end** is 99 years later, i.e.:

<century window end> = <century window start> + 99.

WINDOW-BOUNDARY = *ADD-TO-CURRENT-YEAR
The **end** of the century window is computed as follows:

<century window end> = <current year> + <integer in century-window-shift>.

Since the century window is 100 years long, its **start** is 99 years earlier, i.e.:

<century window start> = <century window end> - 99.

Example:

The current year is 2002, and the SET-SORT-OPTIONS statement contains CENTURY-WINDOW-SHIFT=55. The century window therefore ranges from 1947 (=2002-55) through 2046 (=1947+99).

For the sort process, this means:
The two-digit year numbers 00 through 46 are interpreted as 2000 through 2046 and are therefore larger than the years 47 through 99 since these are interpreted as 1947 through 1999.

In this case, the sort sequence (with ascending sort) looks as follows:

44, 45, ..., 98, 99, 00, 01, ..., 42, 43.

Command return codes

SORT delivers a command return code, which can be used with SDF-P statements for control in S procedures. The command return code allows you to provide specific responses to specific error situations.

The command return code consists of three parts:

– the main code, which corresponds to a message code; this allows detailed information to be queried with the HELP-MSG-INFORMATION command
– the subcode1 (SC1), which classifies the error situations into error classes as they occur; this code indicates the severity of an error
– the subcode2 (SC2), which can contain additional information (value other than zero)

If SORT is invoked as a subroutine, no command return code is delivered.

If several warnings occur, only the command return code for the last warning is delivered.

If both warnings and errors occur, the command return code for the error is delivered.

(SC2)	SC1	Maincode	Meaning
0	0	CMD0001	Normal termination of the SORT run. Neither errors nor warnings occurred.
2	0	SRTxxxx	Normal termination of the SORT run. Errors occurred but not warnings. The main code contains the number of the last warning message to occur. Affected SRT messages (xxxx =): 1034, 1042, 1054, 1055, 1058, 1060, 1065, 1066, 1070, 1082, 1086, 1129, 1133, 1159, 1161, 1162, 1163, 1166, 1167, 1173, 1174, 1179, 1183, 1186, 1189, 1190, 1191, 1192, 1196, 1207, 1219, 1221, 1222, 1227, 1228, 1229, 1230, 1256, 1310

(SC2)	SC1	Maincode	Meaning
0	1	SRTxxxx	Abnormal termination of the SORT run with an error. The main code contains the number of the abort message. Affected SRT messages (xxxx =): 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079, 1080, 1081, 1085, 1088, 1099, 1103*, 1104*, 1105*, 1106*, 1107*, 1108*, 1109*, 1110*, 1111*, 1112*, 1113*, 1114*, 1115*, 1116*, 1117*, 1118*, 1119*, 1120*, 1121*, 1122*, 1123*, 1124*, 1125*, 1126*, 1131*, 1134, 1135, 1136, 1138, 1139, 1140*, 1141*, 1142*, 1143*, 1144*, 1145*, 1146*, 1147*, 1148*, 1155*, 1156*, 1157*, 1158*, 1168*, 1169*, 1170*, 1171, 1172, 1178, 1180, 1181, 1182, 1184*, 1185, 1187*, 1188*, 1193*, 1194*, 1195, 1199*, 1201, 1203, 1204, 1205, 1206, 1209, 1211, 1212, 1213, 1215, 1218, 1220, 1223, 1224, 1225, 1226, 1231, 1232, 1233, 1234, 1235, 1236, 1237, 1238, 1239, 1240, 1250, 1251, 1253, 1254, 1260, 1261, 1301, 1302, 1303, 1304, 1305, 1307, 1308, 1309, 1311, 1317 * The messages marked with an asterisk refer to errored SORT control statements. If these were entered in the dialog and corrected by a subsequent statement, the return code is not given a value.
0	32	SRTxxxx	Abnormal termination of the SORT run with an internal error. The main code contains the number of the abort message. Affected SRT messages (xxxx =): 1039, 1069, 1127
0	64	SRTxxxx	Abnormal termination of the SORT run with a different type of error. The main code contains the number of the abort message. Affected SRT messages (xxxx =): 1032, 1035, 1036, 1040, 1043, 1044, 1045, 1047, 1048, 1051, 1056, 1057, 1064, 1067, 1068, 1083, 1084, 1087, 1089, 1132, 1137, 1149, 1150, 1197, 1202, 1208, 1214, 1216, 1244, 1252, 1255, 1257, 1258, 1259, 1262, 1263, 1264, 1266, 1306, 1314, 1315, 1316
0	255	SRT1312	Abnormal termination of the SORT run because the requested SORT version is not available.

In the event of an error, the components of the return code can be queried with the SDF-P functions SUBCODE1(), SUBCODE2() and MAINCODE().

After an error-free SORT run (SC1 = 0), the return code can be saved with the SAVE-RETURNCODE command and also evaluated. (For further information on command return codes and on querying return codes, see the “SDF-P” manual [13].)

Example of querying return codes

```
/SORT-FILE INPUT,OUTPUT ————————————————————————————————————  (1)
/SAVE-RETURNCODE ———————————————————————————————————————————  (2)
/IF-BLOCK-ERROR ————————————————————————————————————————————  (3)
/  WRITE-TEXT '————————————————————————————————————————————'  ———  (4)
/  WRITE-TEXT '--- SORT run terminated with error              -'
/  WRITE-TEXT '————————————————————————————————————————————'
/ELSE
/  IF &SUBCODE2 > 0 ————————————————————————————————————————————  (5)
/    WRITE-TEXT '————————————————————————————————————————————'  ——  (6)
/    WRITE-TEXT '--- SORT run terminated with warning            -'
/    WRITE-TEXT '————————————————————————————————————————————'
/  ELSE
/    WRITE-TEXT '————————————————————————————————————————————'  ——  (7)
/    WRITE-TEXT '--- SORT run terminated without warning or error -'
/    WRITE-TEXT '————————————————————————————————————————————'
/  END-IF
/END-IF
/IF &SUBCODE1 > 0 OR &SUBCODE2 > 0 ———————————————————————————  (8)
/  WRITE-TEXT 'Subcode 1: &SUBCODE1'
/  WRITE-TEXT 'Subcode 2: &SUBCODE2'
/  WRITE-TEXT 'Main code: &MAINCODE'
/  HELP-MSG-INFORMATION &MAINCODE
/END-IF
```

(1) Start command for SORT.

(2) Saves the SORT return code
(only executed if SORT terminates normally, i.e. if the spin-off mechanism was not activated).

(3) Checks the SORT abortion (also restart point in the procedure if the spin-off mechanism was activated).

(4) Output after abortion of SORT.

(5) If the SORT run was not aborted, checks whether a warning was issued (subcode 2 greater than zero).

(6) Output if at least one warning but no error occurred in the SORT run.

(7) Output if neither warnings nor errors occurred in the SORT run.

(8) If a warning or an error occurred (at least one of the two subcodes is greater than 0), the three components of the return code and the corresponding message are output.

5.2 Calling SORT as a subroutine

SORT is invoked as a subroutine by means of a subroutine branch to SORTU or ILSORT. The return address must be stored in register 14.

Note

SORT V8.0 can be started with a starter module SRT80, Version level SORT V7.3 and later. In contrast, there is no guarantee that older SORT versions with a SRT80 starter module, version SORT V8.0, will function.

Example

```
L     15,=V(SORTU)        or    L     15,=V(ILSORT)
BALR  14,15                     BALR  14,15
```

Before the program is executed, the unresolved external reference to the SORT starter module SRT80 must be resolved. The easiest way to do this is using the DBL (dynamic binder loader).

So that the DBL can find the starter module SRT80, it must be informed of the name of the associated library in a ADD-FILE-LINK command with the link name BLSLIBxx.

If SORT was installed using IMON-GPN, the name of the sort library is freely selectable. It then has to be determined from the IMON-SCI (Software-Configuration-Inventory), the central IMON data basis. SDF-P procedures are available for doing this.

The command

```
/SET-VARIABLE SORTLIB=INSTALLATION-PATH -
/                     (LOGICAL-ID        = 'SYSLNK' -
/                     ,INSTALLATION-UNIT = 'SORT' -
/                     ,VERSION           = '*STD' -
/                     ,DEFAULT-PATH-NAME = '$.SORTLIB')
```

assigns the name of the current sort library to the SORTLIB variable.

In the subsequent command

```
/ADD-FILE-LINK LINK-NAME=BLSLIBxx,FILE-NAME=&SORTLIB
```

the name ascertained in this way is used.

The following excerpt from a sample procedure shows how, following successful assembly of the main program, the generated *OMF file can be invoked.

```
                :
          Assemble main program
                :
/SET-VARIABLE SORTLIB = '$.SORTLIB'
/IF (SDF-P-VERSION >= 'V02.0A00')
/ SET-VARIABLE SORTLIB = INSTALLATION-PATH -
/                        (LOGICAL-ID        = 'SYSLNK' -
/                        ,INSTALLATION-UNIT = 'SORT' -
/                        ,VERSION           = '*STD' -
/                        ,DEFAULT-PATH-NAME = '&SORTLIB')
/END-IF
/ADD-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=&SORTLIB
/START-EXE-PROG FROM-FILE=*OMF
                :
```

Procedures for static linking using BINDER must be similarly structured.

A similar procedure is available for specifying the macro library (previously $.SORTMACLIB) in assembler runs.

If the program is called using `/START-PROGRAM FROM-FILE=*MODULE(*OMF)` or `/EXEC *` then RUN-MODE=*STD is valid. In this case, the library used to fulfill external references must be assigned using /SET-TASKLIB.

Register conventions

Reg 1 Contains the address of the input block SVB (static communication area).

Reg 13 Contains the address of the 18-word save area to which the register contents of the calling program are saved. This save area must be set up by the user.

Reg 14 Return address of the calling program.

Reg 15 The low-order byte contains the return code which is passed at the end of the sort/merge run.

X'00' The sort/merge run terminated normally.

X'FF' The sort/merge run terminated abnormally. The errors are indicated in a message.

If an error occurs, the two high-order bytes contain the last 4 digits of the SORT message key, as an unsigned packed decimal number. For example, if a SORT run terminates abnormally with error message SRT1035, register 15 contains X'103500FF'.

Data must be transferred to SORT from the program address space.

If the operand STXIT=YES is specified in the associated SORT macro (SRT0, SRT1), SORT checks the validity of the transferred data and addresses. If a program error is detected, a message (SRT1071 to SRT1077) is displayed and the program run is terminated. In this case, the program error must be rectified and the program must be run again. Transferred data which is valid, but not correct, can result in unpredictable program behavior.

5.2.1 Passing control information to SORT

When SORT is called as a subroutine by another program, the control parameters for the sort/merge run must be passed by means of

– SORT statements (see chapter “SORT statements” on page 125) or

– SORT macro calls (see section “SORT macros” on page 212).

Two different levels are available for passing the control information to SORT.

Level 0 SORT expects statements via SYSDTA. The user creates the associated input block SVB by means of a SRT0 macro (see section “SORT macros” on page 212). Register 1 then points to the address of the SRT0 macro.

Level 1 SORT expects statements in the calling program as variable-length records. The user sets up the statements and the associated input block SVB by means a sequence of SRT1 macros (see section “SORT macros” on page 212). Register 1 points to the address of the first SRT1 macro in the sequence.

5.2.1.1 Level 0

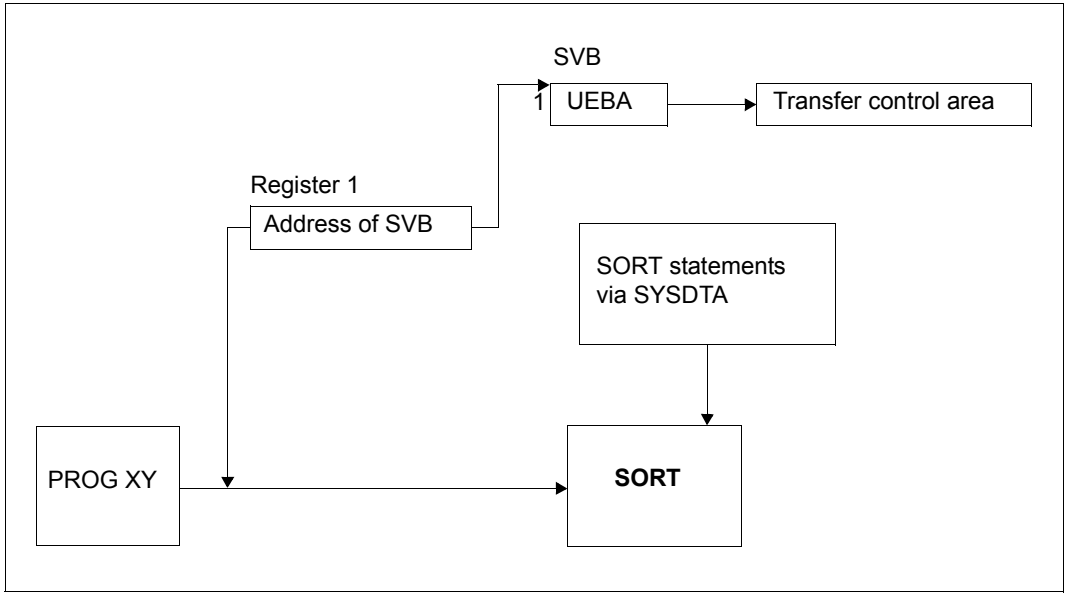


Figure 8: Calling SORT as a subroutine at level 0

At level 0, SORT expects the SORT statements to be supplied via SYSDTA. Register 1 must point to the input block (SVB). The user can set up the SVB input block by means of the SRT0 macro (see section “SORT macros” on page 212). The structure of the SVB is described in the appendix on page 404.

5.2.1.2 Level 1

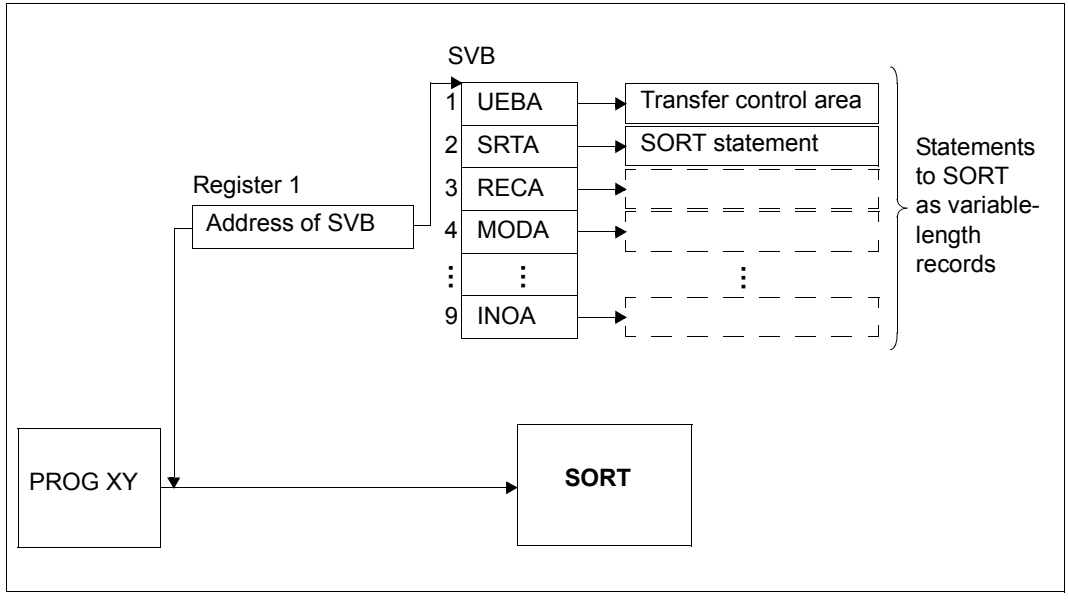


Figure 9: Calling SORT as a subroutine at level 1

At level 1, SORT expects the SORT statements to be passed as variable-length records in the calling program. Register 1 must point to the SVB input block. The user generates the level 1 statements by means of SRT1 macros (see section “SORT macros” on page 212). The structure of the SVB input block is described in the appendix on page 404.

5.2.2 SORT macros

The following macros are available for calling SORT as a subroutine at levels 0 and 1:

SRT0 Call to SORT at level 0
SRT1 Call to SORT at level 1

These macros also set up the input block SVB, including the transfer control area.

5.2.2.1 SRT0: call to SORT at level 0

The SRT0 macro allows the user to input control information to the SORT sort/merge program via level 0.

Name	Operation	Operands
[name]	SRT0	[SDF = NO / YES] [,STXIT = YES / NO] [,RDONLY = NO / YES] [,RCF = NO / aaddr / saddr] [,RCFL = length] [,MSGPROT = OUT / LST / BOTH / NO] [,VERSION=*STD / version] [,MULTI=*STD / OPT / NOIMON

name — A short symbolic name (max. 4 characters long) used to address the SVB input block for passing control data at level 0. The address must be loaded into register 1. The name is added as a prefix to all symbolic names generated by means of the SRT0 macro.

SDF = NO / YES — If SDF is to be used, SDF=YES must be specified explicitly. Omitting the operand or specifying NO means that subsequent inputs have to be in ISP format.

STXIT = YES / NO — Specifies whether the STXIT facility is to be used (see the “Executive Macros” manual [6]).

YES — SORT uses the STXIT entries.

NO — SORT does not use the STXIT entries.

STXIT=NO has the following effect:

– the user cannot interrupt the SORT run by means of the SEND-MSG command.
– SORT does not output a special dump when an error condition occurs.
– user data and addresses are not checked for validity.

RDONLY = YES / NO Specifies whether the SRT0 macros are stored in a write-protected or non-write-protected module.

YES The SRT0 macros are stored in a write-protected READONLY module (e.g. for shared access).

NO The SRT0 macros are stored in a module which is not write-protected.

RCF = aaddr / saddr / NO

Specifies an area for storing the return code and SORT messages. The first 4 bytes of the RCF area are used to store the return code RC (see appendix, page 401). Following this, the SORT messages output as a result of the MIN-MSG-WEIGHT operand of the SET-SORT-OPTIONS statement are stored contiguously as variable-length records (corresponding to an output to SYSOUT/SYSLST).

aaddr Symbolic address of the area.

Example

RCF=RCAREA

saddr Symbolic address of the area in S-type address format.

"saddr" must be in parentheses, as in Assembler notation, but without an S as prefix.

Example

RCF=(RCAREA)
RCF=(DISPLACEMENT(REG))
RCF=(0(5))

NO No storage area is set up for the return code or the SORT messages.

RCFL = length Defines the length of the specified RCF area. The maximum permitted length is 32767 bytes. If the RCFL operand is not specified or has a value less than 4, a length of 4 bytes is assumed.

MSGPROT = OUT / LST / BOTH / NO

Specifies whether SORT messages are to be written to SYSOUT and/or SYSLST. If the RCF operand is specified, the SORT messages are additionally written to the RCF area.

OUT SORT messages are written to SYSOUT and to the RCF area.

LST SORT messages are written to SYSLST and to the RCF area.

BOTH SORT messages are written to SYSOUT and SYSLST and to the RCF area.

NO — No SORT messages are written to SYSOUT or SYSLST. The messages are written to the RCF area only.

VERSION = *STD / version

Specifies the SORT version to be dynamically loaded as a subroutine.
The version specification is interpreted in the same way as when the main program is called.

*STD — No special SORT version is requested.

version — Version name in the format [v]v.v[a[nn]]

[v]v.v : Product version

a : Release status

nn : Revision status

– It is not possible to specify an additional “V” and single quotes, as with the start command.

– Release and correction status can be specified, although they will not be evaluated since coexistence and exchangeability are only permitted for main versions.

MULTI = STD / OPT / NOIMON

Specifies whether the file names should be determined and the SYSPAR parameter file read during every sort run.

If SORT is called frequently (more than 1000 times) as a subroutine, reducing the operations that need to be performed optimizes the runtime.

STD — All preparatory operations are executed. The SYSPAR parameter file is read at the start of each sort run.

OPT — The file names are determined by means of IMON only when the SORT subroutine is called for the first time. The SYSPAR parameter file is read only at the start of the first sort run.

NOIMON — The file names are not determined by means of IMON. The default names apply (e.g. SYSLNK.SORT.ver). The SYSPAR parameter file is read at the start. The generated defaults apply for the parameters.

Note

If MULTI= OPT or MULTI=NOIMON is set and the MODIFY-SORT-DEFAULTS statement is specified during a sort run, the parameters selected there apply for this sort run. These parameters can be saved in the SYSPAR file using SAVE-DEFAULTS=*YES. However, the changes are not effective for subsequent calls of a sort subroutine in the same main program call.

5.2.2.2 SRT1: call to SORT at level 1

The SRT1 macro allows the user to input all the necessary control information to the SORT sort/merge program via level 1, including the SVB input block and the transfer control area.

One SRT1 macro call is required for each statement. For the SVB input block and for each statement specified, SORT generates a symbolic name consisting of the short name and the first 2 or 3 characters of the statement. These names appear in the macro listing if “PRINT GEN” has been specified. A statement whose contents are to be modified can be addressed using its symbolic name.

Name	Operation	Operands
[name]	SRT1	(statementtext) [SDF = NO / YES] [STXIT = YES / NO] [,RDONLY = NO / YES] [,RCF = NO / aaddr / saddr] [,RCFL = length] [,MSGPROT = OUT / LST / BOTH / NO] [,VERSION=*STD / version [,MULTI=*STD / OPT / NOIMON

name

A short symbolic name (max. 4 characters long) that identifies a group of SRT1 macros used to generate the statements for a SORT run. It must be specified in the first SRT1 macro of the group. None of the other SRT1 macros in the same SORT run may specify “name”.

The name is used for addressing the SVB input block for passing control data at level 1 and thus creates the reference to the statement. The address must be loaded into register 1.
The name is added as a prefix to all symbolic names generated by means of the SRT1 macro.

statementtext — The relevant SORT statement, enclosed in parentheses, must be specified here. The format is the same as for input at level 0. Abbreviated keywords and positional operands are permitted.

SDF = YES / NO — If SDF is to be used, SDF = YES must be specified explicitly. Omitting the operand or specifying NO means that the statements have to be specified in ISP format.

STXIT = YES / NO — Specifies whether the STXIT facility is to be used (see the "Executive Macros" manual [6]). This operand may only be specified in the first SRT1 macro, identified by "name".

YES — SORT uses the STXIT entries.

NO — SORT does not use the STXIT entries.

STXIT=NO has the following effect:
- the user cannot interrupt the SORT run by means of the SENDMSG command
- SORT does not output a special dump when an error condition occurs
- user data and addresses are not checked for validity

RDONLY = YES / NO — Specifies whether the SRT1 macros are stored in a write-protected or non-write-protected module.

YES — The SRT1 macros are stored in a write-protected READONLY module (e.g. for shared access).

NO — The SRT1 macros are stored in a module which is not write-protected.

RCF = aaddr / saddr / NO

Specifies an area for storing the return code and SORT messages. The first 4 bytes of the RCF area are used to store the return code RC (see appendix, page 401). Following this, the SORT messages output as a result of the MIN-MSG-WEIGHT operand of the SET-SORT-OPTIONS statement are stored contiguously as variable-length records (corresponding to an output to SYSOUT/SYSLST).

aaddr — Symbolic address of the area.

Example

RCF=RCAREA

saddr — Symbolic address of the area in S-type address format.

"saddr" must be in parentheses, as in Assembler notation, but without an S as prefix.

Example

```
RCF=(RCAREA)
RCF=(DISPLACEMENT(REG))
RCF=(0(5))
```

NO — No storage area is set up for the return code or the SORT messages.

RCFL = length — Defines the length of the specified RCF area.

The maximum permitted length is 32767 bytes. If the RCFL operand is not specified or has a value less than 4, a length of 4 bytes is assumed.

MSGPROT = OUT / LST / BOTH / NO

Specifies whether SORT messages are to be written to SYSOUT and/or SYSLST. If the RCF operand is specified, the SORT messages are additionally written to the RCF area.

OUT — SORT messages are written to SYSOUT and to the RCF area.

LST — SORT messages are written to SYSLST and to the RCF area.

BOTH — SORT messages are written to SYSOUT and SYSLST and to the RCF area.

NO — No SORT messages are written to SYSOUT or SYSLST. The messages are written to the RCF area only.

VERSION = *STD / version

Specifies the SORT version to be dynamically loaded as a subroutine.
The version specification is interpreted in the same way as when the main program is called.

*STD — No special SORT version is requested.

version — Version name in the format [v]v.v[a[nn]]

- [v]v.v : Product version
- a : Release status
- nn : Revision status

– It is not possible to specify an additional “V” and single quotes, as with the start command.

– Release and correction status can be specified, although they will not be evaluated since coexistence and exchangeability are only permitted for main versions.

MULTI = STD / OPT / NOIMON

Specifies whether the file names should be determined and the SYSPAR parameter file read during every sort run.

If SORT is called frequently (more than 1000 times) as a subroutine, reducing the operations that need to be performed optimizes the runtime.

STD — All preparatory operations are executed. The SYSPAR parameter file is read at the start of each sort run.

OPT — The file names are determined by means of IMON only when the SORT subroutine is called for the first time. The SYSPAR parameter file is read only at the start of the first sort run.

NOIMON — The file names are not determined by means of IMON. The default names apply (e.g. SYSLNK.SORT.ver). The SYSPAR parameter file is read once when it is first called. Its values are then used to create the SORT standard parameters.

Note

If MULTI= OPT or MULTI=NOIMON is set and the MODIFY-SORT-DEFAULTS statement is specified during a sort run, the parameters selected there apply for this sort run. These parameters can be saved in the SYSPAR file using SAVE-DEFAULTS=*YES. However, the changes are not effective for subsequent calls of a sort subroutine in the same main program call.

Conventions for SRT1 macros

– One SRT1 macro is required for each statement.
– The statement text has to be enclosed in parentheses.
– Abbreviated keywords and positional operands are permitted.
– The first SRT1 macro must be identified by a 4-character name. None of the remaining SRT1 macros, including SRT1 (END), may specify a symbolic name.
– SDF=YES must also be added to the end of the first SRT1 macro, outside of the parentheses which surround the macro itself.
– Apart from this, the SRT1 macro calls must conform to Assembler macro conventions, e.g. in respect of continuation lines, length constraints per operand=127, etc.
The entire statement text thus forms an operand of the macro statement.
In the SELECT-INPUT-RECORDS statement, the comma before and after AND/OR must **not** be omitted.
– When a SRT1 macro is converted into Assembler DC statements, every occurrence of a single quote (in the representation of constants) is doubled in accordance with Assembler conventions. If single quotes are also to be used within a constant, however, all the single quotes must be explicitly included (e.g. FILLER=’’’’’).

Example

```
         START
         .
         .
         .
         LA        13,SAVE
         LA        1,VS1
         L         15,=V(SORTU)
         BALR      14,15
         .
         .
         .
SAVE     DS        18F
VS1      SRT1      (SORT-RECORDS FIELDS=*FIELD-EXPLICIT(POSITION=30,
                   LENGTH=8)),SDF=YES,STXIT=NO
         SRT1      (ASSIGN-EXITS PLANNING=*DIALOG)
         SRT1      (SET-SORT-OPTIONS MIN-MSG-WEIGHT=*ALL)
         SRT1      (END)
```

5.3 SORT access method SORTZM

For full sorts and selection sorts, the user can call SORT not only as a standalone program (using the /START-SORT command) or as a subroutine (via SORTU or ILSORT) but also via a special access method (SORTZM).

5.3.1 Function of the SORT access method SORTZM

Figure 10 shows how SORTZM works. The SORTZM access method passes the input records to SORT for sorting and receives them back from SORT once they have been sorted. The main program must provide a special record buffer in which the records to be sorted are made available to SORT. Which records are selected for sorting is always left up to the user. SORT does not process records selected via the SELECT-INPUT-RECORDS statement. SORT also ignores any entry for INPUT-RANGE in the SORT-RECORDS statement and the INPUT user exit.

The sorted records are subsequently returned by SORT to a separate record buffer.
The link between the main program and SORT is established by the following macros, which must be specified in the order given:

SRTOPEN	Initiate sorting
SRTPUT	Pass record to SORT (one SRTPUT call required per record)
SRTGET	Fetch record from SORT (one SRTGET call required per record)
SRTCLSE	Terminate sorting

SRTCLSE is also necessary in the case of empty sorts. SRTPUT and SRTGET may be omitted (empty sorts). The SRTGET loop may also be closed prematurely, if required, by a SRTCLSE macro.
The SORTZM access method can pass the SORT statements to SORT at level 0 or level 1 (see page 209).

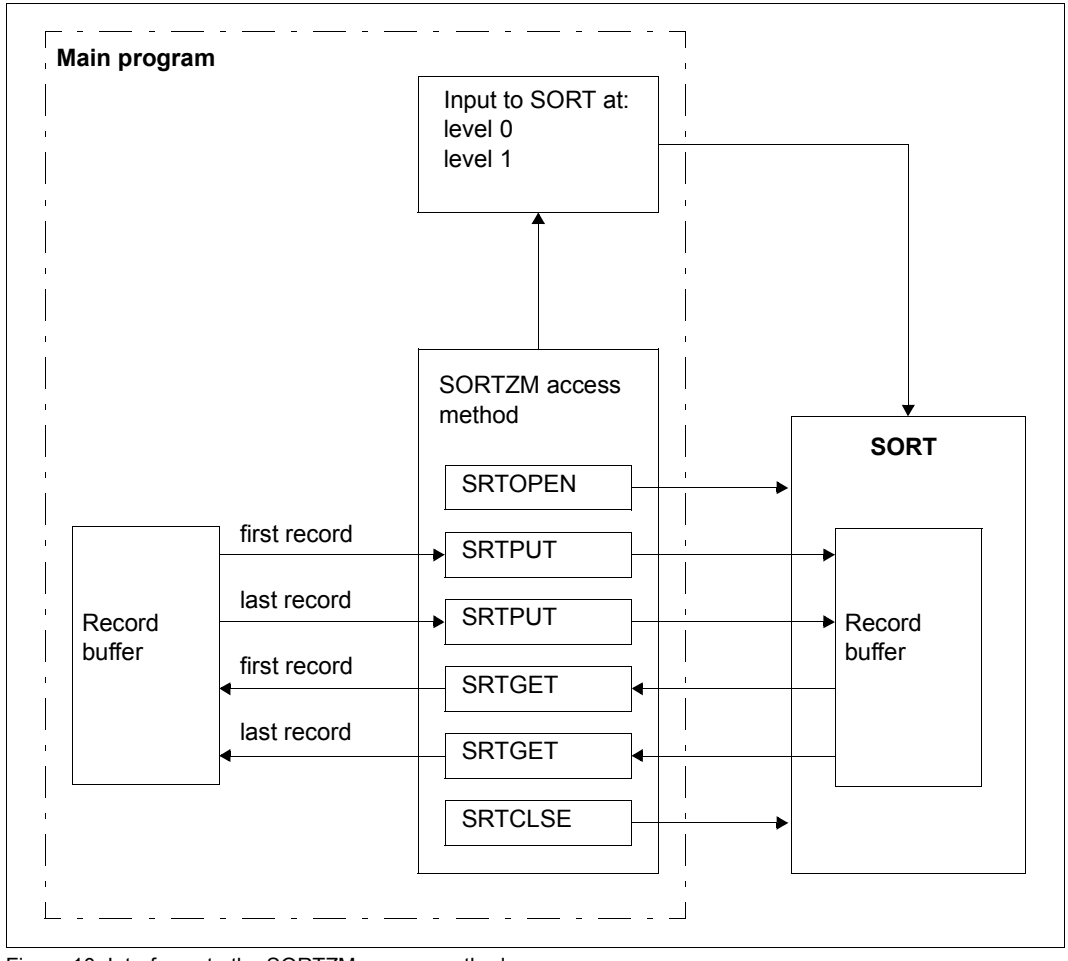


Figure 10: Interfaces to the SORTZM access method

Conditions for the use of SORTZM

- Mixing files is not permitted.
- Specified input and output files and the INPUT and OUTPUT user exits are ignored.
- SORTZM supports only full sorts and selection sorts; tag sorts are not possible.
- The SET-RECORD-ATTRIBUTES statement must be included in the input specifications.
- Registers 0, 1, 14 and 15 are modified by SORTZM.
- If the operand STXIT=YES is specified in the associated SORT macro (SRT0, SRT1), SAORT checks the validity of the transferred record address and length. If an error is detected, message SRT1079 is displayed and the program run is terminated abnormally (see the ERROR operand for SRTOPEN). In this case, the program error must be rectified and the program must be run again.
- The macros of SORTZM can be run in both write-protected (RDONLY=YES) and overwrite memory pages (RDONLY=NO).
  - If RDONLY=YES is specified (see the SRT0 and SRT1 macros), SORT also uses register 13. Consequently, the contents of this register must not be modified between succeeding SORTZM macros (if necessary, its contents must be saved and restored by the user).
  - If RDONLY=NO (default) applies, SORTZM sets up a 4-byte save area in the transfer control area and handles the saving and restoring of register 13 itself (see appendix, page 406).

Multiple sorting with SORTZM

SORTZM permits nesting of sort runs, with input records being passed to a number of different sort runs within a nested structure.
The corresponding macro calls may be specified in any order in the main program. The only constraint is that the macros for a given sort run must be specified in the proper sequence, i.e. SRTOPEN, SRTPUT, SRTGET and SRTCLSE. If the sequence of the macro calls is not correct (e.g. duplicate SRTOPEN, SRTPUT following SRTGET), SORT issues message SRT1308 and terminates the program run abnormally (see also the ERROR operand for SRTOPEN).
A separate input area for control information must be set up for each sort run (see also examples for SORTZM on page 366ff).
The operands of the SORTZM macros may be specified as positional or keyword operands.

5.3.2 Macros of the SORT access method SORTZM

The macros of earlier SORT versions (V7.0 to V7.3) are fully compatible with SORT V8.0 without needing to be recompiled; however, the correct sequence of the macros is not checked.
Macros as of version 7.4 can be used in SORT V8.0 without restrictions (except for the LINKADR operand).
For macros as of version 8.0 there is no guarantee that these macros can be run with older versions of SORT. When running SORT version V7.3 or lower unresolved external references occur.

5.3.3 SRTOPEN: initiate sorting

The SRTOPEN macro invokes the SORT program.

Name	Operation	Operands
[name]	SRTOPEN	[[SCB =] addr1 / (r1)] [,[ERROR =] addr2 / (r2) [,LINKADR = linkaddr / (r3)]

name — Symbolic name by which the macro is addressed (e.g. for use as a branch address).

SCB = addr1 / (r1) — Identifies the SVB input block.

addr1 — Symbolic address of the SVB input block.

(r1) — Register containing the address of the SVB input block.
$0 \leq r1 \leq 15$

If the SCB operand is not specified, r1=0 is used as the default value.

ERROR = addr2 / (r2)

Identifies the error routine to which a branch will be made if an error condition occurs in the SORT run.
If there is an error during execution and a branch is taken to the ERROR address, no SRTCLSE macro may subsequently be called for this sort run.

addr2 — Symbolic address of the error routine.

(r2) Register containing the address of the error routine.
$2 \leq r2 \leq 12$
The contents of register r2 must not be changed at any time during the sort run.

If the ERROR operand is not specified and an error is detected, SORT aborts the run with TERM MODE=ABNORMAL,UNIT=STEP.

LINKADR = linkaddr / (r3)
Specifies the address of the entry point for the access method. This is the entry point SORTZM1.

linkaddr Symbolic address of a word in which the user has stored the address of the entry point before execution of the macro.

(r3) Register containing the address of a word that refers to the address of the entry point.
$2 \leq r3 \leq 12$

If no LINKADR operand is specified, V(SORTZM1) is taken as the default value.

Note

If the LINKADR operand is used, it must be specified with **all** access method macros.

5.3.4 SRTPUT: pass record to SORT

The SRTPUT macro passes one record from the input area to SORT. One SRTPUT call is required for each input record to be transferred. No SRTPUT calls may be issued after the first SRTGET call in the sequence.

Name	Operation	Operands
[name]	SRTPUT	[[SCB =] addr1 / (r1)] [,[RECORD =] addr2 / (r2) [,LINKADR = linkaddr / (r3)]

name — Symbolic name by which the macro is addressed (e.g. for use as a branch address).

SCB = addr1 / (r1) — Identifies the SVB input block.

addr1 — Symbolic address of the SVB input block.

(r1) — Register containing the address of the SVB input block.
$0 \leq r1 \leq 15$

If the SCB operand is not specified, r1=0 is used as the default value.

RECORD = addr2 / (r2)
Identifies the record buffer in the main program from which the record is to be passed to SORT.

addr2 — Symbolic address of the record buffer.

(r2) — Register, containing the address of the record buffer.
$0 \leq r2 \leq 12$

If the RECORD operand is not specified, r2=1 is used as the default value.

LINKADR = linkaddr / (r3)
Specifies the address of the entry point for the access method. This is the entry point SORTZM1.

linkaddr — Symbolic address of a word in which the user has stored the address of the entry point before execution of the macro.

(r3)

Register containing the address of a word that refers to the address of the entry point.
$2 \leq r3 \leq 12$

If no LINKADR operand is specified, V(SORTZM1) is taken as the default value.

Note

If the LINKADR operand is used, it must be specified with **all** access method macros.

5.3.5 SRTGET: fetch record from SORT

The SRTGET macro accepts one record from SORT after sorting has been completed. One SRTGET call must be issued for each record in the SORT output area.

Name	Operation	Operands
[name]	SRTGET	[[SCB =] addr1 / (r1)] [,[RECORD =] addr2 / (r2) [,[EOS =] addr3 / (r3) [,LINKADR = linkaddr / (r4)]

name — Symbolic name by which the macro is addressed (e.g. for use as a branch address).

SCB = addr1 / (r1) — Identifies the SVB input block.

addr1 — Symbolic address of the SVB input block.

(r1) — Register containing the address of the SVB input block.
$0 \leq r1 \leq 15$

If the SCB operand is not specified, r1=0 is used as the default value.

RECORD = addr2 / (r2)
Identifies the record buffer in the main program from which the record is to be transferred from the SORT output area (move mode).

addr2 — Symbolic address of the record buffer.

(r2) — Register containing the address of the record buffer.
$0 \leq r2 \leq 12$

If the RECORD operand is not specified, the record from the SORT output area is made available in a special SORT record buffer whose address is supplied in register 1 (locate mode).

EOS = addr3 / (r3) — Identifies the routine to which a branch is to be made at the end of the record transfer operation.

addr3 — Symbolic address of the routine.

(r3) — Register containing the address of the routine.
$2 < r3 \leq 12$

If the EOS operand is not specified, the completion of the record transfer is signaled to the main program by an address value of zero in register 1.

LINKADR = linkaddr / (r4)

Specifies the address of the entry point for the access method. This is the entry point SORTZM1.

linkaddr

Symbolic address of a word in which the user has stored the address of the entry point before execution of the macro.

(r4)

Register containing the address of a word that refers to the address of the entry point.
$2 \leq r4 \leq 12$

If no LINKADR operand is specified, V(SORTZM1) is taken as the default value.

Note

If the LINKADR operand is used, it must be specified with **all** access method macros.

5.3.6 SRTCLSE: terminate sorting

The SRTCLSE macro terminates the sort operation. The SRTCLSE macro must not be called if SORT has returned to the main program via the ERROR exit.

Name	Operation	Operands
[name]	SRTCLSE	[[SCB =] addr / (r1)] [,LINKADR = linkaddr / (r2)]

name — Symbolic name by which the macro is addressed (e.g. for use as a branch address).

SCB = addr / (r1) — Identifies the SVB input block.

addr — Symbolic address of the SVB input block.

(r1) — Register containing the address of the SVB input block.
$0 \leq r1 \leq 15$

If the SCB operand is not specified, r=0 is used as the default value.

LINKADR = linkaddr / (r2)
Specifies the address of the entry point for the access method. This is the entry point SORTZM1.

linkaddr — Symbolic address of a word in which the user has stored the address of the entry point before execution of the macro.

(r2) — Register containing the address of a word that refers to the address of the entry point.
$2 \leq r2 \leq 12$

If no LINKADR operand is specified, V(SORTZM1) is taken as the default value.

Note

If the LINKADR operand is used, it must be specified with **all** access method macros.

5.3.7 Example

```
SRTZM    START
         .
         .
         .
SRTOPEN  SCB=B1,ERROR=BUGS ————————————————————————————————————— (1)
         .
         .
         .
RDLOOP   RDATA    INPT,BUGS ————————————————————————————————————— (2)
         SRTPUT   SCB=B1,RECORD=INPT
         B        RDLOOP
         .
         .
         .
OUTLOOP  SRTGET   SCB=B1,RECORD=OUTB,EOS=CLOSE ———————————————————— (3)
         B        OUTLOOP
CLOSE    SRTCLSE  SCB=B1 ——————————————————————————————————————————— (4)
         .
         .
         .
         TERM
BUGS     TERM MODE=ABNORMAL,UNIT=STEP
         .
         .
         .
INPT     DS       CL80 ————————————————————————————————————————— (5)
         .
         .
         .
OUTB     DS       CL80 ————————————————————————————————————————— (6)
         .
         .
         .
B1       SRT1     (SORT-RECORDS FIELDS=FIELD-EXPLICIT(POSITION=5,- ——— (7)
                  LENGTH=5)),SDF=YES
         SRT1     (SET-RECORD-ATTRIBUTES INPUT=VARIABLE(80))
         SRT1     (END)

         END      SRTZM
```

(1) The SRTOPEN macro calls SORT as a subroutine. Address B1 indicates where SORT can find the macros that contain the statements for this sort/merge run. BUGS is the address to which a branch is to be made if the SORT run terminates prematurely.

(2) This is the start of the read loop. The records to be sorted are read into the input area INPT (e.g. via RDATA). Then the record is passed from the input area to SORT by the SRTPUT macro. The SRTPUT call is repeated once for each record to be transferred to SORT.

(3) This is the start of the output loop. The SRTGET macro transfers the sorted records from SORT to the output area OUTB. The SRTGET call is repeated once for each record to be transferred. When all the records have been transferred (end condition EOS), a branch is made to the address of the SRTCLSE macro in order to terminate the SORT run.

(4) The SRTCLSE SCB=B1 terminates the SORT run defined under the symbolic address B1.

(5) Input area for the records to be sorted.

(6) Output area for the sorted records.

(7) SRT1 macros containing the statements for the B1 SORT run.

6 SORT user exits

The SORT sort/merge program provides a number of user exits at which the user can initiate certain actions.

The following user exits are available:

PLANNING	Activated when the planning phase has been completed and SORT has determined the sort strategy (cf. description on page 238).	Planning phase
INPUT	Activated when SORT accepts a record from the input area. The record can be updated, deleted or inserted. The structure of the interface is dependent on the PARAMETER-MODE operand of the ASSIGN-EXITS statement. The functional scope is virtually identical for the 24-bit and 31-bit addressing modes (cf. description on page 239).	Initial input
OUTPUT	Activated immediately before SORT writes to the output file. Output records can be validated, updated, inserted or deleted The structure of the interface is dependent on the PARAMETER-MODE operand of the ASSIGN-EXITS statement. The functional scope is virtually identical for 24-bit and 31-bit addressing modes (cf. description on page 244).	Final output
WORK-FILE-OVERFLOW	Activated if the capacity of the work file is exhausted (SORT) (cf. description on page 249).	Initial input
EXLST-FOR-INPUT	Allows a DMS EXLST macro to be connected for input (cf. description on page 251).	Preparatory phase
EXLST-FOR-OUTPUT	Allows a DMS EXLST macro to be connected for output (cf. description on page 253).	Output file processing
PHYSICAL-TRANSLATE	Specifies 2 code tables for conversion of the PHYSICAL-TRANSLATE format fields for determining a different sorting sequence (cf. description on page 255).	Initial input, internal merge, final output, merge run

VIRTUAL-TRANSLATE	Specifies a code table for VIRTUAL-TRANSLATE format fields for determining a different sorting sequence (cf. description on page 256).	Initial input, internal merge, final output, merge run
EXTERNAL-COMPARE	Activated for each record for sort fields with EXTERNAL-COMPARE as the sorting order so that the user routine can determine the sequence. The structure of the interface is dependent on the PARAMETER-MODE operand of the ASSIGN-EXITS statement. The functional scope is virtually identical for 24-bit and 31-bit addressing modes (cf. description on page 258).	Initial input, internal merge, final output, merge run
TRANSLATE-CHARACTER	Specifies 2 code tables for recoding the TRANSLATE-CHARACTER format fields for determining a different sorting sequence (cf. description on page 260).	Initial input, internal merge, final output, merge run
INT	Activated if the user enters a SEND-MESSAGE command to request information on the status of the sort/merge run or wants to have some other permitted action executed. In batch processing operation the command can also be entered by the operator at the console. In this case the task sequence number must be specified (cf. description on page 261).	All phases

Except for the INT user exit, the user can specify actions for all exits in the ASSIGN-EXITS statement. The user can connect user-written routines via the exits, enter into an interactive dialog with the sort/merge program or intervene in the SORT run at certain predetermined points in its execution.

One of the following actions must be specified in the user exit (provided it is a valid option for the user exit).

DIALOG

SORT is to report the current execution status via SYSOUT and request a further action via SYSDTA. When a user exit is reached, DIALOG causes SORT to output the associated message together with the actions permitted as a response. Specifying DIALOG is possible **in interactive mode only**. One of the following responses is possible, depending on the user exit concerned. At the INT exit it is also possible for a number of actions to be executed in succession.

C[ONTINUE] Processing is to continue.

F[INISH] SORT is to terminate record input and process the records already entered.

S[TART] The SORT run is to be restarted with corrected statements. Only the amended statements need to be entered.

T[ERMINATE] The SORT run is to be terminated.

The following actions are also possible via the INT exit (which cannot be specified in the ASSIGN-EXITS statement):

D[ISPLAY] SORT is to report on the processing status, i.e. the currently active phase of the SORT run, usage of CPU time, and the processing counters in use.

CK[PT] SORT is to write a checkpoint, if this is possible.

Notes

– If the user specifies no action for DIALOG in a procedure, processing is continued as if CONTINUE had been specified.

– Entries for the DIALOG actions can be abbreviated from the right as long the meaning remains clear.

– The entry of DIALOG in a batch procedure is ignored, processing is continued.

FINISH-INPUT

SORT is to terminate record input and process the records already entered.

MODULE (NAME=<text 1..8>)

SORT is to link in a user routine and call it at the specified exit. The user routine must be stored as an object module in the TASKLIB used or in a library assigned with the file link name SORTMODS.

Register conventions

– All registers to be used in a user routine must first be saved (e.g. via SAVE (14,12) or STM 14,12,12(13)).

– Before control is returned from the user routine to the sort/merge program, the saved registers (except for registers 1 and 15) have to be reloaded, e.g. using RETRN (14,12),RC=8.

– The following registers have a fixed usage:

Reg 1 When a branch is made to a user routine, register 1 contains the address of an operand list with record addresses and possibly flags.
When control is returned to the sort/merge program, register 1 must have the following contents:
  – with the INPUT user exit, the address of the processed record
  – with the OUTPUT and EXTERNAL-COMPARE user exits, the address of the corresponding input area.

Reg 13 points to an 18-word save area in the sort/merge program.

Reg 14 contains the return address to the sort/merge program. The branch to the user routine is effected using BALR 14,15.

Reg 15 When a branch is made to a user routine, register 15 contains the address of the entry point into the routine.
When PARAMETER-MODE=24 is used, with the return of control to the sort/merge program the rightmost byte of register 15 contains the return code (this applies to user exits INPUT, OUTPUT and EXTERNAL-COMPARE). The remaining bytes must be set to zero. When PARAMETER-MODE=ANY is used, register 1 points to the parameter area. The fourth pointer in this area points to the action word in which the return code is stored.

	Data must be passed to SORT from the program address area. If the STXIT=YES operand is specified in the associated SORT macro (SRT0, SRT1), SORT checks the validity of the data and addresses sent back by the user routine. If a program error is detected, a corresponding message is displayed and the sort run is terminated. In this case, the program error must be rectified and the sort run must be repeated.
TERMINATE-ABNORMAL	The sort run is to be terminated.

6.1 PLANNING: planning completed

The PLANNING exit is activated when all the information for the sort/merge run has been evaluated and an execution strategy determined. The user can make use of the PLANNING exit for optimizing a SORT run by specifying the action DIALOG. Then, after SORT has checked the allocated resources, the run can be restarted with the improved statements.

The PLANNING exit allows one of the following actions to be specified in the ASSIGN-EXITS statement.

DIALOG — SORT outputs the following messages (only with MIN-MSG-WEIGHT=*ALL):

SRT1031	estimated size of the work file
SRT1033	intensively used storage
SRT1050	block size of work and auxiliary file
SRT1061	maximum number of merge runs

In batch mode, processing is continued with CONTINUE as the action. In interactive mode, the following actions are permitted:

C[ONTINUE] Processing is to continue.

S[TART] A restart of the SORT run is to take place using the improved statements. Only the statements that have been modified need to be reentered.
START may only be specified for runs with SORT as a standalone program or when SORT is called as a subroutine at level 0.

T[ERMINATE] The SORT run is to be terminated.

TERMINATE-ABNORMAL — The SORT run is to be terminated.

6.2 INPUT: input record processing

The INPUT user exit can be used to monitor and control the record input to the sort/merge program. Records can be validated, modified, inserted or deleted. The user can also route the entire input via INPUT. A branch is made to INPUT each time an input record is to be passed to SORT.

The structure of the interface is dependent on the PARAMETER-MODE operand of the ASSIGN-EXITS statement.

Interface to the user routine when PARAMETER-MODE=ANY

In 31-bit addressing mode, the user exit also offers an extension in the form of the user constants and the name of the coded character set. When the sort/merge program passes control to the user routine, register 1 contains the address of a 20-byte input area with the following structure:

Bytes 0 - 3: Address of the next input record

To insert or modify an input record, the user must pass its address in this field. The address of the next input record is set to zero when:

- the end of the input file is sensed
- the end of a file sequence is sensed (multi-file sort)
- the input file is missing.

Bytes 4 - 7: Address of the user constant

The address of the user constant points to a 4-byte area via which the user can pass information to the OUTPUT user exit (e.g. the address of a dynamically requested memory area).

Bytes 8 - 11: Address of the file identifier

The address of the file identifier points to a 4-byte area. SORT supplies the identifier of the current file in this area (right-justified) when a record is inserted.
The file identifier of the input file is specified in binary and corresponds to the consecutive number of SORTINxx or MERGExx.
The file identifier is provided only for the information of the user.
On the return to the sort program, this field is not evaluated.

Bytes 12 - 15: Address of the action word

When the user routine returns control to SORT, the rightmost byte of the action word must contain one of the following return codes:

X’00’ SORT is to fetch the input record. Bytes 0 - 3 of the input area must contain the address of the input record. This record may be the record taken from the input stream or a record that has been modified.
If a record is to be extended, the user must provide a separate area for this purpose.
With records of variable lengths, the length field (the first four bytes of the record) may not be modified in the original area. If the length of variable records is to be modified, then a separate record area must be made available, even when shortening.

X’04’ The record whose address is contained in bytes 0 - 3 of the input area is to be deleted. This return code is not valid if the address in the input area is set to zero, i.e. input has been completed or no input file is present.

X’08’ This return code must be specified by the user if no further branches are to be made to the user routine. Code X’08’ is required at the end of the entire input except where an enforced finish is indicated by X’14’, or abortion of the SORT run by X’10’.
If code X’08’ is set before input is complete, SORT reads in the remaining input records but does not call the user routine.

X’0C’ The record whose address is contained in bytes 0 - 3 of the input area is to be inserted. It is also possible to insert records even when the address in the input area is set to zero when the user routine is called, i.e. the input has been completed or there is no input file present. X’0C’ must always be set when the user is personally handling the input.

X’10’ This causes the SORT run to be aborted. If SORT was called as a subroutine, a return to the calling program is made with error code X’FF’.

X’14’ Input is forced to a close. In a sort run this refers to all the files in the sequence, i.e. X’14’ also implies return code X’08’.
In a merge run only the merge input file referenced by the file identifier supplied by SORT is closed. The merge run then continues with the remaining merge input files. X’14’ for the last merge input file also implies X’08’.

Bytes 16 - 19: Address of the CCSN

This address refers to an area of 8 bytes in size in which SORT stores the name of the coded character set of the data.
The name is for the user's information only. This field is not evaluated when returning to the sort program.

Notes

– When control is returned to the sort/merge program with return code X'00' or X'0C' set, bytes 0 - 3 of the input area must contain the address of the record to be transferred or else a zero address.
In sort runs, return code X'00' or X'0C' combined with a zero address in the input area results in the current input file being closed. In a file sequence, the current input file is closed and processing continues with the next. In merge runs, a zero address causes SORT to close the input file referenced by the supplied file identifier (same effect as return code X'14'). The merge run then proceeds with the remaining merge input files.

– The INPUT user exit must not be used for inserting records during tag sorting.

– Processing of the transferred record is initiated while it is still in the user's record transfer area; the record undergoes some modification (conversion) at the same time.

– When inserting new records and extending existing ones the user must ensure that a separate record area is made available.

– With records of variable lengths, the length field (the first four bytes of the record) may not be modified in the original area. If the length of variable records is to be modified, then a separate record area must be made available, even when shortening.

Interface to the user routine when PARAMETER-MODE=24

When the sort/merge program passes control to the user routine, register 1 points to a 4-byte area with the following structure:

Byte 0: File identifier

This byte contains the file identifier of the input file from which the input record was read. When a record is inserted, SORT puts the identifier of the current file in this byte. The identifier is specified in binary and corresponds to the consecutive number xx in SORTINxx or MERGExx.
The file identifier is provided only for the information of the user.
On the return to the sort program, this field is not evaluated.

Bytes 1 - 3: Address of the next input record

To insert or modify an input record, the user must pass its address in this field. The address of the next input record is set to zero when:

- the end of the input file is sensed
- the end of a file sequence is sensed (multi-file sort)
- the input file is missing.

When the user routine returns control to SORT, one of the following return codes must be passed in the rightmost byte of register 15:

X'00' SORT is to fetch the input record. The address of the record must then be loaded into register 1. This record may be a record taken from the input stream or a record that has been modified.
If a record is to be extended, the user has to provide a separate area for this purpose.
With records of variable lengths, the length field (the first four bytes of the record) may not be modified in the original area. If the length of variable records is to be modified, then a separate record area must be made available, even when shortening.

X'04' This causes the record specified in the input area to be deleted.

X'08' This return code must be specified by the user if no further branches are to be made to the user routine.
Code X'08' is required at the end of the complete input except where an enforced finish is indicated by X'14', or abortion of the SORT run by X'10'.
If code X'08' is set before input is completed, SORT reads in the remaining input records but does not call the user routine.

X'0C' The record whose address is contained in register 1 is to be inserted. It is also possible to insert records even when the address in the input area is set to zero, i.e. the input has been completed or there is no input file present. X'0C' must always be set when the user personally handles the input.

X’10’ This causes the SORT run to be aborted. If SORT was called as a subroutine, a return to the calling program is made with error code X’FF’.

X’14’ Input is forced to a close. In a sort run this refers to all the files in the sequence, i.e. X’14’ also implies return code X’08’.
In a merge run only the merge input file referenced by the file identifier supplied by SORT is closed. The merge run then continues with the remaining merge input files. X’14’ for the last merge input file also implies X’08’.

Notes

– In sort runs, return code X’00’ or X’0C’ combined with a zero address in register 1 results in the current input file being closed. In a file sequence, the current input file is closed and processing continues with the next. In merge runs, a zero address causes SORT to close the input file referenced by the file identifier (same effect as return code X’14’). The merge run then proceeds with the remaining merge input files.

– The INPUT user exit must not be used for inserting records during tag sorting.

– Processing of the transferred record is initiated while it is still in the user’s record transfer area; the record undergoes some modification (conversion) at the same time.

– When introducing new records and lengthening existing ones the user must ensure that a separate record area is made available.

– With records of variable lengths, the length field (the first four bytes of the record) may not be modified in the original area. If the length of variable records is to be modified, then a separate record area must be made available, even when shortening.

6.3 OUTPUT: output record processing

The OUTPUT user exit can be used to monitor and control the output of records from the sort/merge program. By this means the user can have records validated, modified, inserted or deleted by a user routine. It is also possible to combine records with identical sort fields (compaction, summation record formation). The user routine connected via OUTPUT can also handle the entire output.

With PARAMETER-MODE=24, activating the OUTPUT exit before the very first output from the sort/merge program is effected by supplying a zero value. In this way the desired type of record processing can be indicated using a flag byte. Subsequently OUTPUT is activated each time a record is to be written to the output file.

With PARAMETER-MODE=ANY, the OUTPUT exit is activated prior to each write of a record to the output file.

The structure of the interface is dependent on the PARAMETER-MODE operand of the ASSIGN-EXITS statement.

Interface to the user routine when PARAMETER-MODE=ANY

In 31-bit addressing mode, the user exit offers an extension in the form of the user constants and the name of the coded character set. The following functions cannot be performed at the user exit:

– Sequence check control.
  This function can be effected using the SEQUENCE-CHECK parameter in the SET-SORT-OPTIONS statement.

– Duplicate records indication.
  This function is contained in the SUM-RECORDS statement.

When the sort/merge program passes control to the user routine, register 1 contains the address of a 20-byte input area with the following structure:

Bytes 0 - 3: Address of the next output record

These bytes contain the address of the next record that will be output by SORT from the SORT record buffer. When the last output record is in the DMS output buffer, the bytes contain a zero address.
On returning control with a return code of X’00’ or X’0C’ in the action word, the user routine must enter the record address in this field.

Bytes 4 - 7: Address of the current output record

These bytes contain the address of the current output in the DMS output buffer.
The address is zero when the first record is processed. The address specified here must not be modified.

Bytes 8 - 11: Address of the user constant

The address of the user constant points to a 4-byte area from which the user can take any information that may have been transferred by the INPUT user exit.

Bytes 12 - 15: Address of the action word

When the user routine returns control to SORT, the rightmost byte of the action word must contain one of the following return codes.

X'00' The record whose address is contained in bytes 0 - 3 of the input block is to be transferred to the output stream.
This can be the record in its original form, a modified record or a replacement record. If the record is to be extended, the user must make it available in a separate area.

X'04' The record whose address is stored in bytes 0 - 3 of the input area is to be deleted. A return code of X'04' is not allowed if the address is zero.
X'04' must always be set if the user personally takes responsibility for handling the output.

X'08' No more calls are to be made to the user routine. This code must be set at the end of the output phase, except where return code X'10' is passed to indicate that processing is to be aborted. If X'08' is set before the end of the output phase, SORT outputs the remaining records without making a further call to the user routine.

X'0C' The record whose address is contained in bytes 0 - 3 of the input area is to be inserted. Records can be inserted even when bytes 0 - 3 contain a zero address.

X'10' The SORT run is to be terminated. If SORT was called as a subroutine, error code X'FF' is set when control is returned to the calling program.

Bytes 16 - 19: Address of the CCSN

This address refers to an area of 8 bytes in size in which SORT stores the name of the coded character set of the data.
The name is for the user's information only. This field is not evaluated when returning to the sort program.

Notes

– If return code X'00' or X'0C' is set when control is returned to the sort/merge program, then bytes 0 - 3 of the input area must contain the address of the record to be transferred or zero. A zero address results in termination of the sorting process (possibly also with forced termination of the output). In this case the effect of return code X'08' is implicit in the action.

– The two records specified in the input area are available to the user routine in the output record format.

– The user has to provide a separate area for any records that are to be inserted or extended.

Interface to the user routine when PARAMETER-MODE=24

When SORT passes control to the user routine, register 1 contains the address of an 8-byte input area with the following structure:

Byte 0: Continuation flag

This byte indicates whether the sort fields of the two records specified are identical. If so, a sum record can be formed, for example. If sort fields are to be eliminated by means of the ELIMINATE operand, no flag is set. The continuation flag is set only if an output file is present.

X'00' is set if the sort fields are not identical.

X'04' is set if the sort fields of two consecutive records are identical.

Bytes 1 - 3: Address of the next output record

These bytes contain the address of the next record that SORT is to output from the SORT record buffer. When the final output record is in the DMS output buffer, this address is zero.
When the user routine returns control to SORT with return code X'00' or X'0C', register 1 must contain this address, another record address or zero.

Byte 4: Flag byte

When the OUTPUT user routine is called for the first time, the type of record processing must be specified in this byte.

X'00' The user exit is to be activated for each output record without a sequence check being performed.

X'04' The exit is to be activated for each output record and a sequence check is to be performed.

X’10’ The exit is to be activated for each output record and a sequence check is to be performed. In addition, the continuation flag byte is to indicate whether the records have identical sort fields, except in the case of sort fields with the ELIMINATE operand.

Note

Sequence checking specifications in the SET-SORT-OPTIONS statement (SEQUENCE-CHECK operand) take precedence over identifier byte settings.

Bytes 5 - 7: Address of the current output record

These bytes contain the address of the current output record in the DMS output buffer.
The address is zero when the first record is processed or there is no output file present. The address specified here must not be modified.

Conventions for return of control to the sort/merge program when PARAMETER-MODE = 24

- First call to the OUTPUT user exit
  When the OUTPUT user routine is called for the first time, the desired type of record processing must be specified in the flag byte of the input area. The remaining bytes of the input area must be set to zero. Register 15 is checked for the presence of return code X’08’. Register 1 has no significance here. The flag byte must not be modified at any time during processing.
- Further calls to the OUTPUT user exit
  When the user routine returns control to SORT, registers 1 and 15 must be loaded with the appropriate value.

  Reg 15 The rightmost byte must contain one of the following return codes

  X’00’ The record whose address is contained in register 1 is to be moved to the output area. The record may be the original input record, a modified record or a replacement. If the record is to be extended, the user must make it available in a separate area.

  X’04’ The record whose address is contained in bytes 1-3 of the input is to be deleted. Return code X’04’ is not allowed if the address is zero.
  X’04’ must always be set if the user personally handles output processing.

X’08’ No more calls are to be made to the user routine. This code must be set at the end of the output phase, except where return code X’10’ is passed to indicate that processing is to be aborted. If X’08’ is set before the end of the output phase, SORT outputs the remaining records without making a further call to the user routine.

X’0C’ The record whose address is contained in register 1 is to be inserted. Records can be inserted even when bytes 1-3 of the input area contain a zero address.

X’10’ The SORT run is to be terminated. If SORT was called as a subroutine, error code X’FF’ is set when control is returned to the calling program.

Reg 1 If return code X’00’ or X’0C’ is set when control is returned to the sort/merge program, register 1 must contain the address of the record to be transferred or zero.
A zero address in register 1 results in termination of the sort/merge run (possibly also entailing forced termination of the output). In this case the action also implies the effect of return code X’08’.

Notes

– With the flag byte set to X’10’ in a run in which records with identical sort fields are to be compacted (cumulation of sum fields), it is simplest for the user to proceed as follows:

  Add the sum fields of the 1st record (address in bytes 1 - 3 of the input area) to the 2nd record (address in bytes 5 - 7 of the input area). Record processing can then be continued, e.g. with an overflow check. Following this, return code X’04’ (delete record) must be set and a return made to the sort/merge program. Register 1 has no significance in this case.

– The records specified in the input area (record addresses in bytes 1 - 3 and 5 - 7) are available to the user routine in the output format.

– X’04’ (delete record) refers to the record address in bytes 1 - 3 of the input area.

– X’00’ or X’0C’ (fetch or insert record) refers to the record address in) register 1.

– Records that are to be inserted or extended must be made available in a separate memory area.

6.4 WORK-FILE-OVERFLOW

The WORK-FILE-OVERFLOW exit is activated if a disk work file is about to overflow and the secondary allocation is zero, SORT is unable to remove the bottleneck and no auxiliary file is available for a further cycle.
The user can abort the run or limit the sorting operation to the records already accepted by SORT.

DIALOG — The number of records read by SORT up to this point is indicated by "`SRT1017 RECORDS TO BE SORTED/MERGED: ...n`"; one of the following actions is expected as a response:

- **C**[ONTINUE] — SORT tries to perform the sort run with more limited resources.
- **F**[INISH] — SORT terminates record input and processes the records already accepted.
- **T**[ERMINATE] — SORT terminates the run.

FINISH-INPUT — Record input is terminated and SORT processes the records read in up to this point.

MODULE (NAME=<name 1..8>, INTERFACE-VERSION=1/2)

A user routine is connected. Register 1 contains the address of an 8-byte input area.

Bytes 0 - 3: INTERFACE-VERSION=1:
Number of records already read in. If this number is greater than 2.147.483.647, then it is set to that value, which can lead to malfunctions in the user exit.

INTERFACE-VERSION=2:
Address of an 8-byte area containing the number of records already read in

Byte 4: File identifier
This byte provides information on the file type.
X'01' work file
X'02' auxiliary file

Byte 5: File sequence number
This byte contains the sequence number from the file link name SORTWKx or SORTWKxx.

Bytes 6 - 7: Size of the work file

Register conventions

On returning control to SORT, the user routine loads one of the following codes into the rightmost byte of register 15:

X'00' SORT is to continue processing (this can lead to an abort because of input/output errors).

X'04' Input is terminated and the records already entered are sorted (FINISH action).

X'08' The sort run is terminated because of an error (TERMINATE action).

6.5 EXLST-FOR-INPUT EXLST: exit for input files

The EXLST-FOR-INPUT user exit enables the user to specify EXLST exits for the input file(s). The listed routines can then be made available to SORT in accordance with DMS conventions.

At the start of this module is an EXLST macro containing references to the user routines for the individual DMS exits used. The operands of the EXLST macro, which are passed on to DMS, are summarized in the following table (for a detailed description see the “DMS Macros” manual [3]).

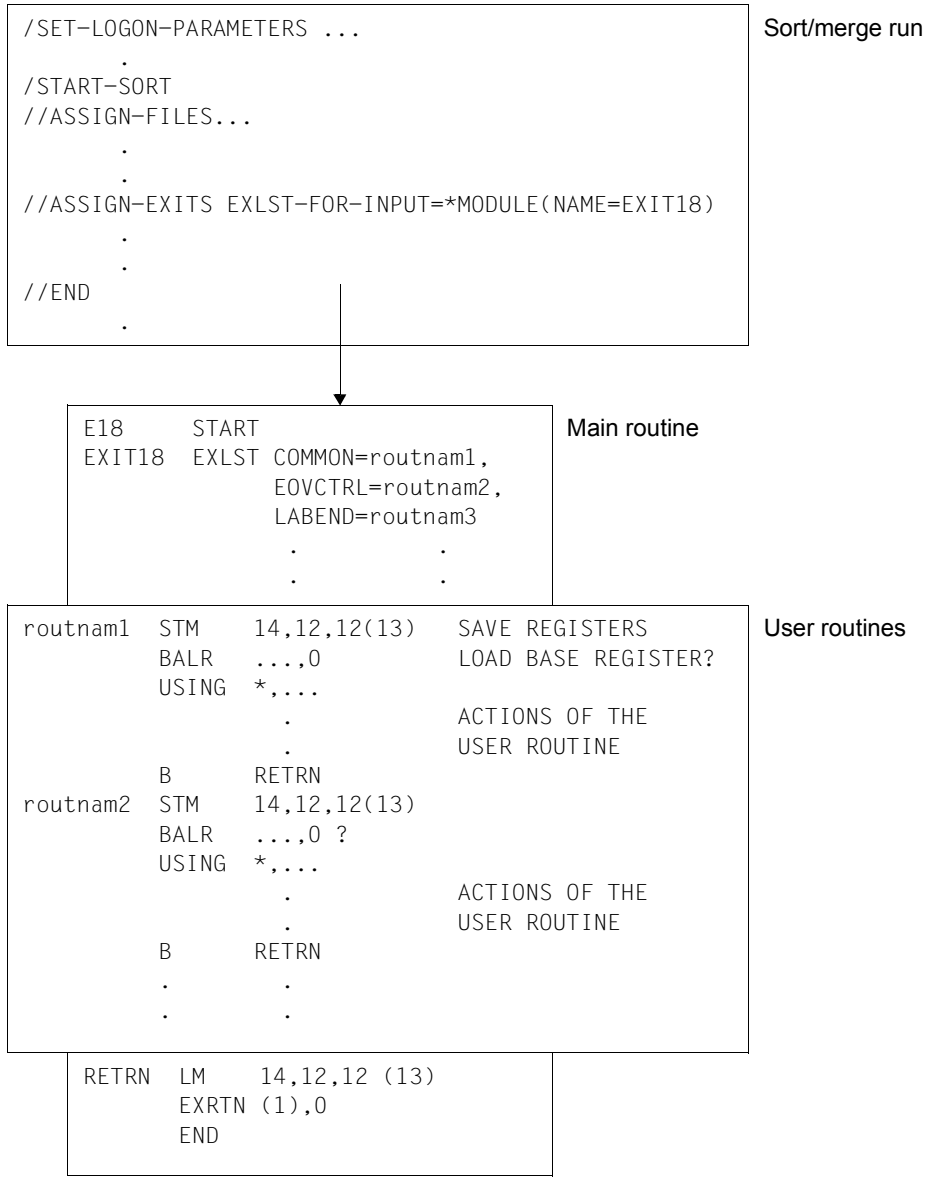
COMMON	Tape/disk	All exits may be specified that are also covered by COMMON, except for EOFADDR and USERERR.
EOVCTRL	Tape	This exit closes label processing after a new data volume is made available.
ERRADR	Tape/disk	Control is passed to this exit if a hardware error has occurred or an input/output has terminated in error.
ERROPT	Tape/disk	This exit is useful for SAM files and connects to routines that are to be executed if a defective block is detected.
LABEND	Tape	This exit checks user labels for end of file (EOF).
LABEOV	Tape	This exit checks user and end of tape (EOT) labels.
LABGN	Tape	This exit is used for checking user labels that precede the data of the input files.
OPENV	Tape	This exit is used for data volumes with non-standard labels.
PGLOCK	Disk	Control is passed to this exit if another job has caused locks to be put on the desired data.

Notes

– When control is passed to the user routine, register 1 contains the address of the FCB for the file.

– An EXLST macro is accepted with either PARMOD=24 or PARMOD=31 when SORT is called in 24-bit addressing mode. PARMOD=31 is, however, mandatory when SORT is called in 31-bit addressing mode.

Example



Note

The return operation using EXRTN is only permitted for some of the EXLST routines (see the EXRTN macro section of the “DMS Macros” manual [3]).

6.6 EXLST-FOR-OUTPUT: user exit for output files

The EXLST-FOR-OUTPUT user exit allows the user to specify EXLST exits for the output file. The routines listed in the exits can then be made available to SORT in accordance with DMS conventions.

At the start of the module is an EXLST macro containing references to the user routines for the individual DMS exits used. The operands of the EXLST macro, which are passed on to DMS for the output file used, are summarized in the following table (for a detailed description see the “DMS Macros” manual [3]).

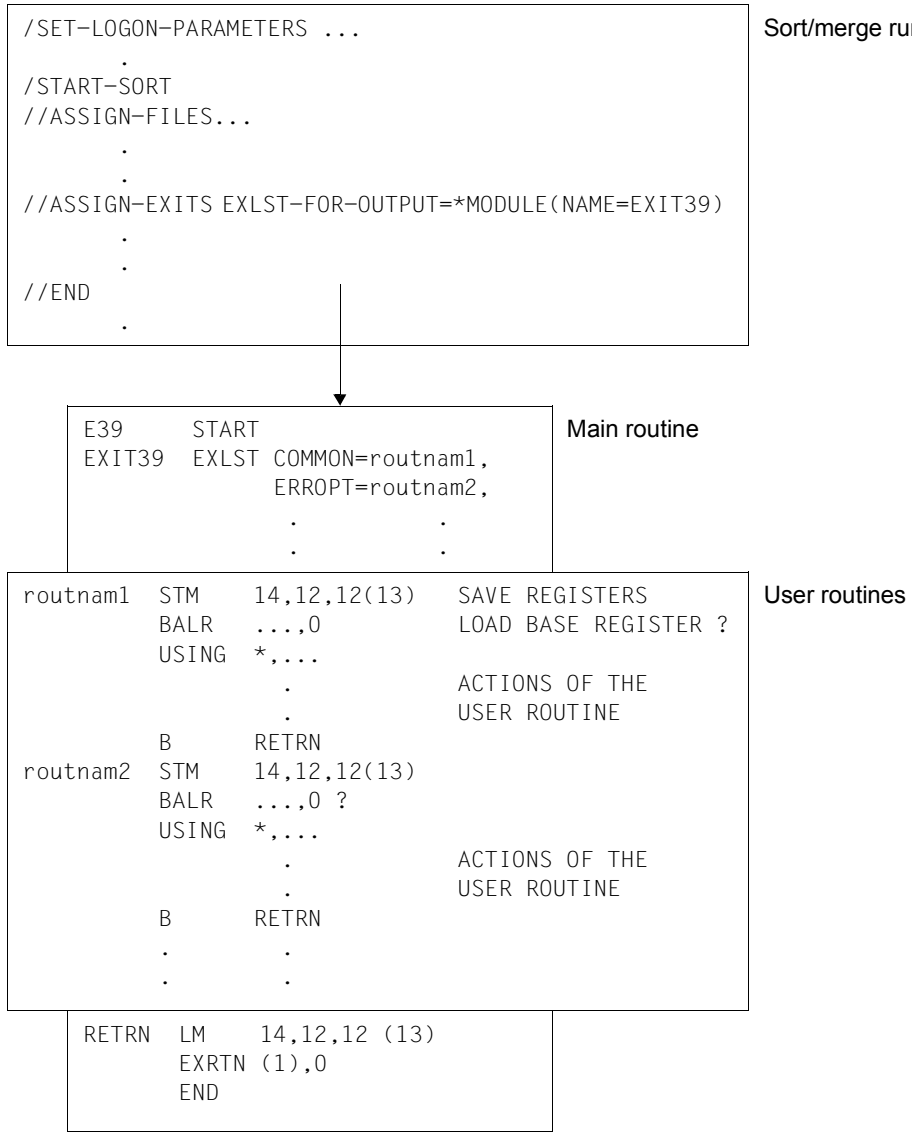
COMMON	Tape/disk	All exits may be specified that are also covered by COMMON.
EOVCTRL	Tape	This exit closes label processing after a new data volume is made available.
ERRADR	Tape/disk	Control is passed to this exit if a hardware error has occurred or an input/output has terminated in error.
ERROPT	Tape/disk	This exit is useful for SAM files and connects to routines that are to be executed if a defective block is detected.
LABEND	Tape	This exit checks user labels for end of file (EOF).
LABEOV	Tape	This exit checks header labels, which are at the end of the tape reel in the case of output files.
LABGN	Tape	This exit is used to check user labels that precede the data of output files, or to generate user header labels (UHLs) for output files.
OPENV	Tape	This exit is used for data volumes with non-standard labels.

Notes

– When control is passed to the user routine, register 1 contains the address of the FCB for the file.

– An EXLST macro is accepted with either PARMOD=24 or PARMOD=31 when SORT is called in 24-bit addressing mode. PARMOD=31 is, however, mandatory when SORT is called in 31-bit addressing mode.

Example



Note

The return operation using EXRTN is only permitted for some of the EXLST routines (see the EXRTN macro section of the “DMS Macros” manual [3]).

6.7 PHYSICAL-TRANSLATE: special character conversion table

This exit allows a user routine to be connected for converting sort fields in PHYSICAL-TRANSLATE format.

The user routine is not actively invoked; it consists simply of two 256-character code conversion tables. The first code table (relative address 0) is used for the initial conversion, and the second (relative address 256) is for converting the characters back at the end of sorting.

Table 1 — This table is used for converting PHYSICAL-TRANSLATE format fields before the comparisons are performed.

Table 2 — This table is used for subsequent reconversion of the fields converted using Table 1. In this way the original contents are restored once processing has been completed.

The position of the characters in the tables determines the association of argument and function value for the conversion.

Table 1: User table

	0	1	2	…	E	F
0	xx	xx	xx	…	xx	xx
1	xx	E2	xx	…	xx	xx
2	xx	xx	xx	…	xx	xx
⋮	⋮	⋮	⋮	⋮	⋮	⋮
E	xx	xx	xx	…	xx	xx
F	xx	xx	xx	…	xx	xx

Table 2: EBCDIC table

	0	1	2	…	E	F
0	xx	xx	xx	…	xx	xx
1	xx	xx	xx	…	xx	xx
2	xx	xx	xx	…	11	xx
⋮	⋮	⋮	⋮	⋮	⋮	⋮
E	xx	xx	xx	…	xx	xx
F	xx	xx	xx	…	xx	xx

xx: Character in hexadecimal notation

The character represented by hexadecimal 11 in the EBCDIC table is assigned to position E2 in the user table. All bytes containing X’11’ in the PHYSICAL-TRANSLATE fields are converted to X’E2’. Upon completion of processing by SORT, X’E2’ is converted back to X’11’.

Note

The conversion tables need to be constructed with great care in order to achieve the desired sorting sequence. This is particularly important when the original code is to be unequivocally restored on reconversion. It may be easier to produce a changed sequence via conversion formats. The MODIFY-CODE statement also provides a simple conversion facility, and the VIRTUAL-TRANSLATE format using the VIRTUAL-TRANSLATE code table is useful for special conversions (combining several code characters).

6.8 VIRTUAL-TRANSLATE: special character conversion table

This exit enables sort fields in VIRTUAL-TRANSLATE format to be sorted according to a coded order of priority. Unlike the PHYSICAL-TRANSLATE exit, it entails no changes to the sort fields themselves.

A 256-byte code table is specified as the user routine for converting the VIRTUAL-TRANSLATE format fields to an auxiliary field for each comparison. This has the advantage that no conversion back to the original code is necessary and a number of characters can be converted to a new character.

The user routine is not actively invoked; it consists simply of a 256-byte code table with an address of 0 relative to the start of the routine (object module).

Example for an user table:

	0	**…**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	00	…	00	00	00	00	00	00	00	00
1	00	…	0B	14	00	00	0B	14	00	00
2	00	…	0C	15	1D	00	0C	15	1D	00
3	00	…	0D	16	1E	00	0D	16	1E	00
4	00	…	0E	17	1F	00	0E	17	1F	00
5	00	…	0F	18	20	00	0F	18	20	00
6	00	…	10	19	21	00	10	19	21	00
7	00	…	11	1A	22	00	11	1A	22	00
8	00	…	12	1B	23	00	12	1B	23	00
9	00	…	13	1C	24	00	13	1C	24	00
A	00	…	00	00	00	00	00	00	00	00
B	00	…	00	00	00	00	00	00	00	00
C	00	…	00	00	00	00	00	00	00	00
D	00	…	00	00	00	00	00	00	00	00
E	00	…	00	00	00	00	00	00	00	00
F	00	…	00	00	00	00	00	00	00	00

This user table causes all capitals and lowercase letters to be treated as equal during sorting. For example, ’A’ (X’C1’) and ’a’ (X’81’) are both assigned to X’0B’ in the user code table and therefore are treated as identical during sorting (see example for VIRTUAL-TRANSLATE, page 353). A table of this type can be built using an Assembler CSECT, which can then be specified as a MODULE(NAME=<name=1..8>) action in the ASSIGN-EXITS statement.

6.9 EXTERNAL-COMPARE: sequence defined by user routine

The EXTERNAL-COMPARE exit allows the user to define ascending or descending order for each sort field comparison, provided that the EXTERNAL-COMPARE option has been specified as the sorting order for the fields in the FIELDS operand of the SORT-RECORDS statement.
The EXTERNAL-COMPARE user exit may be used for 24-bit or 31-bit addressing. The interface is set up in accordance with the PARAMETER-MODE operand of the ASSIGN-EXITS statement.

Interface to the user routine when PARAMETER-MODE=ANY

The user routine is called each time two EXTERNAL-COMPARE sort fields are compared. In 31-bit addressing mode, the user exit offers an extension in the form of the name of the coded character set, the address of the internal record format and the address of the internal sort field position.

When the sort/merge program transfers control to the user routine, register 1 contains the address of a 32-byte input area with the following structure:

Bytes 0 - 3: Address of the EXTERNAL-COMPARE sort field in the first match record

Bytes 4 - 7: Address of the EXTERNAL-COMPARE sort field in the second match record

Bytes 8 - 11: Address of the sequence number of the sort field

This address points to a data area 4 bytes in length in which relevant information is stored in the rightmost byte.

Bytes 12 - 15: Address of the sort field length

This address points to a data area 4 bytes in length in which relevant information is stored in the rightmost byte.

Bytes 16 - 19: Address of the action word

When the user routine returns control to SORT, the rightmost byte of the action word must contain one of the following return codes:

X’00’ The sort field of the 1st match record has priority.

X’04’ The two sort fields have equal priority.

X’08’ The sort field of the 2nd match record has priority.

Bytes 20 - 23: Address of the internal record format

This address points to a data area 4 bytes in length in which the following information is stored in the rightmost byte.

X'02' variable record format

X'04' fixed record format.

The record format is only for the user's information. This field is not evaluated when returning to the sort program.

Byte 24 - 27: Address of the internal sort field position

This address points to a data area 4 bytes in length in which the distance between the sort field and the beginning of the record is stored in its two rightmost bytes. The address of the sort field is stored in the input area in bytes 0 - 3. The sort field position is only for the user's information. This field is not evaluated when returning to the sort program.

Byte 28 - 31: Address of the CCSN

This address points to an area 8 bytes in length in which SORT stores the name of the coded character set which is used to process the data records. The name is only for the user's information in order to effectively compare the sort fields. This field is not evaluated when returning to the sort program.

Note

The user routine must not modify the sort fields.

Interface to the user routine when PARAMETER-MODE=24

The user routine is called each time two EXTERNAL-COMPARE sort fields are compared. Register 1 points to the following 8-byte input area:

Byte 0: Sequence number of the sort field

Bytes 1 - 3: Address of the EXTERNAL-COMPARE sort field in the first match record

Byte 4: Sort field length

Bytes 5 - 7: Address of the EXTERNAL-COMPARE sort field in the second match record

The user routine returns control to SORT after comparing the two EXTERNAL-COMPARE sort fields. It must also load one of the following return codes into the rightmost byte of register 15:

X’00’ The sort field of the 1st match record has priority.

X’04’ The two sort fields have equal priority.

X’08’ The sort field of the 2nd match record has priority.

Note

The user routine must not modify the sort fields.

6.10 TRANSLATE-CHARACTER: sequence defined by equating tables and coded character set

This exit allows a user routine to be connected for converting sort fields in TRANSLATE-CHARACTER format when the CCSN of the input file matches the name of the specified module. The user routine consists of two 256-character code conversion tables. SORT uses these tables to generate the code of the character to be sorted for the sort operation. These tables take precedence over the standard tables supplied by SORT in the library with the logical name SYSLNK.TAB. Otherwise the tables are treated like standard tables. It is not necessary for a standard table with the same name to exist. However, XHCS must know the CCSN.
If the file which is to be sorted has a CCSN which is not contained in the statement but is contained in the library with the logical name SYSLNK.TAB, the file in this library is accessed.

6.11 INT: sort/merge run interrupt

The INT exit enables a user at a terminal to hold an interactive dialog with the SORT sort/merge program. The INT exit cannot be specified in the ASSIGN-EXITS statement. This means that it cannot be used to connect to a user routine.

The INT exit is activated

– in interactive mode: by first switching from program mode (SORT) to system mode; this is done by pressing the K2 key of the terminal from which SORT was started. When the system displays the slash prompt, the command /INFORM-PROGRAM can be entered.

– in batch mode: by entering /INFORM-PROGRAM JOB-ID=*TSN(TSN=tsn) at the console (where <tsn> is the number of the task in which SORT is running).

This is followed by a message to the terminal indicating the elapsed runtime and CPU time and requesting one of the following actions:

D[ISPLAY]	Show the processing status, e.g. the number of records processed so far.
CK[PT]	Write a checkpoint at the earliest possible time. If, for example, the TERMINATE action is selected next, SORT takes a checkpoint before terminating the sort/merge run.
C[ONTINUE]	Continue processing.
F[INISH]	Terminate record input to SORT and process (sort/merge) the records already entered.
T[ERMINATE]	Terminate the SORT run.

Notes

– After a DISPLAY or CKPT action has been executed, a further action is requested. If no checkpoint could be written following the CKPT and TERMINATE actions, SORT issues an error message and continues with the sort/merge processing.

– SORT cannot be addressed via the INT user exit if it has been called under the SORTZM access method or as a subroutine and the operand STXIT=NO is specified in the *first* SRT0 or SRT1 macro.

Example

```
/SET-LOGON-PARAMETERS ...
  .
  .
  .
/START-SORT
  .
  .
  .
//SORT-RECORDS...
  .
  .
//END
  .
  .
  .
```

← K2 key or ESCAPE/BREAK key

```
/SEND-MESSAGE TO=PROGRAM
```

Enter desired action, e.g. CKPT

```
  .
  .
  .
```

7 Checkpoint processing

Checkpoints are written by SORT during sort/merge runs when either of the following conditions applies:

– the CHECKPOINT operand has been specified in the SORT-RECORDS or MERGE-RECORDS statement

– a SEND-MESSAGE call is issued in interactive mode and CKPT entered as the desired action (in this case the checkpoint is written at the earliest possible time).

No checkpoints are written if SORT is called as a subroutine and the user does not comply with the restrictions of the CHKPT macro. (For information about using a stack, a memory pool or inter-task communication (ITC) see also “Executive Macros” [6], WRCPT macro.)

No checkpoints can be written during processing of POSIX files as input files either.

In multi-task sorts, checkpoints may only be written before the last merge pass with the final output.

Sort runs

During a sort run, checkpoints are only written at the end of a cycle. A cycle is terminated after SORT has output a sorted subset to an auxiliary file. In the final cycle, checkpoints can be written immediately before the merge pass with the final output.

A checkpoint can be written irrespective of the execution status by the user issuing a SEND-MESSAGE call and, after entering CKPT as the desired action, requesting termination of the SORT run with TERMINATE. This does not apply to multi-task sorts.

Merge runs

During a merge run, there are no limits on the number and frequency of checkpoints. By specifying the CHECKPOINT and RECORDS-PER-CYCLE operands in the MERGE-RECORDS statement, the user can define the number of merge input records in a cycle, i.e. after how many records a checkpoint is to be taken.

In interactive mode, a SEND-MESSAGE command and a CKPT action can be entered at any time. A checkpoint will then be written at the earliest possible point. This requires the checkpoint file to be of adequate size.

PAM key elimination

The value BLOCK-CONTROL-INFO=*WITHIN-DATA-BLOCK must not be specified for checkpoint files.

RESTART-PROGRAM

The RESTART-PROGRAM command enables an interrupted sort/merge run to be resumed from the last checkpoint written. The name of the checkpoint file and the corresponding PAM page logged at each checkpoint must be specified in the command.

8 Optimization of sort runs

Sort runs can be optimized by

– suitable CORE allocation
– virtual merging
– choice of sorting method (code conversion, cycle sorting, multi-task sorting)
– suitable choice of file characteristics
– summation of records
– load SORT as subsystem
– inclusion of the OPTIMIZATION operand in the SET-SORT-OPTIONS statement
– modification of the preset default values of SORT.

8.1 Suitable CORE allocation

The CORE value defines the size of virtual memory for intensive use.

In normal system utilization, SORT should be left to determine the CORE allocation. SORT calculates the CORE value on the basis of the number of records in a sort cycle. The requisite information for this must be made available to SORT (RECORDS-PER-CYCLE or ESTIMATED-RECORDS value, together with the definition of the auxiliary files via ADD-FILE-LINK assignment and/or TAPE-UNITS specification).

Calculation of the CORE value by SORT:

$$\text{CORE} = \frac{\text{cycle data set in bytes}}{2 ** 20} + 16$$

In interactive and batch modes SORT outputs the value calculated for CORE in SORT message SRT1033 if PLANNING=*DIALOG was specified in the ASSIGN-EXITS statement.

In a system with low utilization or a very large memory capacity, sorting can be accelerated by using the ASSIGN-RESOURCES statement to allocate a CORE value greater than the value calculated by SORT.

In a heavily utilized system it may be possible to improve performance by using the ASSIGN-RESOURCES statement to allocate a CORE value that is less than the value calculated by SORT.

You can modify the CORE value calculated by SORT by specifying the MEMORY-SIZE operand with the ASSIGN-RESOURCES statement. The following priority classes are possible:

MEMORY-SIZE =	**Explanation**
MIN	Minimum value. Corresponds to the CORE value, which can be preset ID- or system-specifically with the MODIFY-SORT-DEFAULTS statement and the CORE-MINIMUM operand (factory presetting: 24).
SMALL	Lower value. Equivalent to the CORE value calculated by SORT -33%.
STD	Value calculated by SORT on the basis of the sort/merge run specifications.
LARGE	Upper value. Equivalent to the CORE value calculated by SORT +33%.
MAX	Maximum value. Corresponds to the CORE limit value, which can be preset ID- or system-specifically with the MODIFY-SORT-DEFAULTS statement and the CORE-LIMIT operand (factory presetting: 96).

The specification of MIN-MSG-WEIGHT=*ALL in the statement SET-SORT-OPTIONS gives the user information about the way the sort run is set up:

– With small input files (e.g. 1000 records, each 50 bytes in length) the system attempts to sort all the records immediately in main memory without buffering to a SORTWK file. The message SRT1013 informs the user of this.

– With larger input files, the system attempts to merge all the sequences created during the pre-sort using a merge run at the end of the process. The message SRT1012 informs the user of this. Only when handling large sort runs is it necessary to use intermediate merge runs as well.

– When using the user exit “PLANNING” with DIALOG, the message SRT1061 specifies the maximum number of merge runs that can be combined in the final end merge.

– When the DOMINO phase is completed, the message SRT1028 specifies the number of sequences that were actually created. By modifying MEMORY-SIZE you may be able to create a more suitable constellation. However, the number of sequences created will actually change from sort run to sort run as the input data changes.

8.2 Virtual merging

The records to be sorted must be placed in an area of virtual memory so that comparisons can be performed. The size of the area can be specified by the user in the MEMORY-SIZE operand of the ASSIGN-RESOURCES statement or is calculated by SORT from the information on the quantity of data. Access to this area is characterized by a very high degree of dispersion, which means that paging may intensify in proportion to the size of the area. This may affect concurrently executing tasks, and this, in turn, may have an impact on SORT runtime.

In a presort with virtual merging, the memory area is divided up in the ratio 1 to 15 into a presorting area and a merging area. This limits the dispersed type of access to 1/16; the remainder (merging area) is accessed sequentially.

A prerequisite for virtual merging is a MEMORY-SIZE value ≥ 400 (defined in the ASSIGN-RESOURCES statement) and a minimum availability of 400 pages of class 6 memory. In addition, the intensively used memory area (1/16 of the specified value) must not exceed the maximum value which can be set using the CORE-MAXIMUM operand of the MODIFY-SORT-DEFAULTS statement. If it does, its size is reduced to this maximum value and the total memory request is recalculated (through multiplication by 15). Only if this newly computed value is ≥ 400 can a virtual merge be performed.

Example

CORE value = 1600, CORE-MAXIMUM = 50

The memory area for intensive use is determined by 1600/16 = 100. As this is greater than the preset limit (50), the latter value is used instead. The definitive CORE value, and thus the amount of virtual memory requested, is reduced by back-calculation (i.e. by multiplying the intensive memory value by 16) to 800 pages (= 50 * 16).

Information concerning the use of virtual merging is provided by the following SORT messages:

The message SRT1033 indicates the size of the requested memory area.

This message SRT1062 indicates that conditions are suitable for virtual merging.

These two messages are output only if DIALOG was specified as the action for the PLANNING user exit in the ASSIGN-EXITS statement and the appropriate message priority was set in the SET-SORT-OPTIONS statement.

8.3 Choice of sort method

Runtime and system throughput in sort operations can be improved by the use of

– code conversion

– cycle sorting

– multi-task sorting.

8.3.1 Code conversion

Code conversions increase the amount of CPU time required. The following is intended to help in the choice of conversion format (assuming such a choice exists).

– Small requirement for conversion and reconversion (MODIFY-CODE, PHYSICAL-TRANSLATE, EBCDIC-ISO-EBCDIC formats)

– Large requirement for conversion per record comparison on auxiliary storage (VIRTUAL-TRANSLATE, EBCDIC-INTERNATIONAL formats)

– Very large requirement for DIN-oriented conversion with special treatment of *umlauts* (EBCDIC-DIN formats).

8.3.2 Cycle sorting

In cycle sorting, the data set to be sorted is divided up into subsets (cycles). SORT sorts each subset separately in a work file (SORTWKx) and then places the result in an auxiliary file (SORTWKxx). Auxiliary files can be set up on tape or disk.

Essentially, cycle sorting is used for the controlled generation of checkpoints (RESTART capability). Some savings in runtime may be possible by setting up the auxiliary files on separate (independent) volumes, thus reducing positioning times. The CPU times will, however, always be greater than for a simple sort run with one cycle.

A cycle sort is performed only if the RECORDS-PER-CYCLE operand is specified, and then only if its value is less than the number of input records.

SORT requires the following resources for cycle sorting:

– 1 disk work file, whose size is given by 1.1 * cycle data set

– n auxiliary files (disk and/or tape), where n=number of cycles-1; the size of each auxiliary file is given by 1.1 * cycle data set.

  The number of auxiliary files required is the maximum value produced by

  the number of files assigned via ADD-FILE-LINK commands with LINK-NAME=SORTWKxx, plus the TAPE-UNITS operand value (in the ASSIGN-RESOURCES statement)

  and

$$\frac{\text{ESTIMATED-RECORDS value}}{\text{RECORDS-PER-CYCLE value}} - 1$$

  For this, the FILE command specifications take precedence.

Notes

– If the number of auxiliary files defined by ADD-FILE-LINK commands with LINK-NAME=SORTWKxx and the TAPE-UNITS operand does not match the cycle number resulting from the ESTIMATED-RECORDS value/RECORDS-PER-CYCLE - 1 then SORT sets up the missing files on disk.

– If there are still not enough auxiliary files, SORT automatically sets up an additional auxiliary disk file. If this also proves insufficient, SORT tries to sort the remainder of the input using an extension of the work file (secondary allocation).

The number of records per cycle is calculated as follows:

$$\text{Cycle record number} = \text{RECORDS-PER-CYCLE value}$$

or alternatively:

$$\text{Cycle record number} = \frac{\text{INPUT-RANGE, NUMBER-OF-RECORDS value}}{\text{Number of auxiliary files} + 1}$$

Example

```
/add-file-link file-name=input.1,link-name=sortin
/add-file-link file-name=output.1,link-name=sortout
/start-sort
%  SRT1001  2014-10-12/15:41:58/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
% SRT1130  PLEASE ENTER SORT STATEMENTS
//sort-records fields=*field-explicit(position=5,length=8, -
//                                    sorting-order=*descending), -
//             records-per-cycle=90
//set-sort-options min-msg-weight=*all
//end
%  SRT1046  2014-10-12/15:42:27/000000.25 END OF PREPARATORY PHASE
%  SRT1010  2014-10-12/15:42:27/000000.27 END OF PRESORT PHASE
%  SRT1027  STRINGS AFTER PRESORTING:.............................1
%  SRT1028  STRINGS AFTER DOMINO:.................................1
%  SRT1012  NO INTERNAL MERGE NECESSARY
%  SRT1015  2014-10-12/15:42:28/000000.30 END OF CYCLE ...........1
%  SRT1010  2014-10-12/15:42:28/000000.30 END OF PRESORT PHASE
%  SRT1027  STRINGS AFTER PRESORTING:.............................1
%  SRT1028  STRINGS AFTER DOMINO:.................................1
%  SRT1012  NO INTERNAL MERGE NECESSARY
%  SRT1015  2014-10-12/15:42:28/000000.33 END OF CYCLE ...........2
%  SRT1010  2014-10-12/15:42:28/000000.33 END OF PRESORT PHASE
%  SRT1027  STRINGS AFTER PRESORTING:.............................1
%  SRT1028  STRINGS AFTER DOMINO:.................................1
%  SRT1012  NO INTERNAL MERGE NECESSARY
%  SRT1015  2014-10-12/15:42:28/000000.36 END OF CYCLE ...........3
%  SRT1010  2014-10-12/15:42:28/000000.36 END OF PRESORT PHASE
%  SRT1027  STRINGS AFTER PRESORTING:.............................1
%  SRT1028  STRINGS AFTER DOMINO:.................................1
%  SRT1012  NO INTERNAL MERGE NECESSARY
%  SRT1016  SORT/MERGE INPUT RECORDS:............................300 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:...........................300
%  SRT1025  PAM INPUTS:...........................................8
%  SRT1026  PAM OUTPUTS:..........................................8
%  SRT1002  2014-10-12/15:47:26/000000.48 SORT/MERGE COMPLETED
```

The auxiliary files are set up by SORT and deleted again at the end of the run.

8.3.3 Multi-task sorting

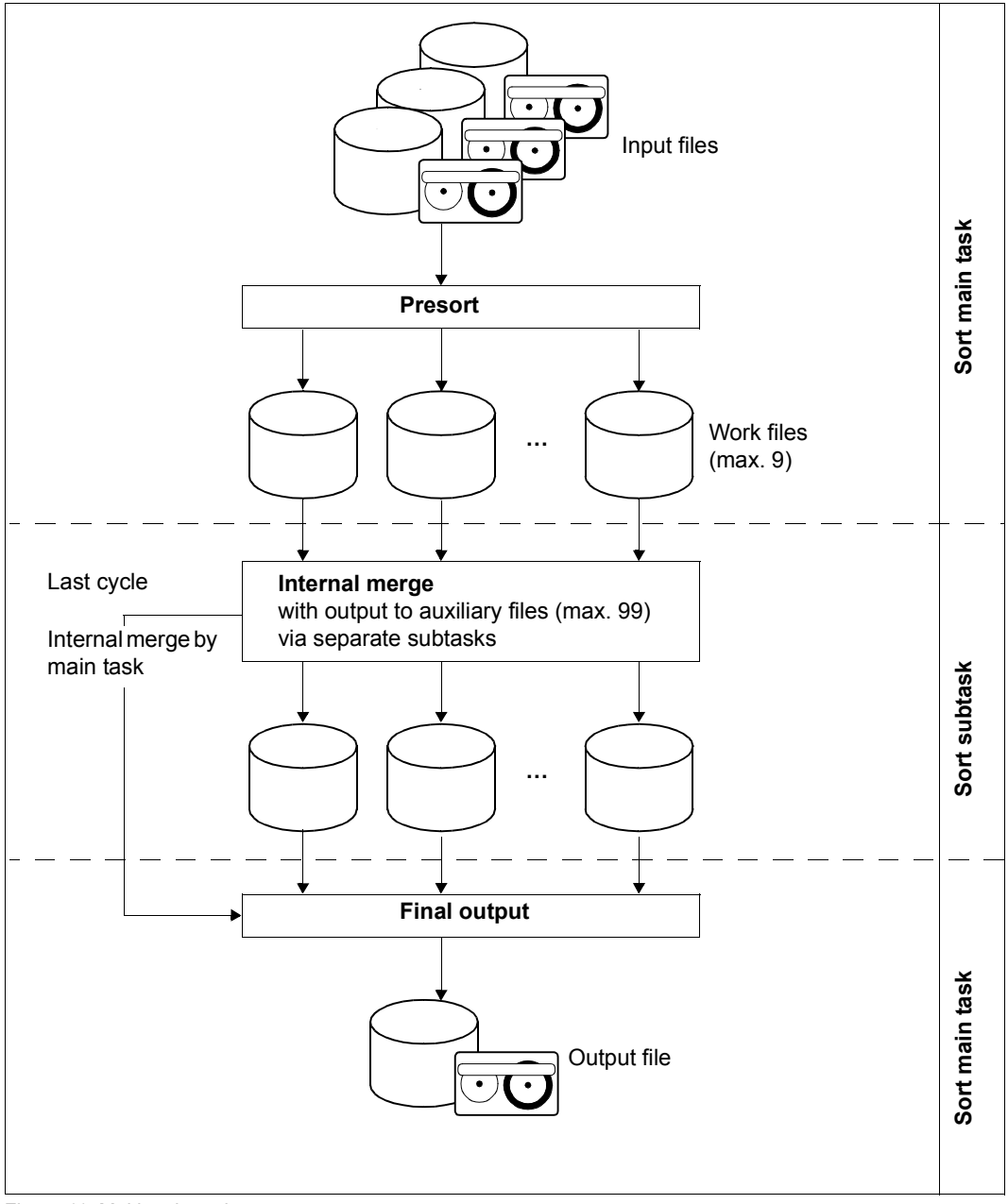


Figure 11: Multi-task sorting

Multi-task sorting involves the use of a number of work files (between 2 and a maximum of 9). This sorting technique is a form of cycle sorting with multiple task runs, where there is one main task and up to 8 subtasks. The number of tasks (one main task and several subtasks) that can execute concurrently is equal to the number of work files. This determines the degree of parallel working by SORT. The total number of subtasks activated is primarily determined by the result of (ESTIMATED-RECORDS/RECORDS-PER-CYCLE). If the corresponding values are not available, SORT uses the number of auxiliary files.

Properly applied, multi-task sorting produces savings in runtime that can be considerable. However, since the CPU time required is always greater than for simple sorting in one cycle or for cycle sorting, increased throughput can only be achieved in a lightly loaded system with adequate reserves of CPU time.

Sort main task

The sort main task

– presorts the first input

– outputs subsets to work files

– coordinates the subtask merge runs

– uses the final sort cycle to merge the auxiliary files written by the subtasks

The first input is presorted in the sort main task. Each of the work files is then assigned to a separate sort subtask for internal merging followed by output to an auxiliary file. In the final cycle the main task merges the generated auxiliary files and performs internal merging in order to produce the final output.

The sort main task and the subtasks communicate with one another in order to synchronize their execution. Thus, for example, the main task may have to wait until a work file becomes free or all subtask runs are completed.

Presorting in the main task and internal merging in the subtasks are performed in parallel.

Sort subtasks

In the sort subtasks, dummy records are used to optimize the sorted sequences (usually accomplished in the main task) and the individual subsets (sorted sequences) are merged internally and output to auxiliary files.

Resources required for multi-task sorting include:

– input file(s) (max. 99)

– work files on disk (2 to max. 9)

– auxiliary files on disk (max. 99)

– output file.

Note

The MIN-MSG-WEIGHT operand of the SET-SORT-OPTIONS statement controls the SYSLST output of the sort subtasks. When MIN-MSG-WEIGHT=*ALL is specified, the SYSLST log is always output; it is otherwise only output when there is an error in the sort subtask.

Requirements for multi-task sorting

The conditions outlined below should apply for multi-task sorting. Otherwise a cycle sort should be performed using a single work file. Where there is uncertainty about certain values (e.g. record length, ESTIMATED-RECORDS), it is better to make no specifications since incorrect values may lead to an abort.

– SORT has to be assigned at least 2 and at most 9 work files. This is how SORT recognizes that a multi-task sort is required.

– The user ID under which the sorting is performed must possess an “express tag” (SHOW-USER-ATTRIBUTES command, START-IMMED=YES operand).

– If ESTIMATED-RECORDS and RECORDS-PER-CYCLE are specified, SORT calculates the number of auxiliary files (max. 99).

– System loading should be light enough to allow the subtasks responsible for internal merging to be started immediately and run without interruption.

– Auxiliary files on work tapes must not be used for multi-task sorting. If the TAPE-UNITS parameter in the ASSIGN-RESOURCES control statement is set to a non-zero value, SORT automatically performs only a cycle sort.

– Devices on which private disks containing work or auxiliary files are used must be shareable.

– The VIRTUAL-TRANSLATE and/or EXTERNAL-COMPARE user exits must not be used.

If no ESTIMATED-RECORDS or RECORDS-PER-CYCLE values have been specified, SORT calculates a RECORDS-PER-CYCLE value from the input data set (size of the input file in PAM pages), the average record length and the number of auxiliary files. With tape files, SORT cannot determine the amount of input data and so cannot calculate a RECORDS-PER-CYCLE value. It follows that in such cases no multi-task sorting is performed.

Special error situations

SORT issues an error message when the following errors have occurred. If necessary the sort run is aborted.

– The “Express” entry is missing in the user ID:
Message SRT1065 is issued; the sort run continues normally with cycle sorting in the main task.

– The “Express” entry is removed during the sort run:
Warning SRT1066 is issued; the sort run continues normally with cycle sorting in the main task.

– Sort subtasks are aborted with “CANCEL-JOB”:
Messages SRT1051, SRT1052 and SRT1053 are issued; the sort run is resumed. The auxiliary files of the aborted auxiliary tasks are ignored in the last sort cycle (main task). The sort/merge run ends normally with message SRT1059.

– Sort subtasks are not initiated:
After a waiting time of 10 minutes, message SRT1068 is issued; the sort run is terminated.

– Other error conditions of sort subtasks:
In this case, messages SRT1067, SRT1052 and SRT1053 are issued during the end merge; the sort run is resumed. The auxiliary files of the affected subtasks are ignored during the final sort cycle (main task). The sort/merge run ends normally with message SRT1059.

Examples:

Variant 1

```
/add-file-link link-name=sortin,file-name=input.1
/add-file-link link-name=sortout,file-name=output.1
/add-file-link link-name=sortwk1,file-name=work.1
/add-file-link link-name=sortwk2,file-name=work.2
/start-sort
%  SRT1001  2014-10-12/13:50:09/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
% SRT1130  PLEASE ENTER SORT STATEMENTS
//sort-records fields=*field-explicit(position=5,length=10), -
//             estimated-records=1000,records-per-cycle=300
//set-sort-options min-msg-weight=*all
//end
% SRT1046  2014-10-12/13:50:10/000000.24 END OF PREPARATORY PHASE
% SRT1010  2014-10-12/13:50:11/000000.30 END OF PRESORT PHASE
% JMS0066 JOB ‚EXAMPLE‘ ACCEPTED ON 96-02-05 AT 13:50, TSN = 8D2R
% SRT1010  2014-10-12/13:50:13/000000.41 END OF PRESORT PHASE
% JMS0066 JOB ‚EXAMPLE‘ ACCEPTED ON 96-02-05 AT 13:50, TSN = 8D2U
% SRT1010  2014-10-12/13:51:00/000000.51 END OF PRESORT PHASE
% JMS0066 JOB ‚EXAMPLE‘ ACCEPTED ON 96-02-05 AT 13:51, TSN = 8D3B
% SRT1010  2014-10-12/13:51:01/000000.59 END OF PRESORT PHASE
% SRT1027  STRINGS AFTER PRESORTING:.............................1
% SRT1028  STRINGS AFTER DOMINO:.................................1
% SRT1012  NO INTERNAL MERGE NECESSARY
% SRT1016  SORT/MERGE INPUT RECORDS:.........................2.000 (FROM 01)
% SRT1030  SORT/MERGE OUTPUT RECORDS:........................2.000
% SRT1025  PAM INPUTS:...........................................2
% SRT1026  PAM OUTPUTS:.........................................11
% SRT1002  2014-10-12/13:51:39/000000.72 SORT/MERGE COMPLETED
```

Altogether, 3 subtasks are started, each sorting 600 records into an auxiliary file. The number of auxiliary files is calculated by SORT (using ESTIMATED-RECORDS/ RECORDS-PER-CYCLE). The final 200 records are sorted by the main task and merged together with the three auxiliary files into the output file.

Variant 2

```
/add-file-link link-name=sortin,file-name=input.1
/add-file-link link-name=sortout,file-name=output.1
/add-file-link link-name=sortwk1,file-name=work.1
/add-file-link link-name=sortwk2,file-name=work.2
/add-file-link link-name=sortwk01,file-name=auxiliary.1
/add-file-link link-name=sortwk02,file-name=auxiliary.2
/add-file-link link-name=sortwk03,file-name=auxiliary.3
/start-sort
%  SRT1001  2014-10-12/13:46:36/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//sort-records fields=*field-explicit(position=5,length=10), -
//             estimated-records=2000
//set-sort-options min-msg-weight=*all
//end
%  SRT1046  2014-10-12/13:46:37/000000.25 END OF PREPARATORY PHASE
%  SRT1010  2014-10-12/13:46:37/000000.31 END OF PRESORT PHASE
%  JMS0066 JOB ‚EXAMPLE‘ ACCEPTED ON 96-02-05 AT 13:46, TSN = 8DZB
%  SRT1010  2014-10-12/13:46:38/000000.42 END OF PRESORT PHASE
%  JMS0066 JOB ‚EXAMPLE‘ ACCEPTED ON 96-02-05 AT 13:46, TSN = 8DZC
%  SRT1010  2014-10-12/13:47:35/000000.51 END OF PRESORT PHASE
%  JMS0066 JOB ‚EXAMPLE‘ ACCEPTED ON 96-02-05 AT 13:47, TSN = 8DZI
%  SRT1010  2014-10-12/13:47:36/000000.60 END OF PRESORT PHASE
%  SRT1027  STRINGS AFTER PRESORTING:.............................1
%  SRT1028  STRINGS AFTER DOMINO:.................................1
%  SRT1012  NO INTERNAL MERGE NECESSARY
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................2.000 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................2.000
%  SRT1025  PAM INPUTS:.......................................3
%  SRT1026  PAM OUTPUTS:.....................................12
%  SRT1002  2014-10-12/13:48:27/000000.70 SORT/MERGE COMPLETED
```

SORT calculates a RECORDS-PER-CYCLE value from the value specified for ESTIMATED-RECORDS and the number of assigned auxiliary files. The number of subtasks is equal to the number of assigned auxiliary files.

Variant 3

```
/add-file-link file-name=input.1,link-name=sortin
/add-file-link file-name=output.1,link-name=sortout
/add-file-link file-name=work.1,link-name=sortwk1
/add-file-link file-name=work.2,link-name=sortwk2
/start-sort
% BLS0517 MODULE 'SRT80' LOADED
% SRT1001  2014-10-12/12:10:46/000000.00 SORT/MERGE STARTED, VERSION
08.0A00/BS2000V18.0
% SRT1130  PLEASE ENTER SORT STATEMENTS
//sort-records fields=*field-explicit(position=5,length=10)
//set-sort-options min-msg-weight=*all
//end
% SRT1046  2014-10-12/12:10:47/000000.24 END OF PREPARATORY PHASE
% SRT1010  2014-10-12/12:10:49/000000.41 END OF PRESORT PHASE
% SRT1027  STRINGS AFTER PRESORTING:.............................2
% SRT1028  STRINGS AFTER DOMINO:.................................2
% SRT1012  NO INTERNAL MERGE NECESSARY
% SRT1016  SORT/MERGE INPUT RECORDS:.........................2.000 (FROM 01)
% SRT1030  SORT/MERGE OUTPUT RECORDS:........................2.000
% SRT1025  PAM INPUTS:.........................................11
% SRT1026  PAM OUTPUTS:........................................11
% SRT1002  2014-10-12/12:10:51/000000.51 SORT/MERGE COMPLETED
```

As SORT cannot calculate a RECORDS-PER-CYCLE value and no auxiliary files are assigned either, no multi-task sorting is performed.

Note

Performance can be improved in multi-task sorting, as in cycle sorting, by storing work files and auxiliary files on separate data volumes with independent access facilities. The same applies to the input files and output files, though these do not require separate access facilities.

8.4 Suitable choice of file characteristics

Input files (SORTINxx)
Wherever possible, input files should be set up with a block size of at least 4096 bytes (even chaining just 2 PAM pages can produce significant savings in CPU time and runtime). Setting up input files on private volumes usually results in faster processing (runtime saving).

Output file (SORTOUT)
Output files should be set up with a block size of at least 4096 bytes. Even chaining just 2 PAM pages can produce significant savings in CPU time and runtime).
It is also important to ensure that adequate primary and secondary space allocations are defined. This can reduce administrative overhead in respect of file extensions. Other file attributes should, as far as possible, be taken over by SORT from the input file so as to avoid errors. Runtime improvements can usually be achieved by setting up output files on private volumes.

Work files (SORTWK or SORTWKx)
Significant savings in runtime are possible by setting up work files on a private disk and making sure that the disk will not be accessed concurrently by other executing tasks. In multi-task sorting this applies to each individual work file.
The file attributes of a work file are fully defined by SORT and if the SPACE specification is too low, it will be corrected.

Auxiliary files (SORTWKxx)
The same applies to auxiliary files as to work files, this time including files on tape. Here, the restrictions on multi-task sorting have to be taken into account. Tapes should be processed at high recording density.

8.5 Record summation

The SUM-RECORDS statement should be used for record summation. This causes summation to be initiated in the presort phase, thereby helping to avoid high-volume inputs and outputs. Record summation means that each time that two records with identical sort keys are encountered, the contents of the sum fields specified in the SUM-RECORDS statement are added together in the first record and the second record is deleted. In test measurements, savings in CPU time and runtime of up to 60% (!) have been recorded (dependent on the degree of compression).

8.6 SORT as subsystem

Parts of SORT can be loaded by systems support in the form of a subsystem. This saves part of the load process and thus makes SORT faster.

This is particularly valid for multi-task sort operations which would otherwise reload some of the modules for each subtask.

8.7 Use of the OPTIMIZATION operand in the SET-SORT-OPTIONS statement

The OPTIMIZATION operand of the SET-SORT-OPTIONS statement can be used to optimize the sort run in terms of runtime, CPU time or memory requirements. Thus, OPTIMIZATION = VIRTUAL-MEMORY causes load modules (which account for about 85% of static program code) to be released when they are no longer required. Runtimes and CPU times increase when load modules are released and when they are reloaded for a succeeding sort run. Against this, in a system short of address space OPTIMIZATION = VIRTUAL-MEMORY can be used so that memory freed by releasing the load modules can be made available for sort work areas. As a result, SORT makes more efficient use of available memory.

8.8 Modifying the preset default values for SORT

SORT parameters can be modified and displayed by means of the statements MODIFY-SORT-DEFAULTS (see page 152) and SHOW-SORT-DEFAULTS (see page 174).

9 Installation

SORT V8.0 runs under the operating system BS2000/OSD as of V8.0.

For compatibility reasons with older applications, a library SORTLIB is still required. SORTLIB contains the starter module SRT80 (object module). The object module SRTXKERN is also included. This is used to satisfy calls which are permanently programmed in old applications and refer to this module in the library SORTLIB.
If IMON is not in use, the library SORTLIB is still accessed during subprogram calls. This is also valid for old starter modules from SORT V7.4 and lower as they do not yet work with IMON.

The installation is not tied to fixed file names and IDs. This means that the products themselves have to dynamically establish the location of all components of a version. The tool provided for this is the IMON subsystem (see the “IMON” [11]).

Product files

SYSPRG	Phase (compatibility)	SORT
SYSOML	Module library	SORTLIB
SYSLNK	Module library	SYSLNK.SORT.080
SYSLNK.TAB	Table library	SYSLNK.SORT.080.TAB
SYSPAR	Parameter file	SYSPAR.SORT.080
SYSLIB	Macro library	SYSLIB.SORT.080
SYSSDF	SDF syntax file	SYSSDF.SORT.080
SYSMES	Message file (MSGMAKER format)	SYSMES.SORT.080
SYSSPR	Procedure library	SYSSPR.SORT.080
SYSREP	REP file	SYSREP.SORT.080
SYSSSC	Subsystem declarations upper address space	SYSSSC.SORT.080
SYSSSC.LOW	Subsystem declarations lower address space	SYSSSC.SORT.080.LOW
SYSSII	Installation information for IMON	SYSSII.SORT.080
SYSFGM.D	Release Notice - German	SYSFGM.SORT.080.D
SYSFGM.E	Release Notice - English	SYSFGM.SORT.080.E
SYSACF	ACS model file	SYSACF.SORT.080
SYSRME.D	Readme File - German	SYSRME.SORT.080.D
SYSRME.E	Readme File - Englisch	SYSRME.SORT.080.E
SYSNRF		SYSNRF.SORT.080

Comments

1. The SYSSPR.SORT.080 and SYSACF.SORT.080 files have fixed names and must be provided under these names in the default user ID (DEFLUID).
2. For the purposes of compatibility the macro library SYSLIB.SORT.080 is still to be saved as $.SORTMACLIB, or is to be assigned this name using ACS resources.
3. The library $.SORTLIB is always supplied. It contains the modules SRT80 and SRTXKERN. The correct library is always entered from SRTXKERN (SYSLNK.SORT.080).

Freely selectable file names

The installation is not tied to fixed file names and IDs.

The names of the individual SORT files and the IDs under which they are stored can be defined during installation with IMON-GPN. This allows you, for example, to store the sort library under an ID called SORT.

The prerequisite for this is that the subsystems DSSM and IMON-GPN are installed.

The associated file names of all product files (the complete path names) are stored in the so-called IMON-SCI (Software-Configuration-Inventory), the central IMON data basis and can be retrieved from there in later SORT applications. So the user does not need to know these names - they are ascertained for him or her by the operating system.

The IMON-SCI contains the following three pieces of information for each product file:

1. Logical name of the file (for the sort library this is, for example, SYSLNK)
2. Name of the product (SORT)
3. Product version (e.g. 08.0A00)

Coexistence

Because the names and storage IDs of the product files are not fixed, it is possible to provide several versions of a product on one system. The various versions of the product must only be installed under different IDs or with different file names.

This coexistence of various product versions on a single system is supported by SORT as of V7.5A, this basically means that SORT V8.0A is able to coexist with SORT versions as of V7.5A. Neither version V7.4A nor any of the lower versions are capable of coexistence, since they still contain hard-wired file names and do not contain organisatory utilities.

Coexistence is only permitted for main versions, the various release or correction statuses of the same main version are not capable of coexisting with each other.

Example

Versions V7.9A and V8.0A are capable of coexisting with each other. But version V8.0B or V8.0A10 would not be capable of coexisting with V7.9A00.

In the START-SORT and SORT-FILE commands as well as in the macros for calling subroutines, the user can select the desired SORT version using the VERSION operand.

A further possibility of requesting the desired SORT version is provided by the system command SELECT-PRODUCT-VERSION. In this command the user can define the desired SORT version before the SORT call (in the case of a SORT subroutine variant, before the associated main program is called). This SORT version is then selected if no explicit version is specified when SORT is started. The SCOPE operand of the SELECT-

PRODUCT-VERSION command controls the period of validity of this specification. SCOPE=*PROGRAM defines the SORT version for the subsequent program call, i.e. the SELECT-PRODUCT-VERSION command must be repeated before each main-program call. SCOPE=*TASK defines the SORT version until the end of the task.

If the version is specified in full in the start command or during the subroutine call, this is the version used (provided this version is correctly installed). If the version specification is missing (i.e. *STD) or is not precise enough, more than one version may be possible. In this case, the command server, or SORT in the subroutine call, ascertains all the versions that could be meant. The version is then determined according to the following priorities:

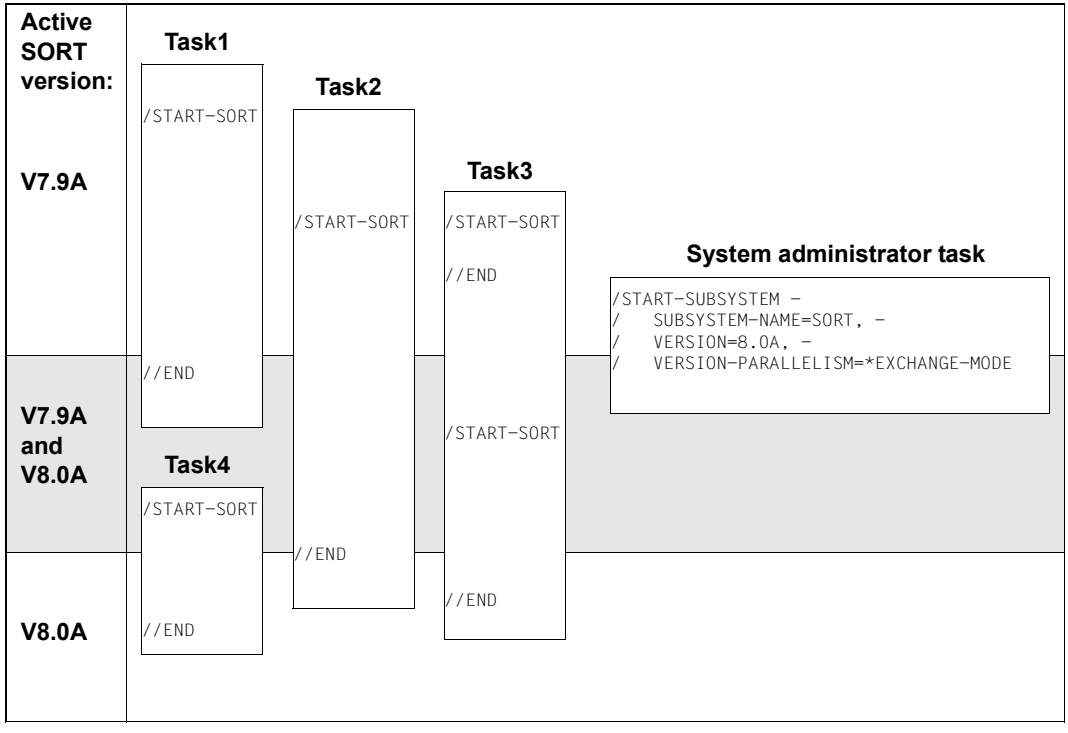
1. The version preset with the /SELECT-PRODUCT-VERSION command.
2. The highest SORT version installed with IMON.

If in an operating system the IMON-GPN subsystem is not installed, the pre-linked SORT module is dynamically loaded from the $.SYSLNK.SORT.080 or $.SORTLIB library. This also applies if the IMON-GPN subsystem is active but its IMON-SCI does not contain the product SORT.

If an explicitly requested SORT version is not found, the process is aborted.

Interchangeability

If several versions of SORT are installed at the same time, it is possible to swap between them during operation. The following figure shows an example of the time sequence involved in exchanging one version (V7.9A) of the SORT subsystem for another (V8.0A):



At the beginning of the example, SORT V7.9A is active. Three tasks (task1, task2 and task3) start one SORT run each.
With the START-SUBSYSTEM command, the system administrator activates version V8.0A of SORT, allowing the temporary coexistence of several versions by specifying VERSION-PARALLELISM=*EXCHANGE-MODE. By this time the SORT run in task 3 has already terminated, but the two tasks task1 and task2 are still working with SORT. Version V7.9A of SORT therefore remains active, but all tasks which start a SORT run from now on (task3 and task4) work with the new version, V8.0A.
The termination of the SORT run in task1 does not affect the termination of V7.9A of SORT. This version of the SORT subsystem is not deactivated until the last task (task2), which is still using it, terminates the SORT run.

10 Examples

This chapter contains application examples for most of the SORT functions. The examples have been tried out in practice and the resulting trace listings reprinted here, except in the case of assembler programs, where for reasons of space the source code is given instead of the assembly listing.

10.1 Introduction

This introduction has the following structure:

– brief recapitulation of the syntax of the SORT statements

– examples illustrating the syntax

Brief recapitulation of the syntax of the SORT statements

Executing a sort run generally requires three steps:

– calling SORT

– assignment of the input and output files required by SORT

– definition of the sort criteria

10.1.1 Calling SORT

SORT can be called with one of the following commands (see chapter “Calling SORT” on page 191):

START-SORT	SORT must be called with this command if the entire SORT functionality is to be available. The examples that follow will therefore use only this type of call.
SORT-FILE	This command can be used to perform simple sorting. No further input is necessary because with this command the file assignments and the definition of sort criteria are made directly.

Calling SORT with START-SORT

```
/start-sort
%  SRT1001  2014-10-12/12:19:38/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V14.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
%//
```

By means of the prompt ’//’ (double slash), SORT indicates that it is ready to receive SORT control statements.

10.1.2 Assigning the files

Before any SORT run can take place, SORT requires (at least) one input file and an output file.
SORT requires the input and output files to be specified explicitly for each SORT run.

Input and output files can be assigned in various ways:

– In a SORT statement:

```
//ASSIGN-FILES INPUT-FILES=inputfile, OUTPUT-FILE=outputfile
```

– Before calling SORT:

```
/ADD-FILE-LINK LINK-NAME=SORTIN,  FILE-NAME=inputfile....
/CREATE-FILE FILE-NAME=outputfile
/ADD-FILE-LINK LINK-NAME=SORTOUT, FILE-NAME=outputfile....
```

– By direct specification of the files in the SORT-FILE command:

```
/SORT-FILE ... INPUT-FILES=inputfile,OUTPUT-FILE=outputfile...
```

Note

In general SORT generates output files with the same file attributes as the input files. Exceptions to this are

– sorts in which SORT has other default specifications (cf. selection sorting)

– sorts in which the output file format is intentionally different from that of the input file

SORT accesses or creates the following files in a sort/merge run:

– input files
– output file
} must be assigned by the user for every sort process

– work files

– auxiliary files
– checkpoint file
} only significant with large input files

Small files can be sorted directly in the CORE space.

10.1.3 Defining the sort criteria

Sort criteria in the wider sense comprise all instructions (statements) issued to SORT. Sort criteria in the narrower sense are specifications which refer to the sort key and the sorting method. *Sort criteria* is always used below in its narrower sense.

Syntax example for a simple sort run

The simplest type of sort can be initiated by

```
//sort-records
```

Here SORT uses the default settings predefined by SDF. These defaults are:

FIELDS = COMPLETE-RECORD	The entire record is evaluated
FORMAT = CHARACTER	The data is interpreted as alphanumeric characters
SORTING-ORDER = ASCENDING	The data is sorted in ascending order. Following the sort, the least significant data comes first and the most significant is located at the end of the file.

Example

Using the command as given in the example to sort a file consisting only of uppercase letters would result in a sorted file with 'A' first and 'Z' last.

SORT does not restrict sorting to whole data records only. The SORT-RECORDS statement offers the following additional possibilities:

– Sections of a record can be specified which are to be taken into account in the sort process. Other parts of the record do not then affect the sorting. For example, a list of names is to be sorted according to last names and within the same last name according to first names, whereby these fields need not necessarily be at the beginning of the record.
– The fields of an input record can be combined to form an output record in a new order defined by the user.

The FIELDS operand in the SORT-RECORDS statement is available for defining fields which are to be taken into account for the sorting or incorporated into the output record.

Field definitions

The sort key consists of one or more sort fields. Sort fields describe all areas of the input record that are evaluated by SORT in order to determine the order of the records in the output file. The fields are defined via the FIELDS operand.
By default SORT evaluates the sort fields in the order given in the command. The most significant (or primary) sort argument should be specified first. (For departures from this rule, cf. PRIORITY in SORT-RECORDS.)

```
//sort-records fields=( -
//     *field-explicit(position=..,length=..,format=..,sorting-order=..), -
//     *field-explicit(...),..)
```

Here, the specifications following FIELD-EXPLICIT have the following function:

POSITION	Start of the field, in bytes
LENGTH	Length of the field, in bytes
FORMAT	Data format
SORTING-ORDER	Sorting direction

Descriptions of the maximum sizes and possible entries for the wildcard characters are given in the preceding chapters of the manual.

Syntax example: one sort key - two sort fields

```
//sort-records fields=( -
//     *field-explicit(position=1,length=11), -
//     *field-explicit(position=23,length=10,sorting-order=*descending))
```

The statement defines two sort fields as the sort key.

1st sort field: *FIELD-EXPLICIT(POSITION=1,LENGTH=11)

SORT begins at byte 1 (POSITION) and evaluates the following 11 bytes (LENGTH) of the record. This part of the record is to be used for sorting in ascending order. The data itself is of the type CHARACTER. These last two criteria do not need to be specified explicitly, as they are already preset as the default values.

2nd sort field: *FIELD-EXPLICIT(POSITION=23,LENGTH=10,
SORTING-ORDER=*DESCENDING)

The second sort field begins at position 23 and extends over the next 10 bytes. The difference is that this part of the record is to be used for sorting in descending order (SORTING-ORDER=*DESCENDING).

10.1.4 Terminating statement input and starting the sort run

Every sort run definition has to be concluded by the END statement. This indicates to SORT that it can begin to execute the statements entered.

10.2 Example: Sorting a file with fixed-length records

SORT is to sort a file called ADDRESSES alphabetically by street names. The records in the ADDRESSES file have the following structure and contents:

Name	First name	Street Sorting is to be performed on this field (sort field)	ZIP	Tel.
Miller	Andrew	Poplar Avenue 47	KT25	544507
Allan	Hilary	High Street 101	AY4	345679
Smith	Albert	Gardener Street 14	PX453	047913
Majors	Christine	Railway Cuttings 12	PX23	987650
Smythe	Brenda	Thomas Square 1	BT34	965471
Kennedy	George	Edgeware Road 62	NY211	873250
Stevens	Henry	Market Square 13	NY12	987234
Baker	Fred	Scott Street 34	KT23	765921
Johnston	Annette	Richmond Street 98	BT342	345678
Mellors	Ingrid	Salford Drive 4	TI34	456372
Brown	Tony	Skyview Terrace 9	UB81	786534
Charles	Ernest	Millhouse Street 23	TI32	537892
Walters	Claudia	Millford Crescent 31	ZY21	342108
Richards	Bernard	Illsley Square 3	UB12	518376
Drever	James	Rose Drive 31	PX3	875211

Displaying the attributes of the ADDRESSES file

```
/show-file-attributes file-name=addresses, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.ADDRESSES
%  ------------------------------ ORGANIZATION ------------------------------
%  FILE-STRUC = SAM          BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE   IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (F,N)        REC-SIZE   = 62
%  AVAIL      = *STD
%  WORK-FILE  = *NO          F-PREFORM  = *K           SO-MIGR    = *ALLOWED
```

The attributes of the input file ADDRESSES are requested via the SHOW-FILE-ATTRIBUTES command. The system identifies the following attributes, among others, for the ADDRESSES file:

– SAM file (FILE-STRUC=SAM)

– fixed record length (REC-FORM=(F,N))

– length of record (REC-SIZE=62).

SORT sets up the output file with these same attributes.

Calling SORT

```
/start-sort
%  SRT1001  2014-10-12/16:05:25/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
```

SORT reports that it is ready and expects the input of the SORT-RECORDS statement.

Assigning the input and output files

```
//assign-files input-files=addresses,output-file=addresses.sort
```

The ASSIGN-FILES statement is used to assign the ADDRESSES file as the input file and the ADDRESSES.SORT file as the output file.
ADDRESSES.SORT is set up as the output file by SORT with the same file attributes as the ADDRESSES input file.

Defining the sort field

```
//sort-records fields=(*field-explicit(position=23,length=26))
```

The ADDRESSES file is to be sorted by street names. The “Street” field begins at byte 23 and the following 26 bytes are to be evaluated. Thus, the sort field (POSITION=23,LENGTH=26) is specified as the sort key in the *FIELD-EXPLICIT operand of the SORT-RECORDS statement.

Concluding the statement sequence and starting the sort run

```
//end
```

The input of statements to SORT is concluded by means of the END statement, and the sort run is started.

Messages from SORT

```
% SRT1016 SORT/MERGE INPUT RECORDS:.............................15 (FROM 01)
% SRT1030 SORT/MERGE OUTPUT RECORDS:............................15
% SRT1002 2014-10-12/16:12:59/000000.13 SORT/MERGE COMPLETED
```

SORT reports that 15 records have been read in and 15 records output. The result of the sort can be viewed by displaying the output file ADDRESSES.SORT.

Contents of the ADDRESSES.SORT file (output file)

Name	First name	Street (sort field)	ZIP	Tel.
1	11	23	49	56 62
Kennedy	George	Edgeware Road 62	NY211	873250
Smith	Albert	Gardener Street 14	PX453	047913
Allan	Hilary	High Street 101	AY4	345679
Richards	Bernard	Illsley Square 3	UB12	518376
Stevens	Henry	Market Square 13	NY12	987234
Walters	Claudia	Millford Crescent 31	ZY21	342108
Charles	Ernest	Millhouse Street 23	TI32	537892
Miller	Andrew	Poplar Avenue 47	KT25	544507
Majors	Christine	Railway Cuttings 12	PX23	987650
Johnston	Annette	Richmond Street 98	BT342	345678
Drever	James	Rose Drive 31	PX3	875211
Mellors	Ingrid	Salford Drive 4	TI34	456372
Baker	Fred	Scott Street 34	KT23	765921
Brown	Tony	Skyview Terrace 9	UB81	786534
Smythe	Brenda	Thomas Square 1	BT34	965471

File attributes of the ADDRESSES.SORT output file

```
/show-file-attributes file-name=addresses.sort, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.ADDRESSES.SORT
%  ------------------------------ ORGANIZATION -----------------------------
%  FILE-STRUC = SAM          BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE   IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (F,N)        REC-SIZE   = 62
%  AVAIL      = *STD
%  WORK-FILE  = *NO          F-PREFORM  = *K           SO-MIGR    = *ALLOWED
```

The output file has the same file attributes as the input file.

10.3 Example: Sorting a SAM file with variable record format

The file named LITERATURE is a SAM file containing variable-length records (RECFORM=V).

File attributes of the LITERATURE.SAM file

```
/show-file-attributes file-name=literature, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.LITERATURE
%  ------------------------------ ORGANIZATION -----------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (V,N)       REC-SIZE   = 0
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K           SO-MIGR    = *ALLOWED
```

The file contains records with the following structure and contents and is to be sorted by title.

RL	Name	First name	Title (sort field)	Genre
1	5	19	30	63
	Pasternak	Boris	Doctor Zhivago	Novel
	Capote	Truman	In Cold Blood	Novel
	Boyle	Jimmy	A Sense of Freedom	Autobiography
	Arden	John	Sergeant Musgrave's Dance	Theater
	Milligan	Spike	Puckoon	Novel
	Dahl	Roald	Kiss Kiss	Short Stories
	Shakespeare	William	Romeo and Juliet	Theater
	Fielding	Henry	Tom Jones	Novel
	Jonson	Ben	Volpone	Theater
	Dumas	Alexandre	The Three Musketeers	Novel
	Troyat	Henri	Pushkin	Biography
	Shaw	Bernard	Pygmalion	Theater
	Sharpe	Tom	Riotous Assembly	Novel
	Thomas	Dylan	Fern Hill	Poem
	Gogol	Nikolai	Dead Souls	Novel

Calculating the position of the first data byte in records of variable length

Every variable-length record is prefixed by a 4-byte record length field. In the example this field is identified by RL. Although the user has no control over this field, it still forms part of the data record. Therefore the number of bytes in the record length field must be added to each data byte position.

Trace listing of the sort by title

```
/start-sort
%  SRT1001 2014-10-12/15:56:27/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130 PLEASE ENTER SORT STATEMENTS
//assign-files input-files=literature,output-file=literature.sort
//sort-records fields=(*field-explicit(position=30,length=33))
//end
%  SRT1016 SORT/MERGE INPUT RECORDS:.............................15 (FROM 01)
%  SRT1030 SORT/MERGE OUTPUT RECORDS:............................15
%  SRT1002 2014-10-12/15:56:31/000000.26 SORT/MERGE COMPLETED
```

Result of the sort in the output file LITERATURE.SORT

```
1    5             19          30                              63

     Boyle         Jimmy       A Sense of Freedom              Autobiography
     Gogol         Nikolai     Dead Souls                      Novel
     Pasternak     Boris       Doctor Zhivago                  Novel
     Thomas        Dylan       Fern Hill                       Poem
     Capote        Truman      In Cold Blood                   Novel
     Dahl          Roald       Kiss Kiss                       Short Stories
     Milligan      Spike       Puckoon                         Novel
     Troyat        Henri       Pushkin                         Biography
     Shaw          Bernard     Pygmalion                       Theater
     Sharpe        Tom         Riotous Assembly                Novel
     Shakespeare   William     Romeo and Juliet                Theater
     Arden         John        Sergeant Musgrave‘s Dance       Theater
     Dumas         Alexandre   The Three Musketeers            Novel
     Fielding      Henry       Tom Jones                       Novel
     Jonson        Ben         Volpone                         Theater
```

The LITERATURE.SORT file is now arranged in ascending alphabetical order by title.

File attributes of the output file LITERATURE.SORT

```
/show-file-attributes file-name=literature.sort, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.LITERATURE.SORT
%  ------------------------------ ORGANIZATION -----------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (V,N)       REC-SIZE   = 2044
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K          SO-MIGR    = *ALLOWED
```

10.4 Overview of the application examples

10.4.1 SORT as main program

No.	Sort method	File type and record format of input file	File type and record format of output file	Supplementary criteria
01	Full sort	SAM, fixed	SAM, fixed	SORT-RECORDS
02	Full sort	SAM, variable	SAM, variable	Position of sort field with RLF
03	Full sort	ISAM, variable	SAM, variable	Different file attributes for output file
04	Full sort	ISAM, variable	ISAM, variable	Output file with different key position
05	Full sort	ISAM, variable	SAM, variable	More than one unsorted input files
06	Full sort	SAM, variable	SAM, variable	Input file identical to output file; use of symbolic name
07	Full sort	SAM, variable	SAM, variable	Difference between DIN/EBCDIC sorting
08	Full sort	SAM, variable	SAM, variable	MODIFY-CODE: user-own character sequence
09	Full sort	SAM, variable	SAM, variable	Sorting with extended character sets
10	Full sort	SAM, variable	SAM, variable	Summation/exclusion of certain records
11	Selection sort	SAM, variable	SAM, variable	2 sort fields, 1 remainder field
12	Selection sort	SAM, fixed	SAM, fixed	Sort field definition in binary format
13	Selection sort	POSIX	POSIX	1 sort field, 2remainder fields
14	Tag sort	SAM, fixed	SAM, fixed	SORT-TYPE= *TAG-TRAILER
15	Merge	SAM, variable	SAM, variable	3 sorted input files MERGE-RECORDS

10.4.2 Connection of user routines

16	Full sort	SAM, fixed	SAM, fixed	INPUT user exit
17	Full sort	SAM, variable	SAM, variable	OUTPUT user exit
18	Full sort	SAM, fixed	SAM, fixed	PHYSICAL-TRANSLATE user exit
19	Full sort	SAM, variable	SAM, variable	VIRTUAL-TRANSLATE user exit

10.4.3 SORT as a subroutine

20	Full sort	SAM, fixed	SAM, fixed	Control statements passed at level 0
21	Full sort	SAM, fixed	SAM, fixed	Control statements passed at level 1
22	Full sort	SAM, variable	SAM, variable	SORT access method SRTZM
23	Multiple sort (full/selection sort)	SAM, fixed	SAM, fixed	SORT access method SRTZM

10.4.4 Sorting according to Unicode

24	Full sort	SAM, variable	SAM, variable	Unicode character format

10.5 Examples

Various files are used for demonstration purposes in the examples. The contents of the unsorted files are identical if the first part of the file name is also identical. The file properties do, however, change in various examples, because ultimately the type of the file determines the position of the data. The file type is therefore contained within the file name.
Thus, for example, the file with RESTAURANT as the first part of its name makes use of the following files:

Attributes	File name
SAM file with fixed record length	RESTAURANT.SAM.FIX
SAM file with variable record length	RESTAURANT.SAM
ISAM file with variable record length	RESTAURANT.ISAM

Variable files therefore indicate just the access type in the file name, while files containing fixed-length records have been given the suffix FIX.

The output files are identified accordingly, but with the string SORT appended to their names.
RESTAURANT.ISAM.SORT is thus the sorted file corresponding to RESTAURANT.ISAM.
As the contents of the input file do not change, a printout is given at the end of this chapter.

10.5.1 Example 1: Full sort of fixed format records

Input: SAM file RESTAURANT.SAM.FIX with fixed record format

Output: SAM file RESTAURANT.SAM.FIX.SORT with fixed record format

Exercise:

- Determine position of sort field
- Call SORT
- Enter sort command: SORT-RECORDS FIELDS=*FIELDS-EXPLICIT

Structure of the input file RESTAURANT.SAM.FIX

Restaurant name (Sort field)	Street	Tel.	Cuisine	
1	21	48	56	66

The complete contents of this file are listed in section “Contents of the example files” on page 383.

File attributes of the file RESTAURANT.SAM.FIX

```
/show-file-attributes file-name=restaurant.sam.fix, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.RESTAURANT.SAM.FIX
%  ------------------------------ ORGANIZATION ------------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (F,N)       REC-SIZE   = 66
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K          SO-MIGR    = *ALLOWED
```

Trace listing

```
/start-sort
%  SRT1001  2014-10-12/14:16:58/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=restaurant.sam.fix, -
//             output-file=restaurant.sam.fix.sort
//sort-records fields=*field-explicit(position=1,length=20)
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................10 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................10
%  SRT1002  2014-10-12/14:16:58/000000.27 SORT/MERGE COMPLETED
```

Contents (extract) of the output file RESTAURANT.SAM.FIX.SORT

```
1                    21                             48       56         66
```

10.5.2 Example 2: Full sort of variable format records

Input: SAM file RESTAURANT.SAM with variable record format

Output: SAM file RESTAURANT.SAM.SORT with variable record format

Exercise:

- Determine position of sort field
- Call SORT
- Enter sort command: SORT-RECORDS FIELDS=FIELDS-EXPLICIT

The records of the input file RESTAURANT.SAM are structured as follows:

RL	Restaurant name	Street (sort field)	Tel.	Cuisine
1	5	25	52	60 70
	Orlando‘s	Thompson Street 62	220061	Italian

The complete contents of this file are listed in section “Contents of the example files” on page 383.

File attributes of the file RESTAURANT.SAM

```
/show-file-attributes file-name=restaurant.sam, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.RESTAURANT.SAM
%  ------------------------------ ORGANIZATION -----------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (V,N)       REC-SIZE   = 0
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K           SO-MIGR    = *ALLOWED
```

RESTAURANT.SAM is a file containing variable-length records. A 4-byte record length (RL) field must therefore be added to the position of the first data byte. The “O” in “Orlando’s” is thus at position 5 in the record.

Trace listing

```
/start-sort
%  SRT1001  2014-10-12/14:17:05/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=restaurant.sam,output-file=restaurant.sam.sort
//sort-records fields=*field-explicit(position=25,length=27)
//set-record-attributes output=*variable(maximum-record-size=74)
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................10 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................10
%  SRT1002  2014-10-12/14:17:06/000000.29 SORT/MERGE COMPLETED
```

Contents of the output file RESTAURANT.SAM.SORT (extract)

RL	Restaurant name	Street (sort field)	Tel.	Cuisine
1	5	25	52	60 74
	Golden Fleece	Arran Street 44	242437	Yugoslavian
	Java	Hope Street 51	522221	Indonesian
	Le Gourmet	Lime Street 46	505397	French

```
/show-file-attributes file-name=restaurant.sam.sort, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.RESTAURANT.SAM.SORT
%  ------------------------------ ORGANIZATION ------------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (V,N)       REC-SIZE   = 74
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K          SO-MIGR    = *ALLOWED
```

The record length specified via SET-RECORD-ATTRIBUTES is entered in the catalog by SORT.

10.5.3 Example 3: Full sort of an ISAM input file into a SAM output file

Input: ISAM file RESTAURANT.ISAM with variable record format

Output: SAM file RESTAURANT.ISAM.SORT.SAM with variable record format

Exercise:

– Determine position of sort field

– Call SORT

– Enter sort command: SORT-RECORDS FIELDS=*FIELDS-EXPLICIT

– Convert ISAM file into SAM file

Comment

Two things must be taken into account in the following sort run:

1. the file types are different (ISAM input file, SAM output file).
2. the position of the first data byte is shifted to the right by the number of bytes in the record length field and the ISAM key.

The first point has an impact on the SET-FILE-LINK command, the second on how the position of the sort field is calculated.

By default, SORT sets up the file assigned as SORTOUT with the same file attributes as the SORTIN file. If other attributes are desired, the new type must be specified explicitly. This is accomplished in the SET-FILE-LINK command by means of the operand entry ACCESS-METHOD=*SAM.

Contents of the input file RESTAURANT.ISAM

RL	ISAM key	Restaurant name	Street (sort field)	Tel.	Cuisine
1	5	13	33	60	68
	00010000	Orlando's	Thompson Street 62	220061	Italian
	00020000	Java	Hope Street 51	522221	Indonesian
	00030000	Golden Fleece	Arran Street 44	242437	Yugoslavian
	00040000	Le Gourmet	Lime Street 46	505397	French
	00050000	Palenque Mexico	Millwood Drive 2	980149	Mexican
	00060000	Strawberry	Sauchiehall Street 8	595521	Vegetarian
	00070000	Persepolis	Salford Square 20	597004	Persian
	00080000	Vietnam	Thurston Street 47	522518	Vietnamese
	00090000	Chayota's	Thurston Street 60	292742	Japanese
	00100000	Willi's Bar	Westland Street 113	748293	German

Trace listing

```
/add-file-link link-name=sortin,file-name=restaurant.isam
/create-file file-name=restaurant.isam.sort.sam
/set-file-link link-name=sortout,file-name=restaurant.isam.sort.sam, -
/              access-method=sam
/start-sort
%  SRT1001  2014-10-12/14:17:13/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//sort-records fields=*field-explicit(position=13,length=20)
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................10 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................10
%  SRT1002  2014-10-12/14:17:14/000000.23 SORT/MERGE COMPLETED
```

Structure and contents of the output file RESTAURANT.ISAM.SORT.SAM

RL	original ISAM key	Restaurant name	Street (sort field)	Tel.	Cuisine
1	5	13	33	60	68
	00090000	Chayota‘s	Thurston Street 60	292742	Japanese
	00030000	Golden Fleece	Arran Street 44	242437	Yugoslavian
	00020000	Java	Hope Street 51	522221	Indonesian
	00040000	Le Gourmet	Lime Street 46	505397	French
	00010000	Orlando‘s	Thompson Street 62	220061	Italian
	00050000	Palenque Mexico	Millwood Drive 2	980149	Mexican
	00070000	Persepolis	Salford Square 20	597004	Persian
	00060000	Strawberry	Sauchiehall Street 8	595521	Vegetarian
	00080000	Vietnam	Thurston Street 47	522518	Vietnamese
	00100000	Willi‘s Bar	Westland Street 113	748293	German

File attributes of the output file RESTAURANT.ISAM.SORT.SAM

```
/show-file-attributes file-name=restaurant.isam.sort.sam, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.RESTAURANT.ISAM.SORT.SAM
%  ------------------------------ ORGANIZATION -----------------------------
%  FILE-STRUC = SAM          BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE   IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (V,N)        REC-SIZE   = 2044
%  AVAIL      = *STD
%  WORK-FILE  = *NO          F-PREFORM  = *K           SO-MIGR    = *ALLOWED
```

Comment

The original ISAM key forms part of the data record. Normally, when the file is processed with editors (e.g. EDT), the key is not visible; but in SAM files it has no DMS function and is therefore visible.
The ISAM keys are an indicator for the changed (sorted) order of the data records. The factor which determines the order of the records, however, is the sort field, which begins at position 13.

10.5.4 Example 4: Full sort of ISAM files with variable record format

Input: ISAM file RESTAURANT.ISAM with variable record format

Output: ISAM file RESTAURANT.ISAM.SORT with variable record format

Exercise:

– Record the ISAM characteristics of the input and output files

SORT creates the output file with new file attributes. The sort key determines the position and length of the ISAM key of the output file.

If ISAM files are used as the output files of a sort run, the following important rules apply:

– sorting must be in ascending order only

– the sort fields must be contiguous

Contents of the input file RESTAURANT.ISAM (extract)

RL	ISAM key	Restaurant name	Street (sort field)	Tel.	Cuisine
1	5	13	33	60	68
	00010000	Orlando's	Thompson Street 62	220061	Italian
	00020000	Java	Hope Street 51	522221	Indonesian
	00030000	Golden Fleece	Arran Street 44	242437	Yugoslavian
	00040000	Le Gourmet	Lime Street 46	505397	French

The complete contents of this file are listed in section “Contents of the example files” on page 383.

File attributes of the file RESTAURANT.ISAM

```
/show-file-attributes file-name=restaurant.isam, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.RESTAURANT.ISAM
%  ------------------------------ ORGANIZATION ------------------------------
%  FILE-STRUC = ISAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE   IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (V,N)        REC-SIZE   = 0
%  KEY-LEN    = 8            KEY-POS    = 5
%  AVAIL      = *STD
%  WORK-FILE  = *NO          F-PREFORM  = *K          SO-MIGR    = *ALLOWED
```

The file has the typical key position and length for the editor EDT.

Trace listing

```
/start-sort
%  SRT1001  2014-10-12/14:17:21/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=restaurant.isam,output-file=restaurant.isam.sort
//sort-records fields=*field-explicit(position=13,length=20)
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................10 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................10
%  SRT1002  2014-10-12/14:17:22/000000.27 SORT/MERGE COMPLETED
```

In the output file in ISAM format, the definition of the sort field corresponds to the new ISAM key. Depending on the length and position of the new key, the output file cannot be processed with the EDT editor. In this example neither the position nor the length of the key conforms to the sizes assumed by EDT.

Structure and contents of the output file RESTAURANT.ISAM.SORT (extract)

RL	(ISAM key of the input file)	Restaurant name (ISAM key)	Street	Tel.	Cuisine
1	5	13	33	60	68
	00090000	Chayota‘s	Thurston Street 60	292742	Japanese
	00030000	Golden Fleece	Arran Street 44	242437	Yugoslavian
	00020000	Java	Hope Street 51	522221	Indonesian

File attributes of the output file RESTAURANT.ISAM.SORT

```
/show-file-attributes file-name=restaurant.isam.sort, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.RESTAURANT.ISAM.SORT
%  ------------------------------ ORGANIZATION -----------------------------
%  FILE-STRUC = ISAM        BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (V,N)       REC-SIZE   = 2048
%  KEY-LEN    = 20          KEY-POS    = 13
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K          SO-MIGR    = *ALLOWED
```

The ISAM key position and key length now correspond to the definitions of the sort field.

10.5.5 Example 5: Full sort of multiple files with variable record format

Input: ISAM file CULTURE.ISAM.1 with variable record format
ISAM file CULTURE.ISAM.2 with variable record format
ISAM file CULTURE.ISAM.3 with variable record format

Output: SAM file CULTURE.SAM.SORT with variable record format

Exercise:

– Assign multiple input files for sort and merge run

– Verify whether assignment has taken place

– Change characteristics of the output file

Preliminary remark

It is possible to sort multiple files in one operation and place the sorted result in a single output file. If the individual input files are already sorted according to a common sort criterion, a merge run should be started. SORT provides the MERGE-RECORDS command for this purpose.

Contents of the input file CULTURE.ISAM.1 (extract)

RL	ISAM key	Restaurant name (sort field)	Street	Tel.	Cuisine
1	5	13	34	56	66
	00010000	Aquitaine	Acacia Avenue 39	284028	French
	00020000	August Gardens	Newton Street 16	2604106	Argentinian
	00030000	Bosna	Freeling Street 11	64115447	Yugoslavian

The complete contents of this and the other files are listed in section “Contents of the example files” on page 383.

Trace listing

```
/add-file-link link-name=sortin01,file-name=culture.1.isam
/add-file-link link-name=sortin02,file-name=culture.2.isam
/add-file-link link-name=sortin03,file-name=culture.3.isam
/create-file file-name=culture.sam.sort
/add-file-link link-name=sortout,file-name=culture.sam.sort, -
/              access-method=*sam
/show-file-link
%-- LINK-NAME --------- FILE-NAME ------------------------------------------
%   SORTIN01            :CTID:$EXAMPLE.CULTURE.1.ISAM
%   SORTIN02            :CTID:$EXAMPLE.CULTURE.2.ISAM
%   SORTIN03            :CTID:$EXAMPLE.CULTURE.3.ISAM
%   SORTOUT             :CTID:$EXAMPLE.CULTURE.SAM.SORT
/start-sort
%  SRT1001  2014-10-12/14:17:30/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//sort-records fields=*field-explicit( -
//                    position=13,length=20,sorting-order=*descending)
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................15 (FROM 01)
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................16 (FROM 02)
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................14 (FROM 03)
%  SRT1017  RECORDS TO BE SORTED/MERGED:......................45
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................45
%  SRT1002  2014-10-12/14:17:31/000000.26 SORT/MERGE COMPLETED
```

SORT issues messages indicating the number of records read from each input file, the total number of records to be sorted and the number of output records.

Contents of the output file CULTURE.SAM.SORT (extract)

```
00150000Zung-Hua            Bond Street 33        555320    Chinese
00140000Zigzag              Taylor Drive 72       226750    Argentinian
00160000Ziggy's             Leslie Street 72      390092    Argentinian
00150000Zagreb              Sutherland Drive 8    6515509   Yugoslavian
00130000Why Not?            Wimbledon Drive 11    399936    French
00120000Watermill           Leslie Street 33      348000    Swiss
00140000Veracruz            Landers Street 207    5702520   Mexican
00130000Venezia             Landers Street 84     847414    Italian
00110000Torino              Gardener Lane 8       469571    Italian
00100000Tivoli              Wilberforce Drive 52  221274    Italian
00120000Tivoli              Wilberforce Drive 52  221274    Italian
00110000Tai Tung            Acacia Avenue 77      281104    Chinese
00100000Sultana             Fulton Street 28      332871    Indian
00140000Spiros              Upman Street 65       366883    Greek
00130000Slavonia            Allcock Street 16     564906    Yugoslavian
00120000Siracusa            Polson Street 33      770613    Italian
00090000Scorpio             Leslie Street 35      399897    Greek
00110000Saint George's      Upman Street 67       363666    English
000800000patija             Robertson Street 2    268353    Yugoslavian
00100000Nitaya              Thompson Street 19    197772    Thai
   .       .                   .                   .         .
   .       .                   .                   .         .
   .       .                   .                   .         .
00060000Don Quixote         Billington Street 6   342318    Spanish
00050000Datscha             King Street 3         341218    Russian
00040000China House         May Street 20         531620    Chinese
00030000Canton              Theresa Street 49     522185    Chinese
00020000Buenos Aires        Zoo Road 22           779646    Argentinian
00040000Bouillabaisse       Falcon Street 10      297909    French
00030000Bosna               Freeling Street 11    64115447  Yugoslavian
00050000Bologna             Leslie Street 23      393939    Italian
00040000Baltic Grillhouse   Dartford Road 33      554401    Yugoslavian
00020000August Gardens      Newton Street 16      2604106   Argentinian
00030000Aubergine           Penman Street 19      674829    Argentinian
00020000Auberge             Richmond Drive 15     347577    French
00010000Asado Steak         Taylor Drive 1        294577    Argentinian
00010000Aquitaine           Acacia Avenue 39      284028    French
00010000Alcazar's           Doubleway Drive 39    8111590   Argentinian
```

The original ISAM key additionally printed here is an indicator of the changed order from the individual files. Duplicate ISAM key references also occur (cf. the final three records) because ISAM keys are created separately for each file.

10.5.6 Example 6: Full sort (input file = output file)

Input: SAM file RESTAURANT.SAM.INOUT with variable record format

Output: SAM file RESTAURANT.SAM.INOUT with variable record format

Exercise:

– Assign the same file as both input and output file

– Use a symbolic field name

The file RESTAURANT.SAM.INOUT has the same contents as RESTAURANT.SAM. The file for this example was created by copying RESTAURANT.SAM.

Trace listing

```
/start-sort
%  SRT1001  2014-10-12/14:17:38/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=restaurant.inout,output-file=restaurant.inout
//add-symbolic-name fields=restaurantname(position=5,length=20)
//sort-records fields=*field-symbolic(restaurantname)
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................10 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................10
%  SRT1002  2014-10-12/14:17:39/000000.27 SORT/MERGE COMPLETED
```

Contents of the output file (extract)

```
Chayota‘s           Thurston Street 60         292742  Japanese
Golden Fleece       Arran Street 44            242437  Yugoslavian
Java                Hope Street 51             522221  Indonesian
```

Comment

The same name is used in the ASSIGN-FILES statement for the input and output files. This means that the input file is made into the output file.
The sort criterion is defined using a symbolic name. Instead of the position and length having to be defined via *FIELD-EXPLICIT, the already existing definition can be referenced by means of the ADD-SYMBOLIC-NAME statement, using the previously declared user-defined symbolic name.
Upon completion of the sort operation, the result is stored under the same name as the input file.

10.5.7 Example 7: Full sort EBCDIC to DIN standard for text ordering

Input: SAM file DEUTSCH with variable record format

Output: SAM file DEUTSCH.DIN with variable record format

Exercise:

– Change the format from the preset default format

– Compare the different results

Preliminary remark

Many sort operations require a different sorting order than that defined by EBCDIC.
An example of this is alphabetic sorting according to the DIN collating sequence.
SORT offers various options with the FORMAT parameter, including one which caters for sorting according to the DIN standard. The following example illustrates how sorting according to DIN produces a different order to EBCDIC.

The DIN order is characterized as follows:
– case-insensitive (no distinction made between uppercase and lowercase)
– the umlauts ä, ö, ü are treated as ae, oe and ue
– the voiceless S (ß) is treated as ss.

This form of sorting is intended, for example, for sorting lists of names which include umlauts. The following example shows the effects of DIN sorting as compared with the EBCDIC sorting order.

Trace listing

```
/sort-file input-files=deutsch,output-file=deutsch.din, -
/          fields=*field-explicit(position=5,length=8,format=*ebcdic-din)
%  SRT1001  2014-10-12/14:17:21/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
%  SRT1016  SORT/MERGE INPUT RECORDS:.............................34 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:............................34
%  SRT1002  2014-10-12/14:17:22/000000.27 SORT/MERGE COMPLETED
```

Comment

As a simple sort is to be performed in this instance, the SORT-FILE command is used. Note the use here of the option FORMAT=*EBCDIC-DIN , which defines the special DIN sorting order.

Result of different sort runs using the input file DEUTSCH

Input file	EBCDIC order	DIN order
scherzen	allgemein	Abendland
allgemein	astronomisch	ändern
überlegen	brasilianisch	allgemein
mexikanisch	bretonisch	Allgemeinheit
Mehrzahl	dichterisch	Astronomie
oder	englisch	astronomisch
Trennung	französisch	Biologie
Verhältniswort	Österreich	brasilianisch
Stilkunde	Übertragung	Brasilien
brasilianisch	mechanisch	bretonisch
philosophisch	medizinisch	Deutschland
medizinisch	mexikanisch	dichterisch
ändern	oder	Druckersprache
dichterisch	philosophisch	englisch
Druckersprache	scherzen	französisch
Astronomie	scherzhaft	Franzosen
mechanisch	ändern	Landwirtschaft
Übertragung	öffentlich	Mechanik
scherzhaft	überlegen	mechanisch
bretonisch	Abendland	medizinisch
Biologie	Allgemeinheit	Mehrzahl
Brasilien	Astronomie	mexikanisch
Allgemeinheit	Biologie	oder
öffentlich	Brasilien	öffentlich
astronomisch	Deutschland	Österreich
Mechanik	Druckersprache	philosophisch
Franzosen	Franzosen	Scherz
englisch	Landwirtschaft	scherzen
Landwirtschaft	Mechanik	scherzhaft
französisch	Mehrzahl	Stilkunde
Abendland	Scherz	Trennung
Österreich	Stilkunde	überlegen
Scherz	Trennung	Übertragung
Deutschland	Verhältniswort	Verhältniswort

When lists of names are sorted, it does not make sense for all lowercase letters to be placed before the uppercase “A” and umlauts between the lowercase letters. In the DIN-based sort, these special characters have been arranged in the correct alphabetical position.

10.5.8 Example 8: Full sort using FORMAT=*MODIFY-CODE

Input: SAM file MODCOD.INPUT with variable record format

Output: SAM file MODCOD.OUTPUT with variable record format

Exercise:

– Define a non-EBCDIC character ordering sequence

– Enter the character definition as a letter pair or as pairs of hexadecimal characters in the MODIFY-CODE command

Contents of the input file MODCOD.INPUT

```
U
o
A
u
0
a
ö
Ä
Ö
ä
Ü
```

Comment

This option may be used when predefined sorting sequences do not match the desired order (e.g. for sorting lists containing French accents or German umlauts).

This file should illustrate this option and indicate that special characters require special handling. The sort order is defined in the way described below:

The character ’a’ should be placed immediately after ’A’, ’Ä’ (X’8B’) after ’a’ (X’81’) and ’ä’ (X’AB’) after ’Ä’. The same procedure applies to the characters ’O’, ’o’ (X’96’), ’Ö’ (X’8C’) and ’ö’ (X’AC’) as well as to ’U’, ’u’ (X’A4’), ’Ü’ (X’8D’) and ’ü’ (X’AD’). The hexadecimal specifications are based on the EBCDIC.SRV.10 code.

Trace listing

```
/start-sort
%  SRT1001  2014-10-12/14:17:38/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=modcode.input,output-file=modcode.output
//sort-records fields=*field-explicit( -
//                    position=5,length=1,format=*modify-code)
//modify-code sequences= -
//        (c'Aa',x'818b',x'8bab',c'Oo',x'968c',x'8cac',c'Uu',x'a48d',x'8dad')
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................12 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................12
%  SRT1002  2014-10-12/14:17:39/000000.27 SORT/MERGE COMPLETED
```

Contents of the output file MODCOD.OUTPUT

```
A
a
Ä
ä
O
o
Ö
ö
U
u
Ü
ü
```

10.5.9 Example 9: Full sort using FORMAT=*EXTENDED-CHARACTER and FORMAT=*TRANSLATE-CHARACTER

Input: SAM file XHCS.SAM with variable record format

Output: SAM file XHCS.SAM.SORT.EXCHAR with variable record format
SAM file XHCS.SAM.SORT.TRCHAR with variable record format

Exercise:

– Call SORT

– Decide which sort format should be used

– Enter sort statements:

```
SORT-RECORDS FIELDS=*FIELD-EXPLICIT(...FORMAT=*EXTENDED-CHARACTER)
SORT-RECORDS FIELDS=*FIELD-EXPLICIT(...FORMAT=*TRANSLATE-CHARACTER)
```

Contents of the input file XHCS.SAM

RL	Sort field	
1	5	9

```
orde ntlich
anbi eten
unor dentlich
Ärge r
Öffe ntlichkeit
Über fluss
Unfa ll
Orga nisation
über flüssig
Aber witz
ärge rlich
öffe ntlich
```

File attributes of the input file

```
/show-file-attributes file-name=xhcs.sam, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.XHCS.SAM
%  ------------------------------ ORGANIZATION -----------------------------
%  FILE-STRUC = SAM          BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE   IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (V,N)        REC-SIZE   = 0
%  COD-CH-SET = EDF03DRV
%  AVAIL      = *STD
%  WORK-FILE  = *NO          F-PREFORM  = *K           SO-MIGR    = *ALLOWED
```

Notes on sorting with different format entries

With some sorts it is necessary to enter a special sort format in order to maintain a sequence which corresponds to the local rules. This may apply when a CCS is used.

The file shown here is to be sorted according to German rules. This can be seen by the umlauts, which sometimes require a special sort procedure. The following points should be considered before beginning the sort:

– If no format is entered, SORT uses the CCS found in the catalog entry of the input file. This may cause umlauts to be processed as special characters and placed at the beginning of the sort list.

– When sorting according to the CCS code sequence (here EDF03DRV), umlauts are always placed after vowels (FORMAT=*EXTENDED-CHARACTER). In other CCSs, other rules may be specified.

– When sorting with the format entry *TRANSLATE-CHARACTER, characters are replaced in accordance with specified equating tables (here the tables on page 51f were used) and are sorted using the code sequence of the CCS (here EDF03DRV). Using predefined or self-defined tables ensures accurate sorting according to your personal specifications.

– The DIN sort procedure previously discussed in “Example 7: Full sort EBCDIC to DIN standard for text ordering” on page 315 converts umlauts and makes lowercase and uppercase letters equivalent, and sorts according to DIN guidelines. No conversion tables can be defined.

The following example illustrates this.

The letter sequences of the original text are sorted according to the entries in the FORMAT operand:

Original	After sorting with EXTENDED-CHARACTER (CCS: EDF03DRV)	After sorting with TRANSLATE-CHARACTER (CCS: EDF03DRV)
oooooooo öööööööö 00000000 ÖÖÖÖÖÖÖÖ	oooooooo 00000000 öööööööö ÖÖÖÖÖÖÖÖ	öööööööö ÖÖÖÖÖÖÖÖ oooooooo 00000000

When sorting with FORMAT=*EXTENDED-CHARACTER, the vowels come first (and lowercase letters before uppercase letters) and then the umlauts.

When using FORMAT=*TRANSLATE-CHARACTER, the character string “öööööööö” is processed as “oeoeoeoeoeoeoeoe” and is therefore alphabetized before “oooooooo”. Thus, in this example, the O umlauts (Ös) come before a double “O”. In a series of “aaaaa” and “äääää” the “aaaaa”s come before the “äääää”s, since “aa” comes before “ae”.

Trace listing

```
/start-sort
%  SRT1001  2014-10-12/14:17:38/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=xhcs.sam,output-file=xhcs.sam.exchar
//sort-records fields=*field-explicit( -
//                    position=5,length=4,format=*extended-character)
//set-record-attributes output=*variable(maximum-record-size=40)
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................12 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................12
%  SRT1002  2014-10-12/14:17:39/000000.31 SORT/MERGE COMPLETED
```

```
/start-sort
%  SRT1001  2014-10-12/14:19:07/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=xhcs.sam,output-file=xhcs.sam.trchar
//sort-records fields=*field-explicit( -
//                    position=5,length=4,format=*translate-character)
//set-record-attributes output=*variable(maximum-record-size=40)
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................12 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................12
%  SRT1002  2014-10-12/14:19:26/000000.27 SORT/MERGE COMPLETED
```

Results of the sort procedure (without record length field)

Input file XHCS.SAM	Output file XHCS.SAM.SORT.EXCHAR	Output file XHCS.SAM.SORT.TRCHAR
ordentlich	anbieten	ärgerlich
anbieten	Aberwitz	Ärger
unordentlich	ärgerlich	Aberwitz
Ärger	Ärger	anbieten
Öffentlichkeit	ordentlich	öffentlich
Überfluss	Organisation	Öffentlichkeit
Unfall	öffentlich	ordentlich
Organisation	Öffentlichkeit	Organisation
überflüssig	unordentlich	Überfluss
Aberwitz	Unfall	überflüssig
ärgerlich	überflüssig	unordentlich
öffentlich	Überfluss	Unfall

File attributes of the sorted files

```
/show-file-attributes file-name=xhcs.sam., -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.XHCS.SAM.EXCHAR
%  ------------------------------ ORGANIZATION -----------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
% IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (V,N)       REC-SIZE   = 40
%  COD-CH-SET = EDF03DRV
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K          SO-MIGR    = *ALLOWED
%0000000003 :CTID:$EXAMPLE.XHCS.SAM.TRCHAR
%  ------------------------------ ORGANIZATION -----------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (V,N)       REC-SIZE   = 40
%  COD-CH-SET = EDF03DRV
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K          SO-MIGR    = *ALLOWED
```

10.5.10 Example 10: Full sort with summation and SELECT-INPUT-RECORDS

Input: SAM file CLIENT.SELECT with variable record format

Output: SAM file CLIENT.SELECT.SORT with variable record format

Exercise:

– Sort records

– Define part of a record as a number

– Sum the defined numbers in identical records

– Select records for summation and sorting

– Extend sum field to prevent overflow

Preliminary remark

An application for the example chosen here might be the preparation of a balance sheet (summation) by customer number, without taking account of orders placed in a particular month.

The following specifications are therefore required for this example:

– on which field is the sort to be based (SORT-RECORDS)?

– what is the position of the sum field (SUM-RECORDS)?

– does the expected sum exceed the existing field length?

– which records are to be excluded from the sort (SELECT-INPUT-RECORDS)?

The result of the summation can be seen in the file CLIENT.SELECT.SUM.

Contents of the input file CLIENT.SELECT

RL	Sort field	Sum field		Match field	
1	5	14	21	32	38
	CLIENT-A	700000	MOTOR	MAY	INPUTRECORD1
	CLIENT-F	000083	AUTO	JAN	INPUTRECORD2
	CLIENT-E	000700	MOTOR	AUG	INPUTRECORD3
	CLIENT-D	076000	AUTO	JUN	INPUTRECORD4
	CLIENT-B	006900	MOTOR	AUG	INPUTRECORD5
	CLIENT-G	000070	AUTO	JAN	INPUTRECORD6
	CLIENT-J	006700	AUTO	JUN	INPUTRECORD7
	CLIENT-A	800000	MOTOR	AUG	INPUTRECORD8
	CLIENT-K	075600	AUTO	JAN	INPUTRECORD9
	CLIENT-S	000099	AUTO	JUN	INPUTRECORD10
	CLIENT-C	000001	MOTOR	AUG	INPUTRECORD11
	CLIENT-B	010000	AUTO	JAN	INPUTRECORD12
	CLIENT-A	030000	AUTO	JUN	INPUTRECORD13
	CLIENT-W	008700	AUTO	JAN	INPUTRECORD14
	CLIENT-D	024000	MOTOR	AUG	INPUTRECORD15
	CLIENT-E	001350	AUTO	JAN	INPUTRECORD16
	CLIENT-C	999999	AUTO	JAN	INPUTRECORD17
	CLIENT-J	000305	MOTOR	AUG	INPUTRECORD18
	CLIENT-X	123456	AUTO	JAN	INPUTRECORD19
	CLIENT-A	600000	MOTOR	MAY	INPUTRECORD20
	CLIENT-T	000058	AUTO	JUN	INPUTRECORD21
	CLIENT-Y	005000	AUTO	JAN	INPUTRECORD22

Trace listing

```
/start-sort
%  SRT1001  2014-10-12/09:31:48/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=client.select,output-file=client.select.sum
//sort-records fields=*field-explicit(position=5,length=8)
//sum-records fields=*field-explicit( -
//            position=14,length=6,format=*zoned-decimal,field-extension=2)
//select-input-records condition=(32,3,ch<>'JUN')
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.............................22 (FROM 01)
%  SRT1024  DELETED SELECT-INPUT-RECORDS RECORDS:.........................5
%  SRT1017  RECORDS TO BE SORTED/MERGED:.........................17
%  SRT1030  SORT/MERGE OUTPUT RECORDS:...........................12
%  SRT1020 DELETED SUM RECORDS:..................................5
%  SRT1002  2014-10-12/09:32:00/000000.29 SORT/MERGE COMPLETED
```

Contents of the file CLIENT.SELECT.SUM

RL		Sum field with sum field extension			
1	5	14	23	34	40 52

```
     CLIENT-A 02100000 MOTOR      MAY    INPUTRECORD1
     CLIENT-B 00016900 AUTO       JAN    INPUTRECORD12
     CLIENT-C 01000000 MOTOR      AUG    INPUTRECORD11
     CLIENT-D 00024000 MOTOR      AUG    INPUTRECORD15
     CLIENT-E 00002050 AUTO       JAN    INPUTRECORD16
     CLIENT-F 00000083 AUTO       JAN    INPUTRECORD2
     CLIENT-G 00000070 AUTO       JAN    INPUTRECORD6
     CLIENT-J 00000305 MOTOR      AUG    INPUTRECORD18
     CLIENT-K 00075600 AUTO       JAN    INPUTRECORD9
     CLIENT-W 00008700 AUTO       JAN    INPUTRECORD14
     CLIENT-X 00123456 AUTO       JAN    INPUTRECORD19
     CLIENT-Y 00005000 AUTO       JAN    INPUTRECORD22
```

Comment

The output file does not contain the records separated out by means of the SELECT-INPUT-RECORDS statement. Equally, all records which match the sort criterion are represented by just a single record in the file. Note, however, that the sum fields of these identical records are added together. The sum for the sort criterion “client” is now stored in the sum field.

Example

The sum fields for CLIENT-A contain the entries 700000, 800000, 30000 and 600000. However, the entry 30000 refers to JUN, and no records for that month were to be included in the sort operation. The sum of the remaining three values is 2100000. This result was entered in the sum field for CLIENT-A.

Specifying an extension of the sum field avoids the risk of overflows.

10.5.11 Example 11: Selection sort of variable format records

Input: SAM file RESTAURANT.SAM with variable record format

Output: SAM file RESTAURANT.SAM.SELECT with variable record format

Exercise:

– Define two sort fields

– Define a remainder field

– Exclude part of the input record (selection sort)

– Change the file attributes predefined by SORT

Preliminary remark

This example is meant to demonstrate the facilities SORT offers for

– constructing output records from individual fields of the input records

– changing the file attributes of the output file within SORT.

The latter possibility is especially important in this example. In a selection sort, SORT generates a file containing fixed-length records, irrespective of the type of the input file. If the output file is to be processed further with an editor such as EDT, then in most cases a file containing variable-length records is preferred.
File attributes are changed by means of the SET-RECORD-ATTRIBUTES statement.

Structure of the input file RESTAURANT.SAM

RL	Restaurant name (2nd sort field)	Street (Remainder field)	Tel.(to be dropped)	Cuisine (1st sort field)
1	5	25	52	60
	Orlando‘s	Thompson Street 62	220061	Italian

The complete contents of this file are listed in section “Contents of the example files” on page 383.

Trace listing

```
/start-sort
%  SRT1001  2014-10-12/09:36:05/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=restaurant.sam,output-file=restaurant.sam.select
//set-record-attributes output=*variable(maximum-record-size=*std), -
//                      filler=' '
//sort-records fields=( -
//             *field-explicit(position=60,length=13), -
//             *field-explicit(position=5,length=20), -
//             *remainder-explicit(position=25,length=27)), -
//             sort-type=*compound-record
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................10 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................10
%  SRT1002  2014-10-12/09:36:14/000000.27 SORT/MERGE COMPLETED
```

File attributes of the output file RESTAURANT.SAM.SELECT

```
/show-file-attributes file-name=restaurant.sam.select, -
/                     information=*parameters(organization=*yes)
%0000000003  :CTID:$EXAMPLE.RESTAURANT.SAM.SELECT
%  ------------------------------ ORGANIZATION ------------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (V,N)       REC-SIZE   = 64
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K          SO-MIGR    = *ALLOWED
```

SORT enters the sum of the length of all the fields as the maximum record length in the catalog.

Contents of the output file RESTAURANT.SAM.SELECT

RL	Cuisine (1st sort field)	Restaurant name (2nd sort field)	Street (Remainder field)
1	5	18	38 68
	French	Le Gourmet	Lime Street 46
	German	Willi's Bar	Westland Street 113
	Indonesian	Java	Hope Street 51
	Italian	Orlando's	Thompson Street 62
	Japanese	Chayota's	Thurston Street 60
	Mexican	Palenque Mexico	Millwood Drive 2
	Persian	Persepolis	Salford Square 20
	Vegetarian	Strawberry	Sauchiehall Street 8
	Vietnamese	Vietnam	Thurston Street 47
	Yugoslavian	Golden Fleece	Arran Street 44

The “Tel.” field in the records of the input file has been excluded from the output file by the selection sort.

10.5.12 Example 12: Selection sort (binary) of fixed-format records

Input: SAM file SAM.BIN.FIX with fixed record format

Output: SAM file SAM.BIN.FIX.SORT with fixed record format

Exercise:

– Specify a bit position

– Change the FORMAT parameter

Contents of the input file SAM.BIN.FIX

printable	hexadecimal	binary			
		1	2	3	Byte
		01234567	0 1234567	01234 567	Bit
NCK	D5C3D2	11010101	1 1000011	11010 010	Record 1
TUM	E3E4D4	11100011	1 1100100	11010 100	Record 2
HEL	C8C5D3	11001000	1 1000101	11010 011	Record 3
ELK	C5D3D2	11000101	1 1010011	11010 010	Record 4
			Sort field (length 1 byte + 4 bits = 12 bits)		

Trace listing

```
/start-sort
%  SRT1001  2014-10-12/12:21:05/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=sam.bin.fix,output-file=sam.bin.fix.sort
//sort-records fields=*field-explicit( -
//             position=2(bit-position=1),length=1(number-of-bits=4), -
//             format=*binary), -
//             sort-type=*compound-record
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.............................4 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:............................4
%  SRT1002  2014-10-12/12:21:17/000000.22 SORT/MERGE COMPLETED
```

Contents of the output file SAM.BIN.FIX.SORT

hexadecimal	binary		
	1	2	Byte
	0 1234567	01234 567	Bit
43D0	0 1000011	11010 000	Record 1
45D0	0 1000101	11010 000	Record 3
53D0	0 1010011	11010 000	Record 4
64D0	0 1100100	11010 000	Record 2
	*)	Sort field	*)

*) Binary zeros entered by SORT to pad out the output record to full bytes

10.5.13 Example 13: Selection sort of a POSIX file

Input: POSIX file “restaurant” with variable record format

Output: POSIX file “sorted/restaurant” with variable record format

Exercise:

– Assign POSIX files

– Define a sort field

– Define two remainder fields

– Exclude part of the input record (selection sort)

Preliminary remark

The POSIX file “restaurant” is to be sorted according to the “Cuisine” field. The output file is to be stored in the “sorted” directory under the name “restaurant”. It should now contain only the fields “Cuisine”, “Restaurant name” and “Tel.” from the input file.

Structure of the input file “restaurant”

Restaurant name (1st remainder field)	Street (to be dropped)	Tel. (2nd rem. field)	Cuisine (sort field)
1	21	48	56
Orlando's	Thompson Street 62	220061	Italian
Java	Hope Street 51	522221	Indonesian
Golden Fleece	Arran Street 44	242437	Yugoslavian
Le Gourmet	Lime Street 46	505397	French
Palenque Mexico	Millwood Drive 2	980149	Mexican
Strawberry	Sauchiehall Street 8	595521	Vegetarian
Persepolis	Salford Square 20	597004	Persian
Vietnam	Thurston Street 47	522518	Vietnamese
Chayota's	Thurston Street 60	292742	Japanese
Willi's Bar	Westland Street 113	748293	German

File attributes of the input file “restaurant”

```
/start-posix-shell
POSIX Basisshell 10.0A43 created Dec 17 2012
POSIX Shell 08.0A43 created Jul 13 2012
Copyright (C) Fujitsu Technology Solutions 2009
              All Rights reserved
.
.
.
$ls -l restaurant
-rw-r--r--   1 EXAMPLE  USROTHER     690 Jan 24 10:13 restaurant
$exit
```

Trace listing

```
/start-sort
%  SRT1001  2014-10-12/12:46:01/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files='restaurant',output-file='sorted/restaurant'
//sort-records fields=( -
//             *field-explicit(position=56,length=11), -
//             *remainder-explicit(position=1,length=20), -
//             *remainder-explicit(position=48,length=8)), -
//             sort-type=*compound-record
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................10 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................10
%  SRT1002  2014-10-12/12:50:45/000000.47 SORT/MERGE COMPLETED
```

File attributes of the output file “sorted/restaurant”

```
/start-posix-shell
POSIX Basisshell 10.0A43 created Dec 17 2012
POSIX Shell 08.0A43 created Jul 13 2012
Copyright (C) Fujitsu Technology Solutions 2009
              All Rights reserved
.
.
.
$cd sorted
$ls -l restaurant
-rw-------   1 EXAMPLE  USROTHER     400 Jan 24 12:50 restaurant
$exit
```

Contents of the output file “sorted/restaurant”

Cuisine (sort field)	Restaurant name (1st remainder field)	Tel. (2nd rem. field)
1	12	32
French	Le Gourmet	505397
German	Willi's Bar	748293
Indonesian	Java	522221
Italian	Orlando's	220061
Japanese	Chayota's	292742
Mexican	Palenque Mexico	980149
Persian	Persepolis	597004
Vegetarian	Strawberry	595521
Vietnamese	Vietnam	522518
Yugoslavian	Golden Fleece	242437

The “Street” field in the input file was not incorporated into the output file by the selection sort.

10.5.14 Example 14: Tag sort of fixed-format records

Input: SAM file RESTAURANT.SAM.FIX with fixed record format

Output: SAM file RESTAURANT.SAM.FIX.ADR with fixed record format

Exercise:

- Use of a SAM file
- Position the addresses generated by SORT at the end of the record stored in the output file

Preliminary remark

Tag sorting is a special sorting technique by means of which the records in the input file can be accessed via the records in the sorted file.
For this type of sort, SORT assigns addresses which indicate where the individual records of the input file were found. The sort type parameter determines whether the address is prefixed (SORT-TYPE=*TAG-HEADER) or appended (SORT-TYPE=*TAG-TRAILER) to the record. The address is also referred to as the retrieval address.

This type of sort can, for example, be used to create an index for a (long) SAM file in a data base.

Trace listing

```
/start-sort
%  SRT1001  2014-10-12/09:23:05/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=restaurant.sam.fix, -
//             output-file=restaurant.sam.fix.adr
//sort-records fields=*field-explicit(position=1,length=13), -
//                     sort-type=*tag-trailer
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................10 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................10
%  SRT1002  2014-10-12/09:23:22/000000.29 SORT/MERGE COMPLETED
```

Contents of the output file

Data bytes (sort field)	SAM address (in hexadecimal notation)	
	File block number	Relative record number

```
1                   14      17

Chayota‘s           000001 09
Golden Fleece       000001 03
Java                000001 02
Le Gourmet          000001 04
Orlando‘s           000001 01
Palenque Mexico     000001 05
Persepolis          000001 07
Strawberry          000001 06
Vietnam             000001 08
Willi‘s Bar         000001 0A
```

10.5.15 Example 15: Merging files

Input: SAM file CULTURE.SAM.1 with variable record format
SAM file CULTURE.SAM.2 with variable record format
SAM file CULTURE.SAM.3 with variable record format

Output: SAM file CULTURE.SAM.123 with variable record format

Preliminary remark

MERGE-RECORDS merges presorted files. The input files must already be sorted according to the sort criterion. This helps save on CPU time for the compute-intensive sorting operation that would otherwise be necessary.

Contents of the input file CULTURE.ISAM.1 (extract)

RL	Restaurant name (Sort field)	Street	Telephone	Cuisine
1	5	26	48	58
	Aquitaine	Acacia Avenue 39	284028	French
	August Gardens	Newton Street 16	2604106	Argentinian
	Bosna	Freeling Street 11	64115447	Yugoslavian

The complete contents of this and the other files are listed in section “Contents of the example files” on page 383.

Trace listing

```
/start-sort
%  SRT1001  2014-10-12/16:19:30/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=(culture.sam.1,culture.sam.2,culture.sam.3), -
//             output-file=culture.sam.123
//merge-records fields=*field-explicit(position=5,length=21)
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................15 (FROM 01)
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................16 (FROM 02)
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................14 (FROM 03)
%  SRT1017  RECORDS TO BE SORTED/MERGED:......................45
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................45
%  SRT1002  2014-10-12/16:19:31/000000.26 SORT/MERGE COMPLETED
```

10.5.16 Example 16: INPUT user exit

Full sort of fixed-format records

Input:	SAM file UE.EXIT.INPUT with fixed record format
Output:	SAM file UE.EXIT.OUTPUT with fixed record format
Library:	SORT.EXAMPLE.LIB (user-defined)
Module:	E15
User exit:	INPUT (PARAMETER-MODE=*ANY)

Exercise:

– Link in a module from a library

– Select the user exit

– Define the addressing method

Description of the user module

In this example, records are to be deleted, inserted and modified by a user routine. The routine is activated for each input record and is meant to perform the following actions:

– Delete all records beginning with the digit 8.

– Modify all records beginning with the digit 9 by replacing the first character with ’X’.

– Insert a record at the position of the first record beginning with the digit 7. This operation is not to be repeated for other records.

A common feature of all user routines for user exits is that they have to be stored in module libraries.

Source text

```
E15      START
         PRINT NOGEN
         ENTRY E15SEL
         USING E15SEL,15
E15SEL   SAVE  (14,12)              SAVE REGISTERS
         L     7,12(1)              LOAD ADDRESS OF ACTION WORD
E1       CLC   0(4,1),=XL4'00'      END?
         BE    ENDE                 YES
         L     8,0(1)               LOAD ADDRESS OF RECORD
         CLI   0(8),C'8'            1ST POSITION=8?
         BE    DELETE              YES
         CLI   0(8),C'9'            1ST POSITION=9?
         BE    MODIFY               YES
SWITCH   NOP   NORMAL
         CLI   0(8),C'7'            1ST POSITION=7?
         BE    INSERT
*        N O   M O D I F I C A T I O N
NORMAL   MVI   3(7),X'00'           MOVE RETURN CODE TO ACTION WORD
         RETRN (14,12)              PROCESS RECORD
*        D E L E T E   R E C O R D
DELETE   MVI   3(7),X'04'           MOVE RETURN CODE TO ACTION WORD
         RETRN (14,12)              DELETE RECORD
*        M O D I F Y   R E C O R D
MODIFY   MVI   3(7),X'00'           MOVE RETURN CODE TO ACTION WORD
         MVI   0(8),C'X'            MODIFY 1ST POSITION IN RECORD
         RETRN (14,12),             PROCESS RECORD
*        I N S E R T   R E C O R D
INSERT   LA    8,SATZ               ADDRESS OF RECORD TO BE INSERTED
         ST    8,0(1)               MOVE ADDRESS OF RECORD TO INPUT AREA
         OI    SWITCH+1,X'F0'       SET SWITCH TO BRANCH
         MVI   3(7),X'0C'           MOVE RETURN CODE TO ACTION WORD
         RETRN (14,12)              INSERT RECORD
*        E N D
ENDE     MVI   3(7),X'08'           MOVE RETURN CODE TO ACTION WORD
         RETRN (14,12)              END OF INPUT FOR THIS EXIT
RECORD   DS    0CL17
         DC    C'INSERT ONE RECORD'
         END   E15
```

Comment on the source text

The source text is assembled with ASSEMBH.

The assembled program is stored as a LLM in the SORT.EXAMPLE.LIB library and given the name E15.

File attributes of the input file UE.EXIT.INPUT

```
/show-file-attributes file-name=ue.exit.input, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.UE.EXIT.INPUT
%  ------------------------------ ORGANIZATION -----------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (F,N)       REC-SIZE   = 17
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K           SO-MIGR    = *ALLOWED
```

Contents of the input file UE.EXIT.INPUT

	Sort field	
1	5	11 17

```
9999 999999 9999999
2222 222222 2222222
0000 000000 0000000
8888 888888 8888888
5555 555555 5555555
1111 111111 1111111
7777 777777 7777777
8888 888888 8888888
0000 000000 0000000
9999 999999 9999999
```

Trace listing

```
/add-file-link link-name=blslib,file-name=sort.example.lib
/start-sort
%  SRT1001  2014-10-12/13:20:33/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=ue.exit.input,output-file=ue.exit.output
//assign-exits input=*module(name=e15,parameter-mode=*any)
//sort-records fields=*field-explicit(position=5,length=6), -
//                     estimated-records=10
//end
```

```
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................10 (FROM 01)
%  SRT1018  INSERTED INPUT RECORDS:...........................1
%  SRT1019  DELETED INPUT RECORDS:............................2
%  SRT1017  RECORDS TO BE SORTED/MERGED:......................9
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................9
%  SRT1002  2014-10-12/13:26:47/000000.32 SORT/MERGE COMPLETED
```

File attributes of the output file UE.EXIT.OUTPUT

```
/show-file-attributes file-name=ue.exit.output, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.UE.EXIT.OUTPUT
%  ------------------------------ ORGANIZATION -----------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (F,N)       REC-SIZE   = 17
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K           SO-MIGR    = *ALLOWED
```

Contents of the output file UE.EXIT.OUTPUT

```
INSERT ONE RECORD   This record has been inserted

00000000000000000
00000000000000000
11111111111111111
22222222222222222
55555555555555555
77777777777777777

X9999999999999999   This record has been modified
X9999999999999999   This record has been modified
```

10.5.17 Example 17: OUTPUT user exit

Full sort of variable format records

Input:	SAM file E35A.INPUT with variable record format
Output:	SAM file E35A.OUTPUT with variable record format
Library:	SORT.EXAMPLE.LIB
Module:	E35B
User exit:	OUTPUT (PARAMETER-MODE=*ANY)

Exercise:

– Link in a module from a library
– Assign the input and output files using ASSIGN-FILES
– Select the user exit for Assembler output routines

Description of the user routine

In this example, records are inserted, modified and deleted by means of a user routine. The routine is activated for each output record and is meant to perform the following actions:

– Output all records that contain the digit ’3’ at byte position 5.
– Insert a record after each record that contains the digit ’1’ at byte position 5.
– Modify all records that contain the digit ’5’ at byte position 5 by replacing the first character with ’A’.
– Delete all records that contain digits not equal to ’1’, ’3’ or ’5’ at byte position 5.

Source text

```
E35B     START
         PRINT NOGEN
         ENTRY E35B01
         USING E35B01,15
E35B01   SAVE  (14,12)              SAVE REGISTERS
         L     7,12(1)              LOAD ADDRESS OF ACTION WORD
         CLC   0(4,1),=XL4'00'      LAST OUTPUT RECORD?
         BE    EOF                  YES
         L     8,0(1)               LOAD ADDRESS OF RECORD INTO REG 8
         CLI   4(8),C'3'            1ST POSITION=3?
         BE    PROCESS
         CLI   4(8),C'5'            1ST POSITION=5?
         BE    MODIFY
SWITCH   NOP   INSERT
         CLI   4(8),C'1'            1ST POSITION=1?
         BNE   DELETE
         OI    SWITCH+1,X'F0'       SET SWITCH
         B     PROCESS+4
*        D E L E T E   R E C O R D
DELETE   MVI   3(7),X'04'           MOVE RETURN CODE TO ACTION WORD
         B     ORIGIN
*        I N S E R T   R E C O R D
INSERT   MVC   0(4,1),=A(EIN)
         MVI   SWITCH+1,X'00'       RESET SWITCH
         MVI   3(7),X'0C'           MOVE RETURN CODE TO ACTION WORD
         B     ORIGIN
*        M O D I F Y   R E C O R D
MODIFY   MVI   4(8),C'A'            CHANGE 1ST POSITION OF RECORD
*        T R A N S F E R   R E C O R D
PROCESS  MVI   SWITCH+1,X'00'       RESET SWITCH
         MVI   3(7),X'00'           MOVE RETURN CODE TO ACTION WORD
         B     ORIGIN
*        E N D   P R O C E S S I N G
EOF      MVI   3(7),X'08'           NO MORE RETURNS
ORIGIN   RETRN (14,12)              RESTORE REGISTERS
INS      DC    Y(ENDE-INS)          RECORD TO BE INSERTED
         DS    CL2
         DC    C'RECORD INSERTED!!'
ENDE     EQU   *
         END   E35B
```

Comment on the source text

The source text is assembled with ASSEMBH.

The assembled program is stored as a LLM in the SORT.EXAMPLE.LIB library and given the name E35B.

Contents of the input file E35A.INPUT

RL	Sort field	
1	5	11 21

```
111111 11111111111
222222 22222222222
333333 33333333333
444444 44444444444
555555 55555555555
666666 66666666666
111111 11111111111
222222 22222222222
333333 33333333333
444444 44444444444
555555 55555555555
666666 66666666666
111111 11111111111
222222 22222222222
333333 33333333333
444444 44444444444
555555 55555555555
666666 66666666666
```

Trace listing of the sort run

```
/add-file-link link-name=blslib,file-name=sort.example.lib
/start-sort
%  SRT1001  2014-10-12/13:20:33/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=e35a.input,output-file=e35a.output
//assign-exits output=*module(name=e35b01,parameter-mode=*any)
//sort-records fields=*field-explicit(position=5,length=6)
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................18 (FROM 01)
%  SRT1021  SORTED/MERGED RECORDS.............................18
%  SRT1022  INSERTED OUTPUT RECORDS:..........................3
%  SRT1023  DELETED OUTPUT RECORDS:...........................9
%  SRT1030  SORT/MERGE OUTPUT RECORDS:.......................12
%  SRT1002  2014-10-12/13:20:47/000000.32 SORT/MERGE COMPLETED
```

Comment

In this example, the records to be output are processed by an Assembler routine. The Assembler module must therefore be assigned as OUTPUT. The name given to the module matches the module name in the SORT.EXAMPLE.LIB library. The addressing mode and the interface between SORT and the user module are defined with the PARAMETER-MODE operand.

In this example, PARAMETER-MODE=*ANY is correct because the OUTPUT user exit used by the Assembler routine supports both 31- and 24-bit addressing.

For user exits which allow 24-bit addressing only, PARAMETER-MODE=24 is **mandatory**.

10.5.18 Example 18: PHYSICAL-TRANSLATE user exit

Full sort of fixed format records

Input: SAM file PHYSICAL.TRANSLATE.INPUT

Output: SAM file PHYSICAL.TRANSLATE.OUTPUT

Library: SORT.EXAMPLE.LIB

Module: ETBSORT

User Exit: PHYSICAL-TRANSLATE

This example demonstrates how to sort a file in a user-defined order (digits, characters, spaces) which differs from both the EBCDIC and the ASCII standard sequence.

The characters present are translated and sorted in accordance with the first code table defined in the user routine. Before the output takes place, the characters are converted back by means of the second code table.

Source text of the user routine

```
ETBSORT  CSECT
       TITLE 'SPECIAL ALPHABET'
*
* CODE CONVERSION FROM EBCDIC
*
* TO THE FOLLOWING SEQUENCE:
* '0123456789ABCDEFGHI'
* 'JKLMNOPQRSTUVWXYZ'
* CHANGING THE SEQUENCE OF UPPERCASE AND LOWERCASE LETTERS
* IS A SIMILAR PROCEDURE
*
         DC    XL64'00'
BLANK    DC    X'25'                    SPACE AFTER LETTERS
         DC    XL128'00'
ATOI     DC    XL9'0B0C0D0E0F10111213'        LETTERS A - I
         DC    XL7'00'
JTOR     DC    XL9'1415161718191A1B1C'        LETTERS J - R
         DC    XL8'00'
STOZ     DC    XL8'1D1E1F2021222324'          LETTERS S - Z
         DC    XL6'00'
NULL9    DC    XL10'0102030405060708090A'     DIGITS 0 - 9 BEFORE LETTERS
         DC    XL6'00'
*
         DC    X'00'
         DC    '0123456789'       RECONVERT DIGITS 0 - 9.
         DC    'ABCDEFGHI'        RECONVERT LETTERS A - I.
         DC    'JKLMNOPQR'        RECONVERT LETTERS J - R.
         DC    'STUVWXYZ'         RECONVERT LETTERS S - Z .
         DC    X'40'              RECONVERT SPACE
         DC    XL127'00'
*
         END
```

Comment on the source text

The source text is assembled with ASSEMBH.

The assembled program is stored as a LLM in the SORT.EXAMPLE.LIB library and given the name ETBSORT.

Trace listing

```
/add-file-link link-name=blslib,file-name=sort.example.lib
/start-sort
%  SRT1001  2014-10-12/16:14:06/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=physical-translate.input, -
//            output-file=physical-translate.output
//assign-exits physical-translate=*module(name=etbsort)
//sort-records fields=*field-explicit(position=1,length=4, -
//                    format=*physical-translate)
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.............................25 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:............................25
%  SRT1002  2014-10-12/16:14:22/000000.26 SORT/MERGE COMPLETED
```

Comment

Before SORT is started, the library file SORT.EXAMPLE.LIB is assigned. ASSIGN-EXITS is specified so that the previously assembled routine ETBSORT can be activated.

Contents of the input and output files

The file PHYSICAL.TRANSLATE.INPUT has been sorted according to the EBCDIC sequence. The output file PHYSICAL.TRANSLATE.OUTPUT has been sorted using the code table defined by the user. The following file comparison is intended to clarify the difference between these two sort types.

PHYSICAL.TRANSLATE.INPUT

```
 FGH 906C
ABCD E234
ADTX 0914
ASDF 9082
BKLM 0127
BSDF P093
ERSK 7654
ERU  897X
JKDS 7809
KLMD 9808
SFTD OPLD
0012 3POL
1178 LLOP
1234 OPRS
1234 6793
2349 MNB
2356 ADFG
4456 PPLO
4569 87  N
4578 DFGH
7658 9XYZ
77  9 AAPK
7777 RRRS
9875 DGFK
9999 9999
```

PHYSICAL.TRANSLATE.OUTPUT

```
0012 3POL
1178 LLOP
1234 OPRS
1234 6793
2349 MNB
2356 ADFG
4456 PPLO
4569 87  N
4578 DFGH
7658 9XYZ
7777 RRRS
77  9 AAPK
9875 DGFK
9999 9999
ABCD E234
ADTX 0914
ASDF 9082
BKLM 0127
BSDF P093
ERSK 7654
ERU  897X
JKDS 7809
KLMD 9808
SFTD OPLD
 FGH 906C
```

10.5.19 Example 19: VIRTUAL-TRANSLATE user exit

Full record sort based on a user-defined sorting order

Input:	SAM file VIRTRAN.INPUT with variable record format
Output:	SAM file VIRTRAN.OUTPUT with variable record format
Library:	SORT.EXAMPLE.LIB (user-defined)
Module:	VIRTRAN
User exit:	VIRTUAL-TRANSLATE

Exercise:

- Define a special code table with the Assembler
- Write the table into a library
- Assign the library and invoke the module
- Compare the result

Source text

```
VIRTRAN  CSECT
VIRTRAN  AMODE ANY
VIRTRAN  RMODE ANY
         TITLE 'USER-DEFINED SORT SEQUENCE'
*CODE CONVERSION FROM EBCDIC TO
*NEW SEQUENCE
*
*SEQUENCE
*UPPERCASE AND LOWERCASE LETTERS
*ARE TREATED AS EQUIVALENT
         DC    XL64'00'
BLANK    DC    X'00'                  SPACE
         DC    XL64'00'
*
*  LOWERCASE LETTERS
LWATOI   DC    XL9'0B0C0D0E0F10111213'      LOWERCASE LETTERS A-I
         DC    XL7'00'
LWJTOR   DC    XL9'1415161718191A1B1C'      LOWERCASE LETTERS J-R
         DC    XL8'00'
LWSTOZ   DC    XL8'1D1E1F2021222324'        LOWERCASE LETTERS S-Z
         DC    XL23'00'
*
*  UPPERCASE LETTERS UND DIGITS
*
ATOI     DC    XL9'0B0C0D0E0F10111213'      UPPERCASE LETTERS A-I
         DC    XL7'00'
JTOR     DC    XL9'1415161718191A1B1C'      UPPERCASE LETTERS J-R
         DC    XL8'00'
STOZ     DC    XL8'1D1E1F2021222324'        UPPERCASE LETTERS S-Z
         DC    XL6'00'
ZERO    DC    XL10'0102030405060708090A'   DIGITS 0-9
         DC    XL6'00'
*
         END
```

Comment on the source text

The source text is assembled with ASSEMBH.

The assembled program is stored as a LLM in the SORT.EXAMPLE.LIB library and given the name VIRTRAN.

Note concerning the code table

A code table is defined via the user exit VIRTUAL-TRANSLATE (cf. also the PHYSICAL-TRANSLATE user exit).

Trace listing

```
/add-file-link link-name=blslib,file-name=sort.example.lib
/start-sort
%  SRT1001  2014-10-12/17:17:55/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=virtran.input,output-file=virtran.output
//assign-exits virtual-translate=*module(name=virtran)
//sort-records fields=*field-explicit(position=5,length=15, -
//                    format=*virtual-translate),estimated-records=29
//set-record-attributes output=*variable(maximum-reocrd-size=45)
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................29 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................29
%  SRT1002  2014-10-12/17:18:09/000000.37 SORT/MERGE COMPLETED
```

Explanation of the run

– The library containing the VIRTRAN module is assigned

– The input and output files are assigned

– The VIRTRAN module is assigned by means of the ASSIGN-EXITS statement

– A reference to the code table is given in the SORT-RECORDS statement

– An estimate of the number of records is given

– The longest entry is indicated (length 45)

SORT sets up the output file with the record length specified in the SET-RECORDS statement and enters this value in the catalog.

```
/show-file-attributes file-name=virtran.output, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.VIRTRAN.OUTPUT
%  ------------------------------ ORGANIZATION ------------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (V,N)       REC-SIZE   = 45
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K          SO-MIGR    = *ALLOWED
```

VIRTRAN.INPUT	VIRTRAN.OUTPUT
scherzen	Abendland
allgemein	allgemein
mexikanisch	Allgemeinheit
Mehrzahl	Astronomie
oder	astronomisch
Trennung	Biologie
Verhaeltniswort	brasilianisch
Stilkunde	Brasilien
brasilianisch	bretonisch
philosophisch	Deutschland
medizinisch	dichterisch
dichterisch	Druckersprache
Druckersprache	englisch
Astronomie	franzoesisch
mechanisch	Franzosen
scherzhaft	Landwirtschaft
bretonisch	Mechanik
Biologie	mechanisch
Brasilien	medizinisch
Allgemeinheit	Mehrzahl
astronomisch	mexikanisch
Mechanik	oder
Franzosen	philosophisch
englisch	Scherz
Landwirtschaft	scherzen
franzoesisch	scherzhaft
Abendland	Stilkunde
Scherz	Trennung
Deutschland	Verhaeltniswort

10.5.20 Example 20: SORT as a subroutine (level 0)

Input: SAM file RESTAURANT.SAM.FIX with fixed record format

Output: SAM file SORT.UPRG.SORT with fixed record format and output to terminal via SYSOUT

Structure of the records in the input file RESTAURANT.SAM.FIX

Restaurant name	Street	Tel.	Cuisine

The contents of this file are listed in section “Contents of the example files” on page 383.

The main program reads the records from the RESTAURANT.SAM.FIX file, edits them into a new format and outputs them to the file SORT.UPRO. The records in SORT.UPRO are to be sorted (in a full sort) and output simultaneously to the file SORT.UPRG.SORT and to the terminal.

Record structure of the intermediate file SORT.UPRO

Cuisine	Restaurant name	Street	Tel.

The control information for SORT is passed at level 0 (SORT statements from SYSDTA).

Source text

```
SUPROGO  START
         TITLE 'THIS PROGRAM CALLS SORT AS A SUBROUTINE AT LEVEL 0'
         PRINT NOGEN
SUPROGO  AMODE ANY
SUPROGO  RMODE ANY
         GPARMOD 31
BEGIN    BALR  3,0
         USING *,3
         OPEN  EIN,INPUT            OPEN THE INPUT FILE
         OPEN  AUS,OUTPUT           OPEN THE OUTPUT FILE
         MVI   AUSB,X'40'
         MVC   AUSB+1(71),AUSB
READ1    EQU   *
         GET   EIN,EINB             READ ONE RECORD
         MVC   AF1,EF4              RE-EDIT INPUT
         MVC   AF2,EF1              RECORD
         MVC   AF3,EF2
         MVC   AF4,EF3
         PUT   AUS,AUSB             OUTPUT ONE RECORD
         B     READ1
*
*                                   FCB OF INPUT FILE
EIN      FCB   FCBTYPE=SAM,                                          -
               LINK=EIN,                                            -
               RECFORM=F,                                           -
               RECSIZE=66,                                          -
               EXIT=EXITEIN
EXITEIN  EXLST EOFADDR=ENDE1        LAST INPUT RECORD
*                                   FCB OF OUTPUT FILE
AUS      FCB   FCBTYPE=SAM,                                          -
               LINK=AUS,                                            -
               RECFORM=F,                                           -
               RECSIZE=72,                                          -
               EXIT=EXITAUS
EXITAUS  EXLST EOFADDR=ENDE9
         DS    OF
ENDE1    EQU   *
         CLOSE ALL
         FILE  SORT.UPRO,LINK=SORTIN        INPUT FILE (FOR SORT RUN)
         FILE  SORT.UPRG.SORT,LINK=SORTOUT  OUTPUT FILE FOR SORT RUN
         LA    1,B1                   LEVEL 0
         LA    13,SAVE                MOVE SAVE AREA ADDRESS TO REG13
         L     15,=V(SORTU)           SORT ENTRY POINT
         BALR  14,15
```

```
ENDE2    EQU   *                       THE OUTPUT FILE IS GIVEN THE
         FILE  SORT.UPRG.SORT,LINK=AUS SAME ATTRIBUTES AS THE FILE WITH
         OPEN  AUS,INPUT               THE ADDRESS AUS.
READ2    EQU   *
         GET   AUS,AUSB
         WROUT OUT,ERRORS
         B     READ2
ERRORS   CLOSE ALL
         TERM MODE=ABNORMAL,UNIT=STEP
ENDE9    CLOSE ALL
         TERM
*
B1       SRTO STXIT=NO,MSGPROT=BOTH,SDF=YES   CONTROL INFO VIA LEVEL 0
*
SAVE     DS    18F
EINB     DS    0CL72
EF1      DS    CL20
EF2      DS    CL27
EF3      DS    CL08
EF4      DS    CL11
OUT      DC    H'77'
         DC    C'  '
         DC    X'01'
AUSB     DS    0CL72
AF1      DS    CL11
         DS    CL03
AF2      DS    CL20
         DS    CL03
AF3      DS    CL24
         DS    CL03
AF4      DS    CL08
         DS    0F
         END   SUPROG0
```

Comment on the source text

The macro call SRT0 provides values at level 0.

The source text is assembled with ASSEMBH. To make this possible, the $.SYSLIB.SORT.078 library containing the SORT macros must be assigned as a macro library.

The assembled program is stored as a LLM in the SORT.EXAMPLE.LIB library and given the name SUPROG0.

Trace listing

```
/add-file-link link-name=ein,file-name=restaurant.sam.fix
/create-file file-name=sort.upro
/add-file-link link-name=aus,file-name=sort.upro
/add-file-link link-name=blslib00,file-name=$.syslnk.sort.080
/start-executable-program library=sort.example.lib, -
/                         element-or-symbol=suprog0, -
/                         alternate-library=*blslib##
%  SRT1001 B1 2014-10-12/13:57:52/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130 B1 PLEASE ENTER SORT STATEMENTS
//sort-records
//end
% SRT1016 B1 SORT/MERGE INPUT RECORDS:.........................10 (FROM 01)
% SRT1030 B1 SORT/MERGE OUTPUT RECORDS:........................10
% SRT1002 B1 2014-10-12/13:58:08/000000.26 SORT/MERGE COMPLETED
French        Le Gourmet            Lime Street 46            505397
German        Willi's Bar           Westland Street 113       748293
Indonesian    Java                  Hope Street 51            522221
Italian       Orlando's             Thompson Street 62        220061
Japanese      Chayota's             Thurston Street 60        292742
Mexican       Palenque Mexico       Millwood Drive 2          980149
Persian       Persepolis            Salford Square 20         597004
Vegetarian    Strawberry            Sauchiehall Street 8      595521
Vietnamese    Vietnam               Thurston Street 47        522518
Yugoslavian   Golden Fleece         Arran Street 44           242437
```

Comment on the trace listing

– Assign the file RESTAURANT.SAM.FIX as the input file for the main program SUPROG0.
– Assign the file SORT.UPRO as output file for the main program SUPROG0. The records edited by the main program will be output to this file.
– Assign the module library $.SYSLNK.SORT.080, where the SORT modules are stored.
– Call the main program SUPROG0.
– The main program invokes SORT; SORT prompts for statements to be input.
– The SORT-RECORDS statement requests a full sort.
– The END statement concludes the sort run definition and starts the sort run.

10.5.21 Example 21: SORT as a subroutine (level 1)

Input: SAM file RESTAURANT.SAM.FIX with fixed record format

Output: SAM file SORT.UPRG.SORT with fixed record format

Structure of the records in the input file RESTAURANT.SAM.FIX

Restaurant name	Street	Tel.	Cuisine

The contents of this file are listed in section “Contents of the example files” on page 383.

The main program reads the records from the RESTAURANT.SAM.FIX file, edits them into a new format and outputs them to the file SORT.UPRO.

Record structure of the intermediate file SORT.UPRO

Cuisine	Restaurant name	Street	Tel.

The SORT.UPRO records are to be sorted on the sort field “cuisine” (full sort) and output simultaneously to the file SORT.UPRG.SORT and to the terminal. A condition on records to be included in the sort operation is that the sort field “cuisine” begins with the letter “I” or “J”.

The control information is passed to SORT at level 1 (SORT statements in main memory).

Source text

```
SUPROG1  START
         TITLE 'THIS PROGRAM CALLS SORT AS A SUBROUTINE AT LEVEL 1'
         PRINT NOGEN
SUPROG1  AMODE ANY
SUPROG1  RMODE ANY
         GPARMOD 31
BEGIN    BALR  3,0
         USING *,3
         OPEN  EIN,INPUT            OPEN INPUT FILE
         OPEN  AUS,OUTPUT            OUTPUT FILE
         MVI   AUSB,X'40'
         MVC   AUSB+1(71),AUSB
LIES1    EQU   *
         GET   EIN,EINB             READ A RECORD
         MVC   AF1,EF4              EDIT INPUT RECORD
         MVC   AF2,EF1              INTO NEW FORMAT
         MVC   AF3,EF2
         MVC   AF4,EF3
         PUT   AUS,AUSB             OUTPUT A RECORD
         B     LIES1
*                                   FCB OF INPUT FILE
EIN      FCB   FCBTYPE=SAM,                                          -
               LINK=EIN,                                            -
               RECFORM=F,                                           -
               RECSIZE=66,                                          -
               EXIT=EXITEIN
EXITEIN  EXLST EOFADDR=ENDE1        LAST INPUT RECORD
*                                   FCB OF OUTPUT FILE
AUS      FCB   FCBTYPE=SAM,                                          -
               LINK=AUS,                                            -
               RECFORM=F,                                           -
               RECSIZE=72,                                          -
               EXIT=EXITAUS
EXITAUS  EXLST EOFADDR=ENDE9
         DS    OF                   ALIGN ON WORD BOUNDARY
ENDE1    EQU   *
         CLOSE ALL
         FILE  SORT.UPRO,LINK=SORTIN  INPUT FILE FOR SORT RUN
         FILE  SORT.UPRG.SORT,LINK=SORTOUT  OUTPUT FILE FOR SORT RUN
         LA    1,B1                   LEVEL 1
         LA    13,SAVE                MOVE SAVE AREA ADDRESS TO REG 13
         L     15,=V(SORTU)           SORT ENTRY POINT
         BALR  14,15
```

```
ENDE2    EQU   *
         FILE  SORT.UPRG.SORT,LINK=AUS THE OUTPUT FILE IS GIVEN
         OPEN  AUS,INPUT               THE SAME ATTRIBUTES AS THE
*                                      FILE WITH THE ADDRESS 'AUS'.
READ2    EQU   *
         GET   AUS,AUSB
         WROUT OUT,ERRORS
         B     READ2
ERRORS         CLOSE ALL
         TERM MODE=ABNORMAL,UNIT=STEP
ENDE9    CLOSE ALL
         TERM
         PRINT GEN
B1       SRT1  (SORT-RECORDS FIELDS=*FIELD-EXPLICIT(POSITION=1,LENGTH=1-
               1)),SDF=YES
         SRT1  (SELECT-INPUT-RECORDS CONDITION=(((1,1,CH)='I')OR((1,1,C-
               H)='J')))
         SRT1  (SET-SORT-OPTIONS MIN-MSG-WEIGHT=*ALL)
         SRT1  (END)
SAVE     DS    18F
EINB     DS    OCL66
EF1      DS    CL20
EF2      DS    CL27
EF3      DS    CL08
EF4      DS    CL11
OUT      DC    H'77'
         DC    C'  '
         DC    X'01'
AUSB     DS    OCL72
AF1      DS    CL11
         DS    CL03
AF2      DS    CL20
         DS    CL03
AF3      DS    CL24
         DS    CL03
AF4      DS    CL08
         DS    OF
         END   SUPROG1
```

Comment on the source text

The SRT1 macro sends the following control statements to SORT (values supplied at level 1):

– SORT-RECORDS to request a full sort
– SELECT-INPUT-RECORDS to exclude all input records which do not begin with I or J
– SET-SORT-OPTIONS to control the message output
– END statement to terminate the statement sequence.

The source text is assembled with ASSEMBH. To make this possible, the $.SYSLIB.SORT.080 library containing the SORT macros must be assigned as a macro library.

The assembled program is stored as an object module in the SORT.EXAMPLE.LIB library and given the name SUPROG1.

Syntax of the SRT1 macro

The SDF statement text of the SRT1 macro must be enclosed in parentheses.
The first SRT1 macro must be terminated by the entry SDF=YES. This is placed after the closing parenthesis.
All SDF statements of the macro must be written sequentially, without spaces (blanks) as fillers, up to the continuation character in column 72. This may produce divisions that do not conform to grammatical rules. This rule also applies to the SELECT-INPUT-RECORDS statement used in this example.

Trace listing

```
/add-file-link link-name=ein,file-name=restaurant.sam.fix
/create-file file-name=sort.upro
/add-file-link link-name=aus,file-name=sort.upro
/add-file-link link-name=blslib00,file-name=$.syslnk.sort.080
/start-executable-program library=sort.example.lib, -
/                         element-or-symbol=suprog1, -
/                         alternate-library=*blslib##
%  SRT1001 B1 2014-10-12/12:56:17/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1046 B1 2001-12-10/12:56:19/000000.26 END OF PREPARATORY PHASE
%  SRT1013 B1 MAIN MEMORY SORT
%  SRT1016 B1 SORT/MERGE INPUT RECORDS:.........................10 (FROM 01)
%  SRT1024 B1 DELETED SELECT-INPUT-RECORDS RECORDS:.....................7
%  SRT1017 B1 RECORDS TO BE SORTED/MERGED:.......................3
%  SRT1030 B1 SORT/MERGE OUTPUT RECORDS:.........................3
%  SRT1002 B1 2014-10-12/12:56:20/000000.28 SORT/MERGE COMPLETED
Indonesian    Java                 Hope Street 51            522221
Italian       Orlando's            Thompson Street 62        220061
Japanese      Chayota's            Thurston Street 60        292742
```

Comment on the trace listing

– Assign the file RESTAURANT.SAM.FIX as the input file for the main program SUPROG0.

– Assign the file SORT.UPRO as output file for the main program SUPROG0. The records edited by the main program will be output to this file.

– Assign the module library $.SYSLNK.SORT.080, where the SORT modules are stored.

– Call the main program SUPROG1.

– The main program invokes SORT; SORT prompts for statements to be input.

File attributes of the input file RESTAURANT.SAM.FIX

```
/show-file-attributes file-name=restaurant.sam.fix, -
/                     information=*parameters(organization=*yes)
%0000000003:CTID:$EXAMPLE.RESTAURANT.SAM.FIX
%  ------------------------------ ORGANIZATION ------------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (F,N)       REC-SIZE   = 66
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K          SO-MIGR    = *ALLOWED
```

File attributes of the output file SORT.UPRG.SORT

```
/show-file-attributes file-name=sort.uprg.sort, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.SORT.UPRG.SORT
%  ------------------------------ ORGANIZATION ------------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (F,N)       REC-SIZE   = 72
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K          SO-MIGR    = *ALLOWED
```

10.5.22 Example 22: SORT access method

Input: SAM file RESTAURANT.SAM with variable record format

Output: SAM file RESTAURANT.SAM.AUS with variable record format

Source text: SORTZM01

Exercise:

– Assign an input file and an output file with the file link names EIN and AUS instead of SORTIN and SORTOUT

– Use of the SORTZM macro

This example shows how to perform sorting using the SORT access method SORTZM.

The input records are provided in a SAM file RESTAURANT.SAM, which has variable record format and is assigned the file link name EIN.

The records of the input file RESTAURANT.SAM are structured as follows:

RL	Restaurant name	Street	Tel.	Cuisine

The output records are to be written by SORT into the output file RESTAURANT.SAM.AUS, which also has variable record format and is assigned the file link name AUS. The following fields are selected:

RL	Cuisine	Restaurant name	Street

The records in RESTAURANT.SAM.AUS are to be sorted on the sort field “cuisine”. The other fields are remainder fields.

Source text

```
SORTZM01 START
         PRINT NOGEN
SORTZM01 AMODE ANY
SORTZM01 RMODE ANY
         GPARMOD 31
         BALR  10,0
         USING ANFANG,10,11
ANFANG   L     11,BASADR
         B     BEGINN
BASADR   DC    A(ANFANG+4096)
*
BEGINN   EQU   *
         OPEN  EIN,INPUT               OPEN INPUT FILE
         OPEN  AUS,OUTPUT              OPEN OUTPUT FILE
*
*        ***************************
         SRTOPEN SCB=B1,ERROR=FEHLER   OPEN SORT RUN
*        ***************************
*
LIES     GET   EIN,EINBER              READ RECORD
*
*        ***************************
         SRTPUT SCB=B1,RECORD=EINBER   PASS RECORD TO SORT RUN
*        ***************************
*
         B     LIES
ENDEIN   LA    4,AUSBER
*
*        *********************************
SCHREIB  SRTGET SCB=B1,RECORD=(4),EOS=CLOSE   FETCH RECORD
*        *********************************
*
         PUT   AUS,AUSBER              OUTPUT RECORD
         B     SCHREIB
*
*        **************
CLOSE    SRTCLSE SCB=B1                CLOSE SORT RUN
*        **************
*
CLALL    CLOSE ALL                     CLOSE INPUT/OUTPUT FILES
         TERM
*
FEHLER   CLOSE ALL                     ERROR EXIT FOR
         TERM  MODE=ABNORMAL,UNIT=STEP ABNORMAL TERMINATION
*
EIN      FCB   FCBTYPE=SAM,LINK=EIN,RECFORM=V,EXIT=EXITEIN
```

```
AUS      FCB   FCBTYPE=SAM,LINK=AUS,RECFORM=V,EXIT=EXITAUS
*
EXITEIN  EXLST EOFADDR=ENDEIN,COMMON=CLALL
EXITAUS  EXLST COMMON=CLALL
*
EINBER   DS    0CL76
SL1      DS    CL4
DATEN1   DS    CL72
*
AUSBER   DS    0CL68
SL2      DS    CL4
DATEN2   DS    CL64
*
B1       SRT1  (SORT-RECORDS FIELDS=(*FIELD-EXPLICIT(POSITION=60,LENGTH-
               =17),REMAINDER-EXPLICIT(POSITION=5,LENGTH=20),*REMAINDER-
               -EXPLICIT(POSITION=25,LENGTH=27)),SORT-TYPE=*COMPOUND-RE-
               CORD),SDF=YES
         SRT1  (SET-RECORD-ATTRIBUTES OUTPUT=*VARIABLE)
         SRT1  (SET-SORT-OPTIONS MIN-MSG-WEIGHT=*ALL)
         SRT1  (END)
         END   SORTZM01
```

Comment on the source text

The following actions are performed in the source text:

– Open the input file EIN.

– Open the output file AUS.

– The SRTOPEN macro initiates the SORT run.

– Read the input record from the input file EIN.

– The SRTPUT macro passes an input record to the SORT run.

– The next input record is read. The read loop is repeated until the last input record is read.

– End of the input (EOFADDR in the EXLST macro).

– The SRTGET macro fetches the sorted records from the sort run. When all the records have been transferred, a branch is made to close the sort run (EOS operand).

– Output the records of the sort run to the output file AUS.

– Following output of the records, the sort run is terminated with the SRTCLSE macro.

– Close the input and output files.

– SRT1 macros for the sort run (selection sort).

Notes on the syntax of the SRT1 macro

The SDF statement text of the SRT1 macro must be enclosed in parentheses.
The first SRT1 macro must be terminated by the entry SDF=YES. This is placed after the closing parenthesis.
All SDF statements of the macro must be written sequentially, without spaces (blanks) as fillers, up to the continuation character in column 72. This may produce divisions that do not conform to grammatical rules.

The source text is assembled with ASSEMBH. To make this possible, the $.SYSLIB.SORT.078 library containing the SORT macros must be assigned as a macro library.

The assembled program is stored as a LLM in the SORT.EXAMPLE.LIB library and given the name SORTZM01.

Trace listing

```
/add-file-link link-name=blslib00,file-name=$.syslnk.sort.080
/add-file-link link-name=ein,file-name=restaurant.sam
/create-file file-name=sort.upro
/add-file-link link-name=aus,file-name=sort.upro
/start-executable-program library=sort.example.lib, -
/                         element-or-symbol=sortzm01, -
/                         alternate-libraries=*blslib##
%  SRT1001 B1 2014-10-12/12:56:17/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1046 B1 12:56:19/000000.26 END OF PREPARATORY PHASE
%  SRT1013 B1 MAIN MEMORY SORT
%  SRT1021 B1 SORTED/MERGED RECORDS:.............................10
%  SRT1030 B1 SORT/MERGE OUTPUT RECORDS:.........................10
%  SRT1002 B1 12:56:20/000000.28 SORT/MERGE COMPLETED
```

Explanation of the run:

– Assign the module library $.SYSLNK.SORT.080 in which the SORT modules are stored.

– Assign the file RESTAURANT.SAM as the input file for the main program SORTZM01.

– Assign the file SORT.RESTAURANT.AUS as the output file for the sort run.

– Call the SORTZM01 program which calls SORT as a subroutine.

– SORT reports the number of sorted records and confirms normal termination of the sort run.

Contents of the output file RESTAURANT.SAM.AUS

```
French      Le Gourmet          Lime Street 46
German      Willi's Bar         Westland Street 113
Indonesian  Java                Hope Street 51
Italian     Orlando's           Thompson Street 62
Japanese    Chayota's           Thurston Street 60
Mexican     Palenque Mexico     Millwood Drive 2
Persian     Persepolis          Salford Square 20
Vegetarian  Strawberry          Sauchiehall Street 8
Vietnamese  Vietnam             Thurston Street 47
Yugoslavian Golden Fleece       Arran Street 44
```

10.5.23 Example 23: SORT access method (multiple sort)

Input:	SAM file RESTAURANT.SAM.FIX with fixed record format
Output:	SAM files SORT.AUS1, SORT.AUS2 and SORT.AUS3 with fixed record format
Source text:	SORTZM02
Libraries:	$.SYSLIB.SORT.080 macro library for the assembly $.SYSLNK.SORT.080 as TASKLIB

Exercise:

– Assign an input file and multiple output files with the file link names EIN and AUS1, AUS2 and AUS3 instead of SORTIN and SORTOUT.

– Define different sort criteria for the three output files

– Call SORT as a subroutine

This example shows how to perform a *multiple sort* using the SORT access method SORTZM.
The input records are made available in a SAM file RESTAURANT.SAM.FIX, which has fixed record format and is assigned with the file link name EIN.

The records of the RESTAURANT.SAM.FIX input file are structured as follows:

Restaurant name	Street	Tel.	Cuisine

The input records are read one at a time and passed to three concurrently executing sort runs.

First sort run (full sort)

Output file SORT.AUS1 with fixed record format, which is assigned with the file link name AUS1.

Restaurant name	Street	Tel.	Cuisine

The records in SORT.AUS1 are to be sorted on the sort field “restaurant name”.

Second sort run (selection sort)

Output file SORT.AUS2 with fixed record format, which is assigned with the file link name AUS2. The following fields are selected:

Cuisine	Restaurant name	Street

The records in SORT.AUS2 are to be sorted on the sort field “cuisine”. The other fields are remainder fields.

Third sort run (selection sort)

Output file SORT.AUS3 with fixed record format, which is assigned with the file link name AUS3. The following fields are selected:

Restaurant name	Cuisine	Street

The records in SORT.AUS3 are to be sorted on sort field “street”. The other fields are remainder fields.

Source text

```
SORTZM02 START
         PRINT NOGEN
SORTZM02 AMODE ANY
SORTZM02 RMODE ANY
         GPARMOD 31
         BALR  10,0
         USING ANFANG,10,11
ANFANG   L     11,BASADR
         B     BEGINN
BASADR   DC    A(ANFANG+4096)
*
BEGINN   EQU   *
         OPEN  EIN,INPUT               OPEN INPUT FILE
         OPEN  AUS1,OUTPUT             OPEN OUTPUT FILE1
         OPEN  AUS2,OUTPUT             OPEN OUTPUT FILE2
         OPEN  AUS3,OUTPUT             OPEN OUTPUT FILE3
*
*       ******************************
         SRTOPEN SCB=S01,ERROR=FEHLER  OPEN SORT RUNS
         SRTOPEN S02,FEHLER
         SRTOPEN S03,FEHLER
*       ******************************
*
LIES     GET   EIN,EINBER              READ RECORD
*
*       ******************************
         SRTPUT SCB=S01,RECORD=EINBER     TRANSFER
         SRTPUT S02,EINBER                RECORD TO
         SRTPUT S03,EINBER                SORT RUNS
*       ******************************
*
         B     LIES
```

```
*
ENDEIN   LA    4,AUSBER
*
*       *********************************
L1       SRTGET SCB=SO1,RECORD=(4),EOS=L2
*       *********************************
*
         PUT   AUS1,AUSBER             OUTPUT FROM 1ST SORT RUN
         B     L1
*
*       **********************
L2       SRTGET   SO2,(4),L3
*       **********************
*
         PUT   AUS2,AUSBER             OUTPUT FROM 2ND SORT RUN
         B     L2
*
*       **********************
L3       SRTGET SO3,,CLOSE
*       **********************
*
         LR    0,1
         PUT   AUS3,(0)                OUTPUT FROM 3RD SORT RUN
         B     L3
*
*       ******************
CLOSE    SRTCLSE SCB=SO1               CLOSE SORT
         SRTCLSE SO2                   RUNS
         SRTCLSE SO3
*       ******************
*
CLALL    CLOSE ALL                     CLOSE I/O FILES
         TERM
*
FEHLER   CLOSE ALL
         TERM  MODE=ABNORMAL,UNIT=STEP
*
EIN      FCB   FCBTYPE=SAM,LINK=EIN,RECFORM=F,RECSIZE=66,EXIT=EXITEIN
AUS1     FCB   FCBTYPE=SAM,LINK=AUS1,RECFORM=F,RECSIZE=66,EXIT=EXITAUS
AUS2     FCB   FCBTYPE=SAM,LINK=AUS2,RECFORM=F,RECSIZE=58,EXIT=EXITAUS
AUS3     FCB   FCBTYPE=SAM,LINK=AUS3,RECFORM=F,RECSIZE=58,EXIT=EXITAUS
*
EXITEIN  EXLST EOFADDR=ENDEIN,COMMON=CLALL
EXITAUS  EXLST COMMON=CLALL
*
EINBER   DS    CL72
AUSBER   DS    CL72
*
```

```
S01      SRT1  (SORT-RECORDS FIELDS=*FIELD-EXPLICIT(POSITION=1,LENGTH=2-
               0)),SDF=YES
         SRT1  (SET-RECORD-ATTRIBUTES OUTPUT=*FIXED(RECORD-SIZE=66))
         SRT1  (SET-SORT-OPTIONS MIN-MSG-WEIGHT=*ALL,LINK-PREFIX-CHANGE-
               =S01)
         SRT1  (END)
*
S02      SRT1  (SORT-RECORDS FIELDS=(FIELD-EXPLICIT(POSITION=56,LENGTH=-
               11),REMAINDER-EXPLICIT(POSITION=1,LENGTH=20),REMAINDER-E-
               XPLICIT(POSITION=21,LENGTH=27)),SORT-TYPE=*COMPOUND-RECO-
               RD),SDF=YES
         SRT1  (SET-RECORD-ATTRIBUTES OUTPUT=*FIXED(RECORD-SIZE=58))
         SRT1  (SET-SORT-OPTIONS MIN-MSG-WEIGHT=*ALL,LINK-PREFIX-CHANGE-
               =S02)
         SRT1  (END)
*
S03      SRT1  (SORT-RECORDS FIELDS=(*REMAINDER-EXPLICIT(POSITION=1,LEN-
               GTH=20),*REMAINDER-EXPLICIT(POSITION=56,LENGTH=11),*FIEL-
               D-EXPLICIT(POSITION=21,LENGTH=27)),SORT-TYPE=*COMPOUND-R-
               ECORD),SDF=YES
         SRT1  (SET-RECORD-ATTRIBUTES OUTPUT=*FIXED(RECORD-SIZE=58))
         SRT1  (SET-SORT-OPTIONS MIN-MSG-WEIGHT=*ALL,LINK-PREFIX-CHANGE-
               =S03)
         SRT1  (END)
*
         END   SORTZM02
```

Comment on the source text

Different sort conditions are defined for each of the three output files.

Notes on the syntax of the SRT1 macro

The SDF statement text of the SRT1 macro must be enclosed in parentheses.
The first SRT1 macro must be terminated by the entry SDF=YES. This is placed after the closing parenthesis.
All SDF statements of the macro must be written sequentially, without spaces (blanks) as fillers, up to the continuation character in column 72. This may produce divisions that do not conform to grammatical rules.

The following steps are coded into the source text:

– Open the input file EIN.

– Open the output files AUS1, AUS2 and AUS3.

– Initiate all three SORT runs with the SRTOPEN macro.

– Read an input record from the input file EIN.

– Pass an input record to all three SORT runs with the SRTPUT macro.

– The next input record is read. The read loop is repeated until the last input record is read.

– End of the input (EOFADDR in the EXLST macro).

– The SRTGET macro fetches the sorted records from the first sort run. Once all the records have been transferred, a branch is made to the second sort run (EOS operand).

– Output the records from the first sort run to the output file AUS1.

– The SRTGET macro fetches the sorted records from the second sort run. Once all the records have been transferred, a branch is made to the third sort run (EOS operand).

– Output the records from the second sort run to the output file AUS2.

– The SRTGET macro fetches the sorted records from the third sort run.

– Output the records from the third sort run to the output file AUS3.

– Following output of the records from the third sort run, all three sort runs are terminated by means of the SRTCLSE macro.

– Close the input and output files.

– S01: SRT1 macro for the first sort run (full sort).

– S02: SRT1 macro for the second sort run (selection sort).

– S03: SRT1 macro for the third sort run (selection sort).

The source text is assembled with ASSEMBH. To make this possible, the $.SYSLIB.SORT.080 library containing the SORT macros must be assigned as a macro library.

The assembled program is stored as an object module in the SORT.EXAMPLE.LIB library and given the name SORTZM02.

Trace listing

```
/add-file-link link-name=blslib00,file-name=$.syslnk.sort.080
/add-file-link link-name=ein,file-name=restaurant.sam.fix
/create-file file-name=sort.aus1
/add-file-link link-name=aus1,file-name=sort.aus1
/create-file file-name=sort.aus2
/add-file-link link-name=aus2,file-name=sort.aus2
/create-file file-name=sort.aus3
/add-file-link link-name=aus3,file-name=sort.aus3
/start-executable-program library=sort.example.lib, -
/                         element-or-symbol=sortzm02, -
/                         alternate-library=*blslib##
%  SRT1001 S01 2014-10-12/15:39:23/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1046 S01 2014-10-12/15:39:24/000000.25 END OF PREPARATORY PHASE
%  SRT1001 S02 2014-10-12/15:39:24/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
% SRT1046 S02 2014-10-12/15:39:24/000000.05 END OF PREPARATORY PHASE
% SRT1001 S03 2014-10-12/15:39:24/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1046 S03 2014-10-12/15:39:24/000000.05 END OF PREPARATORY PHASE
%  SRT1013 S01 MAIN MEMORY SORT
%  SRT1013 S02 MAIN MEMORY SORT
%  SRT1013 S03 MAIN MEMORY SORT
%  SRT1021 S01 SORTED/MERGED RECORDS:................................10
%  SRT1002 S01 2014-10-12/15:39:24/000000.38 SORT/MERGE COMPLETED
%  SRT1021 S02 SORTED/MERGED RECORDS:................................10
%  SRT1002 S02 2014-10-12/15:39:24/000000.13 SORT/MERGE COMPLETED
%  SRT1021 S03 SORTED/MERGED RECORDS:................................10
%  SRT1002 S03 2014-10-12/15:39:24/000000.08 SORT/MERGE COMPLETED
```

Comment on the trace listing

For this example, five files have to be assigned before the start of the program:

– $.SYSLNK.SORT.080 as the module library in which the SORT modules are stored.

– RESTAURANT.SAM.FIX as the input file for the main program SORTZM02.

– SORT.AUS1 as the output file for the first sort run.

– SORT.AUS2 as the output file for the second sort run.

– SORT.AUS3 as the output file for the third sort run.

– The SORTZM02 module is started.

– SORT is called via the access method SORTZM and outputs a start message for each sort run.

– On termination of each sort run, SORT reports the number of records sorted and the normal termination of the sort run.

File attributes of the input file RESTAURANT.SAM.FIX

```
/show-file-attributes file-name=restaurant.sam.fix, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.RESTAURANT.SAM.FIX
%  ------------------------------- ORGANIZATION -----------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (F,N)       REC-SIZE   = 66
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K           SO-MIGR    = *ALLOWED
```

File attributes of the output files SORT.AUS1, SORT.AUS2 and SORT.AUS3

```
/show-file-attributes file-name=sort.aus*, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.SORT.AUS1
%  ------------------------------- ORGANIZATION -----------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (F,N)       REC-SIZE   = 66
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K           SO-MIGR    = *ALLOWED
%0000000003 :CTID:$EXAMPLE.SORT.AUS2
%  ------------------------------- ORGANIZATION -----------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (F,N)       REC-SIZE   = 58%  AVAIL      = *STD
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K           SO-MIGR    = *ALLOWED
%0000000003 :CTID:$EXAMPLE.SORT.AUS3
%  ------------------------------- ORGANIZATION -----------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (F,N)       REC-SIZE   = 58
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K           SO-MIGR    = *ALLOWED
```

Contents of the output file SORT.AUS1

```
Chayota's           Thurston Street 60        292742  Japanese
Golden Fleece       Arran Street 44           242437  Yugoslavian
Java                Hope Street 51            522221  Indonesian
Le Gourmet          Lime Street 46            505397  French
Orlando's           Thompson Street 62        220061  Italian
Palenque Mexico     Millwood Drive 2          980149  Mexican
Persepolis          Salford Square 20         597004  Persian
Strawberry          Sauchiehall Street 8      595521  Vegetarian
Vietnam             Thurston Street 47        522518  Vietnamese
Willi's Bar         Westland Street 113       748293  German
```

Contents of the output file SORT.AUS2

```
French     Le Gourmet          Lime Street 46
German     Willi's Bar         Westland Street 113
Indonesian Java                Hope Street 51
Italian    Orlando's           Thompson Street 62
Japanese   Chayota's           Thurston Street 60
Mexican    Palenque Mexico     Millwood Drive 2
Persian    Persepolis          Salford Square 20
Vegetarian Strawberry          Sauchiehall Street 8
Vietnamese Vietnam             Thurston Street 47
YugoslavianGolden Fleece       Arran Street 44
```

Contents of the output file SORT.AUS3

```
Golden Fleece       YugoslavianArran Street 44
Java                Indonesian Hope Street 51
Le Gourmet          French     Lime Street 46
Palenque Mexico     Mexican    Millwood Drive 2
Persepolis          Persian    Salford Square 20
Strawberry          Vegetarian Sauchiehall Street 8
Orlando's           Italian    Thompson Street 62
Vietnam             Vietnamese Thurston Street 47
Chayota's           Japanese   Thurston Street 60
Willi's Bar         German     Westland Street 113
```

10.5.24 Example 24: Full sort according to data in Unicode

Input: SAM file NAME with data in Unicode

Output: SAM file NAME.SORT with the sorted records

Exercise:

– Sort according to Unicode (UTF-16)

Record structure of the input file NAME

Family name	First name
Müller (004D 00FC 006C 006C 0065 0072)	Paul (0050 0061 0075 006C)
Ahberg (0041 0068 0062 0065 0072 0067)	Tom (0054 006F 006D)
Werner (0057 0065 0072 006E 0065 0072)	Tom (0054 006F 006D)
Mueller (004D 0075 0065 006C 006C 0065 0072)	Peter (0050 0065 0074 0065 0072)
Åhberg (00C5 0068 0062 0065 0072 0067)	Mike (004D 0069 006B 0065)
Werner (0057 0065 0072 006E 0065 0072)	Tim (0054 0069 006D)
Werners(0057 0065 0072 006E 0065 0072 0073)	Max (004D 0061 0078)
werner (0077 0065 0072 006E 0065 0072)	Moritz (004D 006F 0072 0069 0074 007A)

5 45 84

File attributes of the file name

```
/show-file-attributes file-name=NAME, -
/                     information=*parameters(organization=*yes)
%0000000003 :CTID:$EXAMPLE.NAME
%  ----------------------------- ORGANIZATION ---------------------------
%  FILE-STRUC = SAM         BUF-LEN    = STD(1)      BLK-CONTR  = PAMKEY
%  IO(USAGE)  = READ-WRITE  IO(PERF)   = STD         DISK-WRITE = IMMEDIATE
%  REC-FORM   = (V,N)       REC-SIZE   = 0
%  AVAIL      = *STD
%  WORK-FILE  = *NO         F-PREFORM  = *K           SO-MIGR    = *ALLOWED
/start-sort
%  SRT1001  2014-12-07/12:03:49/000000.00 SORT/MERGE STARTED,
   VERSION 08.0A00/BS2000V18.0
%  SRT1130  PLEASE ENTER SORT STATEMENTS
//assign-files input-files=NAME,output-file=NAME.SORT
//sort-records fields=(*field-explicit(position=5,length=40, -
//                     format=*unicode-character),-
//                     *field-explicit(position=45,length=40,-
//                     format=*unicode-character))
//end
%  SRT1016  SORT/MERGE INPUT RECORDS:.........................8 (FROM 01)
%  SRT1030  SORT/MERGE OUTPUT RECORDS:........................8
%  SRT1002 2014-12-07/12:10:40/000000.14 SORT/MERGE COMPLETED
```

Content of the output file NAME.SORT (sorted)

Line	Family name	First name
1	Ahberg (0041 0068 0062 0065 0072 0067	Tom (0054 006F 006D)
2	Åhberg (00C5 0068 0062 0065 0072 0067)	Mike (004D 0069 006B 0065)
3	Mueller (004D 0075 0065 006C 006C 0065 0072)	Peter (0050 0065 0074 0065 0072)
4	Müller (004D 00FC 006C 006C 0065 0072)	Paul (0050 0061 0075 006C)
5	werner (0077 0065 0072 006E 0065 0072)	Moritz (004D 006F 0072 0069 0074 007A)
6	Werner (0057 0065 0072 006E 0065 0072)	Tim (0054 0069 006D)
7	Werner (0057 0065 0072 006E 0065 0072)	Tom (0054 006F 006D)
8	Werners (0057 0065 0072 006E 0065 0072 0073)	Max (004D 0061 0078)
	5	45 84

Explanation of the output file

Lines 1-2 At level 1 Ahberg and Åhberg are the same.
The sort sequence is defined at level 2: Å > A

Lines 3-4 With Müller the ü is not treated like ue, but like a u with adiactritic.
The sort sequence is defined at level 1: ü > ue

Lines 5-6 At level 1 werner in lowercase is identical to Werner in uppercase.
The sort sequence is defined at level 3: W > w

Lines 6-7 With Werner there is no difference in the first sort field.
The sort sequence is defined by the next sort field: Tom > Tim

Lines 7-8 The sort sequence is defined at level 1 : Werners > Werner

10.6 Contents of the example files

10.6.1 Preliminary remark

In the examples, various files are used for demonstration purposes. The contents of the unsorted files are the same when the first part of the file name is identical. The characteristics of the files do, however, vary between different examples, since ultimately it is the access method which determines the position of the data. This is why the file type is included in the file name.
Thus, a file with a first name part, RESTAURANT, can be used as follows:

Characteristics	File name
SAM file with fixed record length	RESTAURANT.SAM.FIX
SAM file with variable record length	RESTAURANT.SAM
ISAM file with variable record length	RESTAURANT.ISAM.

Thus, variable files merely indicate the access method in the file name, while files having fixed-length record format are identifiable by the suffix FIX.

Similarly, output files are identifiable by the suffix SORT. RESTAURANT.ISAM.SORT, for instance, would be the sorted file corresponding to the file RESTAURANT.ISAM.

Listings of the contents of the unsorted input files are given below. At the end of each listing is a row of numbers, provided to help determine the individual positions more easily.

It should also be pointed out that with SAM files having variable-length record format, a 4-byte record length field needs to be added to the character position. In the case of ISAM files this applies analogously to the ISAM key.

10.6.2 Contents of the file RESTAURANT

Orlando's	Thompson Street 62	220061	Italian
Java	Hope Street 51	522221	Indonesian
Golden Fleece	Arran Street 44	242437	Yugoslavian
Le Gourmet	Lime Street 46	505397	French
Palenque Mexico	Millwood Drive 2	980149	Mexican
Strawberry	Sauchiehall Street 8	595521	Vegetarian
Persepolis	Salford Square 20	597004	Persian
Vietnam	Thurston Street 47	522518	Vietnamese
Chayota's	Thurston Street 60	292742	Japanese
Willi's Bar	Westland Street 113	748293	German
1	21	48	56 SAM fixed 66
5	25	52	60 SAM variable 70
13	33	60	68 ISAM variable 78

10.6.3 Contents of the file LITERATURE

Pasternak	Boris	Doctor Zhivago	Novel
Capote	Truman	In Cold Blood	Novel
Boyle	Jimmy	A Sense of Freedom	Autobiography
Arden	John	Sergeant Musgrave's Dance	Theatre
Milligan	Spike	Puckoon	Novel
Dahl	Roald	Kiss Kiss	Short Stories
Shakespeare	William	Romeo and Juliet	Theatre
Fielding	Henry	Tom Jones	Novel
Jonson	Ben	Volpone	Theatre
Dumas	Alexandre	The Three Musketeers	Novel
Troyat	Henri	Pushkin	Biography
Shaw	Bernard	Pygmalion	Theatre
Sharpe	Tom	Riotous Assembly	Novel
Thomas	Dylan	Fern Hill	Poem
Gogol	Nikolai	Dead Souls	Novel
1	15	26	59 SAM fixed 71
5	19	30	63 SAM variable 75
13	27	38	71 ISAM variable 83

10.6.4 Contents of the file MUSEUM

Ad Lib	Bryant Street 21	331452	Dixieland, trad. jazz
British China Museum	Majestic Square 2	221315	China, porcelain
British Opera House	Prince Albert Square 1	222591	Opera, classical concerts
British Theatre	Majestic Square 2	2185413	Classical theater
Folk Company	St. Vincent Street 15	241977	Traditional folk theater
Gallery Dubois	Sanders Street 59	333510	Contemporary artists
Gallery Dumont	Milton Drive 29	298841	Latin American artists
Gallery Franke	Milton Drive 22	226420	Beckmann, Dix collection
Gallery Schubert	Prince Albert Square 6	475859	Modern art
Gallery Thomas	Milton Drive 60	295517	Contemporary sculpture
Kensington Club	Milton Drive 47	221859	Various art exhibitions
King's Theatre	Gallery Street 4	221152	Experimental theater
Modern Theatre	High Street 3	3445154	Fringe theater
Municipal Museum	St James's Square 1	2332254	History of the city
New Art Gallery	Prince Albert Square 7	5591307	Impressionists
New City Museum	Prince Albert Square 7	5591307	Beuys, Picasso
Scottish Art House	Prince Albert Square 9	224407	Scottish artists
Subway	Soho Lane 23	399482	Modern jazz
Susi's Hideaway	Pratt Street 10	2342660	Nightclub, cabaret
Symposium	George Street 47	363546	Poetry readings
Theatre Royal	Marshall Square 11	225754	Repertory theatre
Vic's	Islington Road 15	223266	Stand-up comics
Wings	Green Street 3	795088	Pop music, disco

1	22	45	53 SAM fixed	77
5	26	49	57 SAM variable	81
13	34	57	65 ISAM variable	89

10.6.5 Contents of the file CULTURE.1

Aquitaine	Acacia Avenue 39	284028	French
August Gardens	Newton Street 16	2604106	Argentinian
Bosna	Freeling Street 11	64115447	Yugoslavian
Bouillabaisse	Falcon Street 10	297909	French
Datscha	King Street 3	341218	Russian
Don Quixote	Billington Street 6	342318	Spanish
Frank's Grill	Forest Road 14	281235	Yugoslavian
Mandarin	Lime Street 21	226888	Chinese
Mifune	Iffman Street 138	987572	Japanese
Nitaya	Thompson Street 19	197772	Thai
Saint George's	Upman Street 67	363666	English
Siracusa	Polson Street 33	770613	Italian
Slavonia	Allcock Street 16	564906	Yugoslavian
Spiros	Upman Street 65	366883	Greek
Zung-Hua	Bond Street 33	555320	Chinese
1	22	44	54 SAM fixed 65
5	26	48	58 SAM variable 69
13	34	56	66 ISAM variable 77

10.6.6 Contents of the file CULTURE.2

1	22	44	54 SAM fixed 65
5	26	48	58 SAM variable 69
13	34	56	66 ISAM variable 77

10.6.7 Contents of the file CULTURE.3

Alcazar's	Doubleway Drive 39	8111590	Argentinian
Buenos Aires	Zoo Road 22	779646	Argentinian
Canton	Theresa Street 49	522185	Chinese
China House	May Street 20	531620	Chinese
El Cid	Belgrade Street 45	3003268	Spanish
Golden Bough	Oswald Street 44	242437	Hungarian
La Barca	Blytheswood Square 1	77613	Italian
Opatija	Robertson Street 2	268353	Yugoslavian
Scorpio	Leslie Street 35	399897	Greek
Tivoli	Wilberforce Drive 52	221274	Italian
Torino	Gardener Lane 8	469571	Italian
Watermill	Leslie Street 33	348000	Swiss
Why Not?	Wimbledon Drive 11	399936	French
Zigzag	Taylor Drive 72	226750	Argentinian
1	22	44	54 SAM fixed 65
5	26	48	58 SAM variable 69
13	34	56	66 ISAM variable 77

11 Messages of the sort/merge program

All messages issued by the sort/merge program are output on SYSOUT. In addition, guranteed messages (warranty messages) can be output in S variables.

11.1 Message output to SYSOUT

Normally, SORT messages are presented in a message format identified by the prefix SRT1001 through SRT13xx. If SORT is called as a subroutine or via the special access method SORTZM, with control information being supplied via the SRT0 or SRT1 macro, each message number is followed by the prefix specified in the first SORT macro call. The message format is then “SRT1xxx prefix”.

The messages include information on the action to be taken and on the status of the sort/merge program after these messages appear.
Each message is assigned a priority; this is indicated in the “Meaning” part of the message description, e.g. priority 2 for message SRT1161. The scope of message output by SORT can be controlled via these priorities by means of the MIN-MSG-WEIGHT operand of the SET-SORT-OPTIONS statement. This causes a message to be output only if the priority defined in MIN-MSG-WEIGHT is less than or equal to the priority of the message. The appropriate key values are permitted as priority specifications: *ALL denotes the lowest priority, and *NONE the highest. Keywords can be assigned a message priority by the user. Specifying *NONE as the priority causes all messages to be suppressed, except for those concerning internal errors. The default value MIN-MSG-WEIGHT=*NORMAL is preset as follows:

– In sort/merge runs where SORT is executing as a standalone program, messages with priority 2 or higher (or with the priority set for the particular installation or user ID) are output.

– When SORT is called as a subroutine, messages with priority 3 or higher are output.

Special cases may occur during interpretation of the SORT statements:

– Where SYSDTA is not a terminal, all inputs, e.g. the statements passed to SORT, are listed unchanged on SYSOUT.

– In interactive mode, depending on the GUIDANCE level set in the MODIFY-SDF-OPTIONS command (see the manual “Commands” [1]), either an incorrectly entered statement must be re-entered correctly following output of the SORT message or the correction dialog offered by SDF must be conducted. If, however, SORT only detects the error during the validation or planning phase, then all the statements have to be reentered.

At certain points during processing, e.g. at the start or end of a sort/merge run, when a user exit is activated, or in the event of an abnormal termination of the run, status messages reporting the current situation are output. In interactive mode, a response to some messages can be made only if the DIALOG option was selected in the user exit being used. These action responses can be entered in abbreviated form. Permitted responses are listed in the associated messages; characters that can be omitted from the response are enclosed in square brackets (e.g. C[ONTINUE]).

Notes

– SORT messages showing the execution status of the sort/merge program usually begin with the date, the clock time (time of day) and the CPU time used. Here, the date is given using the format yyyy-mm-dd, the time format is hh:mm:ss and the CPU time format is ssssss.ss. In the date format yyyy = year, mm = month and dd = day. In the time format, hh = hours, mm = minutes und ss = seconds .
The CPU time is relative to the start of the sort/merge run (SRT1001). Partial execution times must therefore be determined by subtraction; similarly, the total runtime is established by working out the difference between the clock times.

– The remarks below concerning the continuation or termination of the sort/merge run refer to interactive operation. In batch mode, if an invalid statement is encountered, the next statement is read. This usually results in abnormal termination of the sort/merge program.

– Messages SRT1016 to SRT1026 provide information on SORT performance. These messages are displayed whenever necessary and at the end of SORT. If the performance counter is set to zero or has remained unchanged since the last output, the message only appears if MIN-MSG-WEIGHT=*ALL has been entered in the SET-SORT-OPTIONS statement.

– Tables indicating the status of files following a normal and abnormal termination of the sort/merge run are given in section “Close processing for SORT files” on page 115.

– In multi-task sorts, there is a separate trace listing for each task involved (i.e. main task and subtask runs).
However, the trace listings of the subtask runs are produced only if the operand MIN-MSG-WEIGHT=*ALL was specified in the SET-SORT-OPTIONS statement or if an error has been detected in the sort subtask.

11.2 Message output in S variables

For a range of messages, message codes and inserts (number and semantics) are guaranteed as unchangeable components for future SORT and BS2000/OSD-BC versions. Such messages are marked as guaranteed messages.

Guaranteed messages of the product SORT

Guaranteed messages in SORT are messages which provide information on

– start or termination of SORT

– number of records sorted

– number of deleted, inserted or totaled records

The following table shows the numbers of these messages and their meanings:

Message number	Meaning
SRT1001	SORT is starting
SRT1002	SORT is terminating normally
SRT1003	Interruption by the user
SRT1016	Number of input records
SRT1017	Number of records to be sorted
SRT1018	Number of inserted input records
SRT1019	Number of deleted input records
SRT1020	Number of deleted SUM records
SRT1021	Number of sorted records
SRT1022	Number of inserted output records
SRT1023	Number of deleted output records
SRT1024	Number of deleted records on the basis of SELECT-INPUT-RECORDS or INCLUDE/OMIT
SRT1030	Number of output records
SRT1034	No records to be sorted
SRT1038	SORT aborted
SRT1059	SORT terminated with errors

When the chargeable product SDF-P is used, these guaranteed messages can be output in structured S variables. S variables allow you to access specific message data directly, without having to know the output layout of the messages. This makes it possible to control further processing in SDF-P procedures, irrespective of the contents of these variables.

For guaranteed messages the message attribute “Warranty” is documented by
“ ◆ Warranty: Y” in the line after the message text.

Procedure

An S variable which can record messages must comprise a list of structures. Its name is freely selectable and is represented by `varname` in the following. You can define the variable with the following command:

```
/DECLARE-VARIABLE NAME=varname(TYPE=*STRUCTURE(DEFINITION=*DYNAMIC)), -
/                 MULTIPLE-ELEMENTS=*LIST,SCOPE=*TASK
```

Note

SCOPE=*TASK need only be specified if the variable is also to be valid in procedures which will be called after definition of the variable. This is especially the case when SORT is started via the SORT-FILE command, because this command is implemented as a procedure.

For each message to be output, a list element of the variable is provided for, which is itself a structure. This structure comprises the following elements:

Name of the structure element	Contents
`varname`(*LIST).MSG-TEXT	Complete message text
`varname`(*LIST).MSG-ID	Message code
`varname`(*LIST).I0	Insert 0
⋮	
`varname`(*LIST).I`n`	Insert `n` (`n` is the number of inserts, which depends on the message)

Once the S variable has been defined, you assign the message stream to it:

```
/ASSIGN-STREAM STREAM-NAME=SYSMSG,TO=*VARIABLE(varname)
```

Then you start SORT or a program which calls SORT as a subroutine. After termination of SORT or the program, you reassign the message stream to the standard output medium for messages.

```
/ASSIGN-STREAM STREAM-NAME=SYSMSG,TO=*STD
```

The variable `varname` contains all **guaranteed** messages that are output between the two ASSIGN-STREAM command. This means:

– If you start SORT as an autonomous program with START-SORT or SORT-FILE, the variable contains all guaranteed messages that were output by SORT as well as the guaranteed messages of the binder loader system (BLS) which were output during the SORT run.

– If you call SORT in a subroutine, the variable also contains all guaranteed messages that were output by the main program.
To suppress output of the additional messages, you can execute the two ASSIGN-STREAM commands by means of a CMD macro, immediately before and after the SORT call in the main program.

Note

The MIN-MSG-WEIGHT operand in the SET-SORT-OPTIONS statement has no effect on the message output in S variables.

For a detailed description of the S variables, see the “SDF-P (BS2000)” manual [13].

Evaluation of S variables

The contents of an S variable can be output by means of the SHOW-VARIABLE command. After a SORT run, an S variable could contain the following:

```
/show-variable variable-name=sortmip
.
.
.
SORTMIP(*LIST).MSG-ID = SRT1001
SORTMIP(*LIST).I0 =
SORTMIP(*LIST).I1 = 2014-12-10/18:11:11
SORTMIP(*LIST).I2 = 000000.00
SORTMIP(*LIST).I3 = 08.0A00/BS2000V18.0
SORTMIP(*LIST).MSG-TEXT = %  SRT1016  SORT/MERGE INPUT RECORDS:
.............................10 (FROM 01)
SORTMIP(*LIST).MSG-ID = SRT1016
SORTMIP(*LIST).I0 =
SORTMIP(*LIST).I1 = ...........10
SORTMIP(*LIST).I2 = 01
SORTMIP(*LIST).MSG-TEXT = %  SRT1030  SORT/MERGE OUTPUT RECORDS:
.............................10
SORTMIP(*LIST).MSG-ID = SRT1030
SORTMIP(*LIST).I0 =
SORTMIP(*LIST).I1 = ...........10
SORTMIP(*LIST).MSG-TEXT = %  SRT1002 2014-12-10/18:11:11/000000.08 SORT/
MERGE COMPLETED
SORTMIP(*LIST).MSG-ID = SRT1002
SORTMIP(*LIST).I0 =
SORTMIP(*LIST).I1 = 2014-12-10/18:11:11
SORTMIP(*LIST).I2 = 000000.08
```

The individual elements of the variables can also be specifically addressed. For example, the following command gives you the structure of the second message:

```
/show-variable variable-name=sortmip#2
```

The message code of the third message is established in the same way:

```
/show-variable variable-name=sortmip#3.msg-id
```

In general, the position of a sought message within the variable cannot be predicted. It is therefore advisable to use an SDF-P procedure with a FOR loop to query a specific message, as shown in the following example.

Example of an SDF-P procedure for evaluating S variables

After a SORT run you should check whether message SRT1034 (no records sorted) has been output. The result is displayed on the screen.

```
/DECLARE-VARIABLE NAME=SORTMIP(TYPE=STRUCTURE),             -  (1)
/                 MULTIPLE-ELEMENTS=LIST,SCOPE=TASK
/DECLARE-VARIABLE NAME=CURRENT(TYPE=STRUCTURE)                 (2)
/DECLARE-VARIABLE NAME=FOUND,TYPE=BOOLEAN,INITIAL-VALUE=FALSE  (3)
/FREE-VARIABLE NAME=SORTMIP                                    (4)
/ASSIGN-STREAM STREAM-NAME=SYSMSG,TO=*VARIABLE(SORTMIP)        (5)
/SORT-FILE  INPUT-FILES=EINGABE,OUTPUT-FILE=AUSGABE            (6)
/ASSIGN-STREAM STREAM-NAME=SYSMSG,TO=*STD                      (7)
/FOR CURRENT=*LIST(SORTMIP)                                    (8)
/  IF (CURRENT.MSG-ID = 'SRT1034')
/    WRITE-TEXT '--------------------------------------'
/    WRITE-TEXT '---      no records sorted       ---'
/    WRITE-TEXT '--------------------------------------'
/    FOUND = TRUE
/  END-IF
/END-FOR
/IF (FOUND = FALSE)                                            (9)
/  WRITE-TEXT '--------------------------------------'
/  WRITE-TEXT '---  at least one record sorted  ---'
/  WRITE-TEXT '--------------------------------------'
/END-IF
```

(1) The S variable SORTMIP is defined as a list of structures.

(2) The structure variable CURRENT is defined. It is to be used as the control variable of the FOR loop.

(3) The bit switch FOUND is defined for recording the result of the search for message SRT1034. It is initialized with the value FALSE.

(4) If the SORTMIP variable already existed in this procedure before the definition, its contents are deleted.

(5) As of now, guaranteed messages are stored in the SORTMIP variable.

(6) SORT is started.

(7) Storage of messages in the variable is stopped.

(8) With the help of a FOR loop, the whole structure list is searched for an entry with the message ID SRT1034. If such an entry is found, the message “no records sorted” is output and the FOUND switch is set to TRUE.

(9) When the FOR loop is completed, SORT checks whether the FOUND switch is still set to FALSE. If so, the text “at least one record sorted” is output.

11.3 SORT/MERGE messages

You can query the message text and optionally also the meaning and response texts for a message code with the HELP-MSG-INFORMATION command.

You can also use an HTML application to find the SORT/MERGE messages on the Manual Server (URL: *http://manuals.ts.fujitsu.com*) and on the “BS2000 SoftBooks” DVD.

12 Appendix

12.1 SORT error handling

12.1.1 Handling internal SORT errors

Internal SORT errors are recognizable by

– command return codes with subcode1=32

– the SORT message “SORT INTERNAL ERROR....” or

– some other anomalous behavior by the sort/merge program.

If internal errors of this type occur, the following documentation should be collected and made available to system maintenance personnel.

Trace listings

The trace listings should record the command and sort/merge statement sequences. If relevant, the operator’s console log should also be made available.

Memory dumps

Memory dumps provide a record of exception conditions.

On reporting an internal error via message SRT1039, SORT also prints a special SORT dump. In addition, a general dump is output to a file which can be printed using the utility routine DAMP. Dump output can be controlled by means of the DUMP operand of the SET-SORT-OPTIONS statement.

Data errors

If the specified format of a sort, sum or match field is not compatible with the contents of the input record, the following message is output:

```
SRT1047 *** DATA ERROR X'ff', PROGRAM COUNTER X'zzzzzzzz' ***
```

In this case, depending on the DUMP option set, a dump and the records concerned are output on SYSLST. In interactive mode, up to 8 lines per record are additionally written to SYSOUT.

For other types of error, the exception condition should be recorded by means of the DUMP option.

Files

The command

```
/SHOW-FILE-ATTRIBUTES FILE-NAME=filename INFORMATION=*ALL-ATTRIBUTES
```

should be issued so that the file attributes of all files involved in the sort/merge run will be recorded in the listing. Moreover, all the files should be preserved in the state corresponding to the error situation. This applies particularly to errors in connection with a checkpoint or RESTART-PROGRAM.

12.1.2 Error information when SORT is called as a standalone program

When SORT is run as a standalone program, the successful or abnormal termination of the run is indicated as follows:

– normal (error-free) termination of SORT using the macro TERM MODE=NORMAL

– abnormal termination or abort of SORT using the macro TERM UNIT=STEP,MODE=ABNORMAL

If SORT is called in a procedure and subsequently terminates in error (TERM UNIT=STEP,MODE=ABNORMAL), the user can take action by branching into the procedure via the spin-off mechanism and issuing appropriate commands (e.g. SET-JOB-STEP command). (See also the SET-JOB-STEP command in the manual “Commands” [1].)

Status of the SORT run in a job variable

The user can use a program-monitoring job variable (JV) to keep track of the status of the SORT run (see the “Job Variables” manual [8]). This presupposes that the JV function is installed on the system. Information about the sort/merge run is stored in the JV by SORT and can be accessed by the user. The job variable is assigned by the user when calling SORT, with the command:

```
/START-SORT MONJV=jvname
```

The program-monitoring job variable contains the character “P” in byte 16 to identify it as such. Bytes 0 - 6 have the structure and contents shown below. All other bytes are reserved:

<table>
<tr><th>Byte 0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr>
<tr><td colspan="3" rowspan="2">Status indicator</td><td colspan="4">Return code indicator</td></tr>
<tr><td>Termination code</td><td colspan="3">Program information</td></tr>
</table>

The status indicator reflects the current status of the SORT run and can take the following values:

$R_	SORT running	SORT run has been initiated
$T_	Normal end of SORT run	SORT run has been successfully completed (TERM MODE=NORMAL).
$A_	Abnormal end of SORT run	SORT run has been aborted due to a program error or because a defined error exit has been activated (TERM UNIT=STEP,MODE=ABNORMAL).

The next 4 bytes (bytes 3 - 6) contain the return code indicator. This consists of the termination code (byte 3) and program information (bytes 4 - 6).

The termination code can take the following values:

C’0’	Normal end of SORT run. No errors.
C’1’	One or more warning messages issued during SORT run. The results should be checked.
C’2’	Abnormal end of SORT run. The results contain errors or are incomplete.
C’3’	SORT run terminated or aborted due to serious error. Results unusable or not available.

The program information consists of the last 3 decimal digits (in EBCDIC) of the key of the error message or warning message describing the error.

Example

If the SORT run aborts with error message SRT1206, the return code indicator in bytes 3 - 6 will contain C’3206’.

Note

To evaluate the job variable, the user should issue the command /SHOW-JV; this causes the job variable contents to be output (see GETJV command, “Job Variables” manual [8]).

12.1.3 Error information when SORT is called as a subroutine

Error information in register 15

When SORT is called as a subroutine, a return code is passed in the low-order byte of register 15 upon termination of the sort/merge run. It may take one of the following values:

X’00’	Normal termination of the sort/merge run.
X’FF’	Abnormal termination of the sort/merge run. The error condition is reported in a message.

If there is an error, the two high-order bytes additionally contain the last 4 places of the SORT message key. The key is stored as an unsigned packed decimal number.

Example

If the SORT run terminates abnormally with error message SRT1035, register 15 contains the value X’103500FF’.

Error information in the RCF area

If SORT is called as a subroutine and control information is supplied with the SRT0 or SRT1 macro, an RCF area can be defined in the main program (RCF operand in the macro call). SORT places return information in this area at the end of the sort/merge run and also uses it as a store for SORT messages output during the run. The transfer control area points to the address of the RCF area (absolute (A-type) or symbolic (S-type) address).

12.1.4 Structure of the RCF area

Byte 0	**1**	**2**	**3**	**4**	**…**	**.**
RC				Store for SORT messages		

The various fields of the RCF area have the following meanings:

RC

The RC information has the same structure as the return code indicator in the job variable (see page 398).

Bytes 1 - 3: Program information
This contains the last 3 decimal digits (in EBCDIC) of the key of the error or warning message which describes the error.

Example

If the SORT run terminates abnormally with error message SRT1035, the RC information area contains the value C’3045’.

Store for SORT messages

The messages are stored consecutively up to the end of the RCF area (SYSLST format). Messages have the following format (where | denotes the total length of a message):

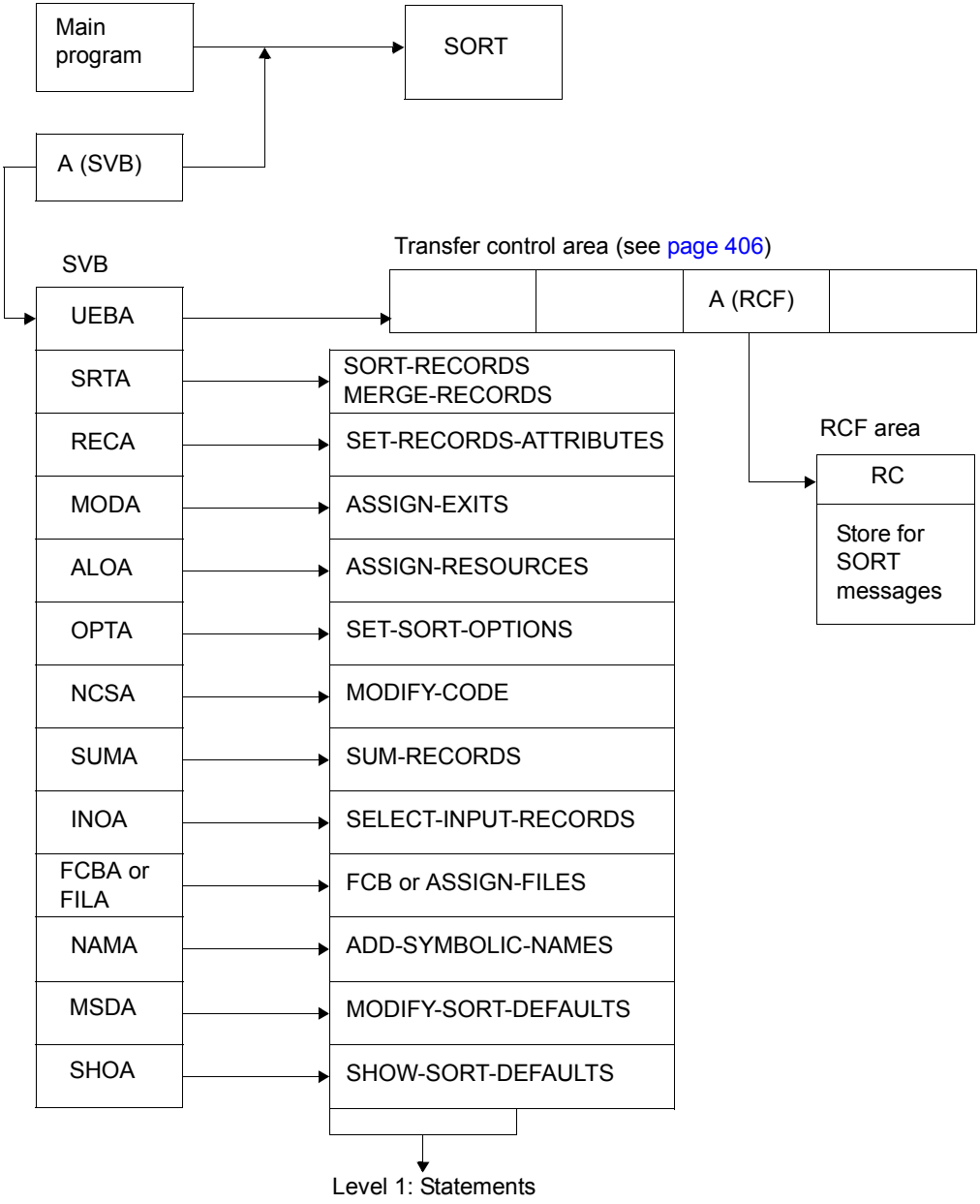
l	undefined	printable message

If the RCF area is not sufficient for displaying all messages, the last message is truncated. The length field is not corrected.

12.2 Structure of SORT control tables

12.2.1 Table overview

The tables and areas that are set up internally in order to supply control information to SORT as a subroutine have the following structure:


Main program
SORT
A (SVB)
SVB
Transfer control area (see page 406)
UEBA
A (RCF)
SRTA
SORT-RECORDS
MERGE-RECORDS
RECA
SET-RECORDS-ATTRIBUTES
RCF area
RC
MODA
ASSIGN-EXITS
Store for SORT messages
ALOA
ASSIGN-RESOURCES
OPTA
SET-SORT-OPTIONS
NCSA
MODIFY-CODE
SUMA
SUM-RECORDS
INOA
SELECT-INPUT-RECORDS
FCBA or FILA
FCB or ASSIGN-FILES
NAMA
ADD-SYMBOLIC-NAMES
MSDA
MODIFY-SORT-DEFAULTS
SHOA
SHOW-SORT-DEFAULTS
Level 1: Statements

12.2.2 Input block SVB

Register 1 points to an input block (referred to by the abbreviation SVB). This consists of an address list of variable length in which the positions of the individual addresses are fixed. The address list must be aligned on a word boundary. The first word contains an address that points to the transfer control area. The remaining words contain addresses pointing to the individual statements (level 1); care must be taken to preserve the proper sequence. A zero address must be used to represent any statements not included in the input block. Zero addresses at the end of the SVB may be omitted, but in this case the most significant bit of the last address has to be set to 1 (COBOL convention).

Structure of input block SVB

Bytes 0 - 3: UEBA
The 1st word of the input block SVB contains the address of the transfer control area.

Bytes 4 - 7: SRTA
Address of the SORT-RECORDS/MERGE-RECORDS statement.

Bytes 8 - 11: RECA
Address of the SET-RECORD-ATTRIBUTES statement.

Bytes 12 - 15: MODA
Address of the ASSIGN-EXITS statement.

Bytes 16 - 19: ALOA
Address of the ASSIGN-RESOURCES statement.

Bytes 20 - 23: OPTA
Address of the SET-SORT-OPTIONS statement.

Bytes 24 - 27: NCSA
Address of the MODIFY-CODE statement.

Bytes 28 - 31: SUMA
Address of the SUM-RECORDS statement.

Bytes 32 - 35: INOA
Address of the SELECT-INPUT-RECORDS statement.

Bytes 36 - 39: :One of the following:

– FCBA
  Address of the FCB reference table
  This address is possible for level 2 only. It is not present if PARMOD=31.

  or

– FILA
  Address of the ASSIGN-FILES statement.

Bytes 40 - 43: NAMA
Address of the ADD-SYMBOLIC-NAMES statement.

Bytes 44 - 47 MSDA
Address of the MODIFY-SORT-DEFAULTS statement.

Bytes 48 - 51 SHOA
Address of the SHOW-SORT-DEFAULTS statement.

Notes

– When input is at level 0 the SVB consists only of the address of the transfer control area (UEBA).
  At level 1, the SVB must contain at least the address pointers to the transfer control area (UEBA) and the SORT-RECORDS/MERGE-RECORDS statement or the SORT/MERGE statement table. These two areas must be specified in every SVB.

– At level 1, the statements are variable-length records prefixed by a 4-byte record length field. The record length is contained in the first halfword of the record length field.

– The user can set up the SVB input block and the transfer control area using the SRT0 and SRT1 macros.

12.2.3 Transfer control area

The transfer control area, the area referenced by the first word of the SVB, is 5 words long and contains the following information:

– The transfer indicator (TI).
Among other things, this indicates which transfer level is being used to pass statements (level 0 or 1).

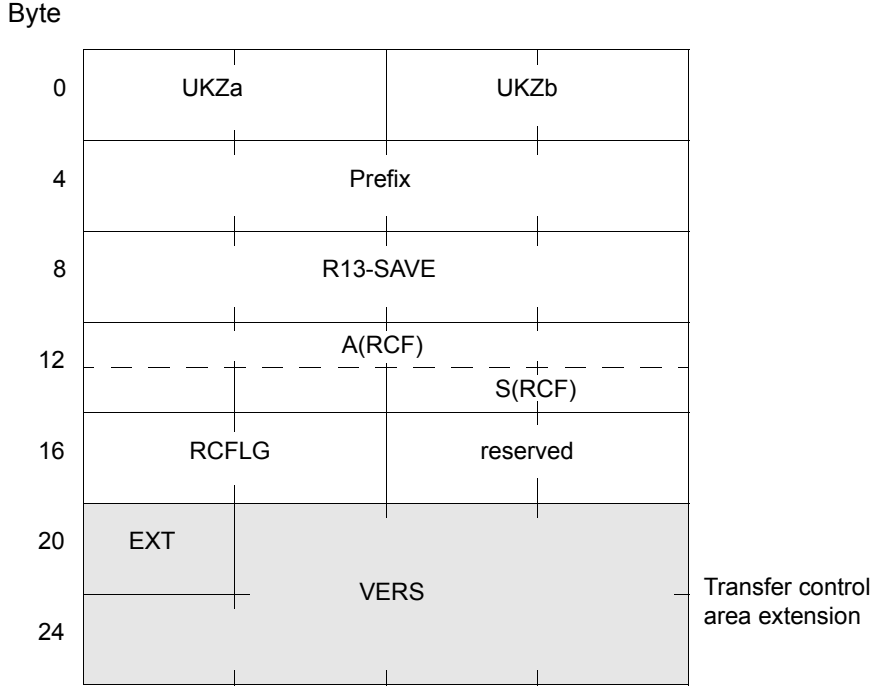
– A 4-character prefix which acts as an identifier for the SORT run.

– A save area for register 13 when the SORT access method SORTZM is used.

– Address and length of the RCF area.
SORT uses the RCF area to pass the return code, and also as a store for SORT messages concerning the status of the sort/merge run (up to the specified length).

– Transfer control area extension:
version number of SORT

Structure of the transfer control area



The individual fields of the transfer control area have the following meanings:

Bytes 0 - 1: UKZa
These bytes contain the information for the MULTI operand in SRT0 and SRT1

X'...0' MULTI = STD

X'...1' MULTI = OPT

X'...2' MULTI = NOIMON

Bytes 2 - 3: UKZb
These bytes contain the transfer indicator, which tells SORT at which level and in which format to expect the SORT statements. The following values can be specified in the UKZ (the weighting factors are given in each case):

X'...0' The SORT statements are passed via SYSDTA (level 0).

X'...4 The SORT statements are passed in the main memory area of the calling program (level 1). Register 1 points to the SVB.

X'...8' The SORT statements are passed as statement tables in the main memory area of the calling program (level 2). Register 1 points to the SVB.

X'..0.' STXIT entries are used by SORT at any level.

X'..2.' STXIT entries are not used by SORT at any level. Special dumps are not taken in exception conditions, and the option of using the SEND-MSG command to intervene in the processing is not available.

X'..4.' Transfer control area extension exists.

X'..8.' The statements are expected in SDF format.

X'.0..' No RCF area is set up. No prefix. Save area for register 13.

X'.2..' Address of the RCF area is specified in A format (as an A-type, or absolute, address).

X'.3..' Address of the RCF area is specified in S format (as an S-type, or symbolic, address).

X'.4..' The R13-SAVE area is used by the SORTZM macros to save the contents of register 13 (provided RDONLY=NO is specified).

X'.8..' Prefix present.

X'0...' SORT messages are output to SYSOUT.

X'1...' SORT messages are output to SYSLST.

X'2...' SORT messages are output to SYSOUT and SYSLST.

X'3...' SORT messages are output to the RCF area only, not to SYSOUT and/or SYSLST.

Note

Weighting factors may be usefully combined by logical ORing.

Example

X’2284’ means:
SORT statements are passed at level 1. Register 1 points to the SVB.
Statements are expected in SDF format.
The address of the RCF area is in A format.
Messages are output to SYSOUT and SYSLST.

Bytes 4 - 7: Prefix
A 4-character prefix identifying the SORT run.

Bytes 8 - 12: R13-SAVE
Save area for register 13 when the SORT access method SORTZM is used (with RDONLY=NO).

Bytes 12 - 15: A(RCF)
Address of the RCF area in A address format.

Bytes 14 - 15: S(RCF)
Address of the RCF area in S address format.

Bytes 16 - 17: RCFLG
Length of the RCF area.

Byte 18 - 19: reserved

Bytes 20: EXT
Transfer control area extension

X‘8.‘ SORT version expected

Bytes 21- 27: VERS
SORT version in the format nn.nann

12.2.4 SORT statement tables

Statement tables (in ISP via level 2) continue to be supported for reasons of compatibility.

12.3 Sort table UTF-16

The table below shows all the codes of Unicode UTF-16 which are currently supported. They are listed in the order in which they are arranged by SORT.

The columns have the following meaning:

hex		Hexadecimal encryption
Character		Printable form of the character or its meaning
var	x	Variables collation element
	(x)	Character to which a variable collation element belongs
multi	x	The character consists of multiple base characters (no diacritics)
Level 1	=	At level 1 the character has the same value as the character in the previous line
	>	At level 1 the character has a higher value than all preceding characters
Level 2	=	At level 2 the character has the same value as the character in the previous line
	>	At level 2 the character has a higher value than the character in the previous line
Level 3	=	At level 3 the character has the same value as the character in the previous line
	>	At level 3 the character has a higher value than the character in the previous line

hex	Character	var	multi	Level		
				1	2	3
0301	´					
0300	`			=	>	
0306	˘			=	>	
0302	^			=	>	
030C	ˇ			=	>	
030A	°			=	>	
0308	¨			=	>	
030B	˝			=	>	
0303	~ **COMBINING TILDE**			=	>	
0307	˙			=	>	
0327	¸			=	>	
0328	˛			=	>	
0304	¯			=	>	
0311	^			=	>	
031B	’			=	>	
0323	.			=	>	
0009	HORIZONTAL TABULATION*	x		>		
000A	LINE FEED*	x		>		
000B	VERTICAL TABULATION*	x		>		
000C	FORM FEED*	x		>		
000D	CARRIAGE RETURN*	x		>		
0085	NEXT LINE*	x		>		
0020	SPACE*	x		>		
00A0	NO-BREAK SPACE*	x		=	=	>

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
0060	`	x		>		
0384	΄	x		>		
00B4	´	x		=	=	=
005E	^	x		>		
00AF	¯	x		>		
02D8	˘	x		>		
02D9	˙	x		>		
00A8	¨	x		>		
0385	΅	x		=	>	
02DD	˝	x		>		
00B8	¸	x		>		
02DB	˛	x		>		
005F	_	x		>		
00AD	-	x		>		
002D	-	x		>		
2015	–	x		>		
002C	,	x		>		
003B	;	x		>		
003A	:	x		>		
0021	!	x		>		
00A1	¡	x		>		
003F	?	x		>		
00BF	¿	x		>		
002E	.	x		>		

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
00B7	·	x		>		
0027	'	x		>		
2018	‘	x		>		
2019	’	x		>		
0022	"	x		>		
00AB	«	x		>		
00BB	»	x		>		
0028	(	x		>		
0029	)	x		>		
005B	[	x		>		
005D	]	x		>		
007B	{	x		>		
007D	}	x		>		
00A7	§	x		>		
00B6	¶	x		>		
00A9	©	x		>		
00AE	®	x		>		
0040	@	x		>		
002A	*	x		>		
002F	/	x		>		
2044	⁄	x		>		
005C	\	x		>		
0026	&	x		>		
0023	#	x		>		

* Character cannot be displayed

hex	Zeichen	var	multi	Ebene		
				1	2	3
0025	%	x		>		
02C7	ˇ	x		>		
00B0	°	x		>		
002B	+	x		>		
00B1	±	x		>		
00F7	÷	x		>		
00D7	×	x		>		
003C	<	x		>		
003D	=	x		>		
003E	>	x		>		
00AC	¬	x		>		
007C	|	x		>		
00A6	¦	x		>		
007E	~	x		>		
2264	≤	x		>		
00A4	¤			>		
00A2	¢			>		
0024	$			>		
00A3	£			>		
00A5	¥			>		
20AC	€			>		
20AF	₯	x		>		
0030	0			>		
0031	1			>		

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
00B9	¹			=	=	>
00BD	½	(x)		=	=	>
00BC	¼	(x)		>		
0032	2			>		
00B2	²			=	=	>
0033	3			>		
00B3	³			=	=	>
00BE	¾	(x)		=	=	>
0034	4			>		
0035	5			>		
0036	6			>		
0037	7			>		
0038	8			>		
0039	9			>		
0061	a			>		
0041	A			=	=	>
00AA	ª			=	=	>
00E1	á			=	>	
00C1	Á			=	=	>
00E0	à			=	>	
00C0	À			=	=	>
0103	ă			=	>	
0102	Ă			=	=	>
00E2	â			=	>	

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
00C2	Â			=	=	>
1EAB	ẫ			=	>	
1EAA	Ẫ			=	=	>
01CE	ǎ			=	>	
01CD	Ǎ			=	=	>
00E5	å			=	>	
00C5	Å			=	=	>
00E4	ä			=	>	
00C4	Ä			=	=	>
00E3	ã			=	>	
00C3	Ã			=	=	>
0105	ą			=	>	
0104	Ą			=	=	>
0101	ā			=	>	
0100	Ā			=	=	>
1EA1	ạ			=	>	
1EA0	Ạ			=	=	>
00E6	æ			>		
00C6	Æ			=	=	>
0062	b			>		
0042	B			=	=	>
0063	c			>		
0043	C			=	=	>
0107	ć			=	>	

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
0106	Ć			=	=	>
0109	ĉ			=	>	
0108	Ĉ			=	=	>
010D	č			=	>	
010C	Č			=	=	>
010B	ċ			=	>	
010A	Ċ			=	=	>
00E7	ç			=	>	
00C7	Ç			=	=	>
0064	d			>		
0044	D			=	=	>
010F	ď			=	>	
010E	Ď			=	=	>
0111	đ			>		
0110	Đ			=	=	>
00F0	ð			>		
00D0	Ð			=	=	>
0065	e			>		
0045	E			=	=	>
00E9	é			=	>	
00C9	É			=	=	>
00E8	è			=	>	
00C8	È			=	=	>
0114	Ĕ			=	>	

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
00EA	ê			=	>	
00CA	Ê			=	=	>
1EC5	ễ			=	>	
1EC4	Ễ			=	=	>
011B	ě			=	>	
011A	Ě			=	=	>
00EB	ë			=	>	
00CB	Ë			=	=	>
1EBD	ẽ			=	>	
1EBC	Ẽ			=	=	>
0117	ė			=	>	
0116	Ė			=	=	>
0119	ę			=	>	
0118	Ę			=	=	>
0113	ē			=	>	
0112	Ē			=	=	>
0066	f			>		
0046	F			=	=	>
0067	g			>		
0047	G			=	=	>
01F5	ǵ			=	>	
01F4	Ǵ			=	=	>
011F	ğ			=	>	
011E	Ğ			=	=	>

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
011D	ĝ			=	>	
011C	Ĝ			=	=	>
01E7	ǧ			=	>	
01E6	Ǧ			=	=	>
0121	ġ			=	>	
0120	Ġ			=	=	>
0123	ģ			=	>	
0122	Ģ			=	=	>
1E21	ḡ			=	>	
1E20	Ḡ			=	=	>
0068	h			>		
0048	H			=	=	>
0125	ĥ			=	>	
0124	Ĥ			=	=	>
1E25	ḥ			=	>	
1E24	Ḥ			=	=	>
0127	ħ			>		
0126	Ħ			=	=	>
0069	i			>		
0049	I			=	=	>
00ED	í			=	>	
00CD	Í			=	=	>
00EC	ì			=	>	
00CC	Ì			=	=	>

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
012D	ĭ			=	>	
012C	Ĭ			=	=	>
00EE	î			=	>	
00CE	Î			=	=	>
01D0	ǐ			=	>	
01CF	Ǐ			=	=	>
00EF	ï			=	>	
00CF	Ï			=	=	>
0129	ĩ			=	>	
0128	Ĩ			=	=	>
0130	İ			=	>	
012F	į			=	>	
012E	Į			=	=	>
012B	ī			=	>	
012A	Ī			=	=	>
1ECB	ị			=	>	
1ECA	Ị			=	=	>
0131	ı			>		
006A	j			>		
004A	J			=	=	>
0135	ĵ			=	>	
0134	Ĵ			=	=	>
006B	k			>		
004B	K			=	=	>

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
1E31	ḱ			=	>	
1E30	Ḱ			=	=	>
0137	ķ			=	>	
0136	Ķ			=	=	>
006C	l			>		
004C	L			=	=	>
013A	ĺ			=	>	
0139	Ĺ			=	=	>
013E	ľ			=	>	
013D	Ľ			=	=	>
013C	ļ			=	>	
013B	Ļ			=	=	>
0142	ł			>		
0141	Ł			=	=	>
006D	m			>		
004D	M			=	=	>
006E	n			>		
004E	N			=	=	>
0144	ń			=	>	
0143	Ń			=	=	>
0148	ň			=	>	
0147	Ň			=	=	>
00F1	ñ			=	>	
00D1	Ñ			=	=	>

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
1E45	ṅ			=	>	
1E44	Ṅ			=	=	>
0146	ņ			=	>	
0145	Ņ			=	=	>
2116	№ Char. equivalent to No (hex: 004E 0366)		x	=	>	
0272	ɲ			>		
014B	ŋ			>		
014A	Ŋ			=	=	>
006F	o			>		
0366	COMBINING LATIN SMALL LETTER O*			=	=	>
004F	O			=	=	>
00BA	º			=	=	>
00F3	ó			=	>	
00D3	Ó			=	=	>
00F2	ò			=	>	
00D2	Ò			=	=	>
014F	ŏ			=	>	
014E	Ŏ			=	=	>
00F4	ô			=	>	
00D4	Ô			=	=	>
1ED7	ỗ			=	>	
1ED6	Ỗ			=	=	>
01D2	ǒ			=	>	

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
01D1	Ǒ			=	=	>
00F6	ö			=	>	
00D6	Ö			=	=	>
0151	ő			=	>	
0150	Ő			=	=	>
00F5	õ			=	>	
00D5	Õ			=	=	>
014D	ō			=	>	
014C	Ō			=	=	>
01A1	ơ			=	>	
01A0	Ơ			=	=	>
1ECD	ọ			=	>	
1ECC	Ọ			=	=	>
0153	œ Char. equivalent to oe (hex: 006F 0065)		x	=	>	
0152	Œ Char. equivalent to OE (hex: 004F 0045)		x	=	=	>
00F8	ø			>		
00D8	Ø			=	=	>
0070	p			>		
0050	P			=	=	>
0071	q			>		
0051	Q			=	=	>
0138	ĸ			>		

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
0072	r			>		
0052	R			=	=	>
0155	ŕ			=	>	
0154	Ŕ			=	=	>
0159	ř			=	>	
0158	Ř			=	=	>
0157	ŗ			=	>	
0156	Ŗ			=	=	>
0213	ȓ			=	>	
0212	Ȓ			=	=	>
0073	s			>		
0053	S			=	=	>
015B	ś			=	>	
015A	Ś			=	=	>
015D	ŝ			=	>	
015C	Ŝ			=	=	>
0161	š			=	>	
0160	Š			=	=	>
1E61	ṡ			=	>	
1E60	Ṡ			=	=	>
015F	ş			=	>	
015E	Ş			=	=	>
1E63	ṣ			=	>	
1E62	Ṣ			=	=	>

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
00DF	ß Char. equivalent to ss (hex: 0073 0073)		x	=	>	
0074	t			>		
0054	T			=	=	>
0165	ť			=	>	
0164	Ť			=	=	>
0163	ţ			=	>	
0162	Ţ			=	=	>
0167	ŧ			>		
0166	Ŧ			=	=	>
0075	u			>		
0055	U			=	=	>
00FA	ú			=	>	
00DA	Ú			=	=	>
00F9	ù			=	>	
00D9	Ù			=	=	>
016D	ŭ			=	>	
016C	Ŭ			=	=	>
00FB	û			=	>	
00DB	Û			=	=	>
01D4	ǔ			=	>	
01D3	Ǔ			=	=	>
016F	ů			=	>	
016E	Ů			=	=	>

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
00FC	ü			=	>	
00DC	Ü			=	=	>
0171	ű			=	>	
0170	Ű			=	=	>
0169	ũ			=	>	
0168	Ũ			=	=	>
0173	ų			=	>	
0172	Ų			=	=	>
016B	ū			=	>	
016A	Ū			=	=	>
01B0	ư			=	>	
01AF	Ư			=	=	>
1EE5	ụ			=	>	
1EE4	Ụ			=	=	>
0076	v			>		
0056	V			=	=	>
0077	w			>		
0057	W			=	=	>
0175	ŵ			=	>	
0174	Ŵ			=	=	>
1E85	ẅ			=	>	
1E84	Ẅ			=	=	>
0078	x			>		
0058	X			=	=	>

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	**2**	**3**
0079	y			>		
0059	Y			=	=	>
00FD	ý			=	>	
00DD	Ý			=	=	>
1EF3	ỳ			=	>	
1EF2	Ỳ			=	=	>
0177	ŷ			=	>	
0176	Ŷ			=	=	>
00FF	ÿ			=	>	
0178	Ÿ			=	=	>
1EF9	ỹ			=	>	
1EF8	Ỹ			=	=	>
1E8F	ẏ			=	>	
1E8E	Ẏ			=	=	>
007A	z			>		
005A	Z			=	=	>
017A	ź			=	>	
0179	Ź			=	=	>
1E91	ẑ			=	>	
1E90	Ẑ			=	=	>
017E	ž			=	>	
017D	Ž			=	=	>
017C	ż			=	>	
017B	Ż			=	=	>

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
1E93	ẓ			=	>	
1E92	Ẓ			=	=	>
00FE	þ			>		
00DE	Þ			=	=	>
03B1	α			>		
0391	A			=	=	>
03AC	ά			=	>	
0386	Ά			=	=	>
03B2	β			>		
0392	B			=	=	>
03B3	γ			>		
0393	Γ			=	=	>
03B4	δ			>		
0394	Δ			=	=	>
03B5	ε			>		
0395	E			=	=	>
03AD	έ			=	>	
0388	Έ			=	=	>
03B6	ζ			>		
0396	Z			=	=	>
03B7	η			>		
0397	H			=	=	>
03AE	ή			=	>	
0389	Ή			=	=	>

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
03B8	θ			>		
0398	Θ			=	=	>
039B	Λ			>		
037A	ͺ			=	>	
0399	Ι			=	>	
03AF	ί			=	>	
038A	Ί			=	=	>
03CA	ϊ			=	>	
03AA	Ϊ			=	=	>
0390	ΐ			=	>	
03BA	κ			>		
039A	Κ			=	=	>
03BB	λ			>		
039B	Λ			=	=	>
03BC	μ			>		
00B5	µ (MICRO SIGN)			=	=	>
039C	Μ			=	=	>
03BD	ν			>		
039D	Ν			=	=	>
03BE	ξ			>		
039E	Ξ			=	=	>
03BF	ο			>		
039F	Ο			=	=	>
03CC	ό			=	>	

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
038C	Ό			=	=	>
03C0	π			>		
03A0	Π			=	=	>
03C1	ρ			>		
03A1	Ρ			=	=	>
03C3	σ			>		
03A3	Σ			=	=	>
03C2	ς			=	=	>
03C4	τ			>		
03A4	Τ			=	=	>
03C5	υ			>		
03A5	Υ			=	=	>
03CD	ύ			=	>	
038E	΄Υ			=	=	>
03CB	ϋ			=	>	
03AB	Ϋ			=	=	>
03B0	ΰ			=	>	
03C6	φ			>		
03A6	Φ			=	=	>
03C7	χ			>		
03A7	Χ			=	=	>
03C8	ψ			>		
03A8	Ψ			=	=	>
03C9	ω			>		

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
03A9	Ω			=	=	>
03CE	ώ			=	>	
038F	Ώ			=	=	>
0430	а			>		
0410	А			=	=	>
0431	б			>		
0411	Б			=	=	>
0432	в			>		
0412	В			=	=	>
0433	г			>		
0413	Г			=	=	>
0434	д			>		
0414	Д			=	=	>
0452	ђ			>		
0402	Ђ			=	=	>
0453	ѓ			>		
0403	Ѓ			=	=	>
0435	е			>		
0415	Е			=	=	>
0451	ё			=	>	
0401	Ё			=	=	>
0454	є			>		
0404	Є			=	=	>
0436	ж			>		

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
0416	Ж			=	=	>
0437	з			>		
0417	З			=	=	>
0455	s			>		
0405	S			=	=	>
0438	и			>		
0418	И			=	=	>
0456	i			>		
0406	I			=	=	>
0457	ï			>		
0407	Ï			=	=	>
0439	й			>		
0419	Й			=	=	>
0458	j			>		
0408	J			=	=	>
043A	к			>		
041A	К			=	=	>
043B	л			>		
041B	Л			=	=	>
0459	љ			>		
0409	Љ			=	=	>
043C	м			>		
041C	М			=	=	>
043D	н			>		

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
041D	Н			=	=	>
045A	њ			>		
040A	Њ			=	=	>
043E	о			>		
041E	О			=	=	>
043F	п			>		
041F	П			=	=	>
0440	р			>		
0420	Р			=	=	>
0441	с			>		
0421	С			=	=	>
0442	т			>		
0422	Т			=	=	>
045B	ћ			>		
040B	Ћ			=	=	>
045C	ќ			>		
040C	Ќ			=	=	>
0443	у			>		
0423	У			=	=	>
045E	ў			>		
040E	Ў			=	=	>
0444	ф			>		
0424	Ф			=	=	>
0445	х			>		

* Character cannot be displayed

hex	Character	var	multi	Level		
				1	2	3
0425	Х			=	=	>
0446	ц			>		
0426	Ц			=	=	>
0447	ч			>		
0427	Ч			=	=	>
045F	ҷ			>		
040F	Ҷ			=	=	>
0448	ш			>		
0428	Ш			=	=	>
0449	щ			>		
0429	Щ			=	=	>
044A	ъ			>		
042A	Ъ			=	=	>
044B	ы			>		
042B	Ы			=	=	>
044C	ь			>		
042C	Ь			=	=	>
044D	э			>		
042D	Э			=	=	>
044E	ю			>		
042E	Ю			=	=	>
044F	я			>		
042F	Я			=	=	>

* Character cannot be displayed

Related publications

You will find the manuals on the internet at *http://manuals.ts.fujitsu.com*. You can order printed copies of those manuals which are displayed with an order number.

[1] **BS2000/OSD-BC**
Commands
User Guide

[2] **BS2000/OSD-BC**
Introductory Guide to DMS
User Guide

[3] **BS2000/OSD-BC**
DMS Macros
User Guide

[4] **Assembler Instructions** (BS2000/)
Reference Manual

[5] **ASSEMBH** (BS2000)
Reference Manual

[6] **BS2000/OSD-BC**
Executive Macros
User Guide

[7] **SDF** (BS2000)
SDF Dialog Interface
User Guide

[8] **JV** (BS2000)
Job Variables
User Guide

[9] **BS2000/OSD-BC**
Introductory Guide to Systems Support
User Guide

[10] **XHCS** (BS2000)
8-Bit Code and Unicode Processing in BS2000
User Guide

[11] **IMON** (BS2000)
Installation Monitor
User Guide

[12] **SORT** (BS2000)
Ready Reference

[13] **SDF-P** (BS2000)
Programming in the Command Language
User Guide

[14] **POSIX** (BS2000)
POSIX Basics for Users and System Administrators
User Guide

[15] **POSIX** (BS2000)
Commands
User Guide

[16] **Unicode in BS2000**
Introduction

Index

0
level 209
level - SORT call at 357, 359
1
level 209
level - SORT call at 361
24-bit addressing, user exit, note 348
31-bit addressing, user exit, note 348

A
abnormal SORT termination 398
abnormal termination of sort/merge run 116
access to POSIX 117
ACS 119
ADD-FILE-LINK command 91, 103
ADD-SYMBOLIC-NAMES statement 405
address field 31
alias catalog service 119
allocation of storage space for work files 106
ASSIGN-EXITS statement 404
OUTPUT 345
ASSIGN-FILES statement 23, 94, 293, 296, 405
example 288
ASSIGN-RESOURCES statement 153, 404
assignment
of files 287
of SORT files 92
auxiliary files 108
on disk 110
on tape 109

C
calculation
of auxiliary files 108
of the CORE value 265
of work file size 106
calling SORT
as a subroutine 206
as a subroutine at level 0 210
as a subroutine at level 1 211
at level 0 212
at level 1 215
CCS
coded character set 81
EDF03DRV 321
CCSN (coded character set name) 81
change file, attributes, default definitions 329
change sorting order
FORMAT=EBCDIC-DIN, example 315
MODIFY-CODE, example 318
character set
coded 81
codes 38
extended 81
characters
hexadecimal, MODIFY-CODE, example 318
checkpoint 263
checkpoint file 112
close processing 113
CHECKPOINT operand
MERGE-RECORDS statement 263
SORT-RECORDS statement 263
close processing
auxiliary files 110
checkpoint files 113
SORT files, overview 115, 116
work files 107

closing
input files 96
merge input files 98
code 81
code conversion for sort fields
ETB user exit 255
VIRTUAL-TRANSLATE user exit 256
code conversion table
for EBCDIC to extended ASCII 46
for extended ASCII to EBCDIC 47
coded character set (CCS) 38, 81
coded character set name (CCSN) 81
coexistence of product versions 283
command
ADD-FILE-LINK 91, 103
GETJV 400
representation of syntax 126
RESTART-PROGRAM 264
SEND-MESSAGE-TO 261
SHOW-JV 400
command return code 202
COMPOUND-RECORD, example 329
condition for selection
SELECT-INPUT-RECORDS, example 325
constant field 56
control fields 37
conversion table
for EBCDIC to extended ASCII code 46
for extended ASCII to EBCDIC code 47
for TRANSLATE-CHARACTER 51, 52
conversion, sort field 43
CORE allocation 265
CORE operand
ASSIGN-RESOURCES statement 266
CORE value 265
calculation 265
creating
auxiliary files 108
work files 105
cycle sorting 268

D

default value, modify 152
definition sequence for sort/merge runs 21
DIN, see EBCDIC
differences, example 315
disk auxiliary files 110
DOMINO phase 19
dummy files 89

E

E15 341
INPUT user exit 341
E23
ASSIGN-EXITS 345
see E35 345
see new E35 345
E35
ASSIGN-EXITS OUTPUT 345
see old E23 345
see OUTPUT 345
see user exit OUTPUT 345
EBCDIC-DIN format 44
EBCDIC-INTERNATIONAL format 44
EBCDIC-ISO-EBCDIC format 45
EBCDIC, see DIN
differences, example 315
EDF03DRV
CCS 321
example 321
edit mask 60
end merge and output phase 19
error information 398, 400
ETB=PHYSICAL-TRANSLATE
example 349
example
define condition with SELECT-INPUT-RECORDS 325
DIN, see EBCDIC, differences 315
EDF03DRV 321
FORMAT=MODIFY-CODE 318
full sort of multiple files 310
full sort, input file = output file 313
ISAM file sort 308
merging sorted files 339
SELECT-INPUT-RECORDS 325
sort field, see new ISAM key 308
SRT0 macro 357

SRT1 macro 361
SRTCLSE macro 367
SRTGET macro 367
SRTOPEN macro 367
SRTPUT macro 367
summation 325
symbolic field name 313
TRANSLATE-CHARACTER 321
EXLST exits 251, 253
EXLST-FOR-INPUT user exit 233, 251
EXLST-FOR-OUTPUT user exit 233, 253
extend, sum field overflow 325
extended character set 81
extended host code support (XHCS) 81
EXTENDED-CHARACTER format 48, 321
EXTERNAL-COMPARE user exit 233, 258

F

field name, symbolic, example 313
file
assignment 287
change attributes 329
full sort, multiple, example 310
input file = output file, example 313
merging, example 339
variable record length 303
file attributes
of merge input files 97
of multiple input files 94
of the output file 99
file link name 90
of the input file 93
of the output file 99
SORTIN 287, 305, 366, 371
SORTOUT 287, 288, 305, 311, 357, 366, 371
SORTWKEX 109
file name, preset 283
file processing with EDT, ISAM key problems 309
files larger than 32 GBytes 120
filler character 60
FINISH-INPUT entry, user exits 235
fixed
file names 283
IDs 283
format
EBCDIC-DIN 44
EBCDIC-INTERNATIONAL 44
EBCDIC-ISO-EBCDIC 45
EXTENDED-CHARACTER 48, 321
MODIFY-CODE 45
PHYSICAL-TRANSLATE 45
TRANSLATE-CHARACTER 48, 321
UNICODE-CHARACTER 53
VIRTUAL-TRANSLATE 45
FORMAT=MODIFY-CODE, example 318
full sort 25
input file = output file, example 313
multiple files, example 310

G

GETJV command 400

H

hexadecimal character pairs
MODIFY-CODE, example 318

I

IGNORE-LENGTH-FIELD 118, 171, 199
ILSORT entry point 206
IMON 281
important information symbol 15
input and presorting phase 19
input block SVB 210, 211, 404
input files
for merge runs 97
for sort runs 93
INPUT user exit 233, 239
example 341
installation 281
INT user exit 233, 261
interchangeability 285
internal merge phase 19
internal SORT errors 397
ISAM file sort, example 308
ISAM files, sort in ascending order 308
ISAM key 31, 101
changed in sorted file 309
duplicate 312

problems with, file processing with EDT 309
ISAM output files 101
ISP statements 127

J
job variable 399

L
level 0 209, 210, 359
SORT call at 357
level 1 209, 211
SORT call at 361
Link name
SORTWK 105
SORTWKx 105
SORTWKxx 108

M
macro
SRTCLSE, example 367
SRTGET, example 367
SRTOPEN, example 367
SRTPUT, example 367
macros
for SORT 212
SRT1, syntax note 364
main code 202
main task 272
mask field 60
match constants 67
match fields 66
MERGE-RECORDS statement 36, 263, 339, 404
best use 310
merging 17, 35
merging files, example 339
message priority 389
messages 389
MODIFY-CODE format 45
MODIFY-CODE statement 404
example 318
FORMAT=, example 318
MODIFY-SORT-DEFAULTS, operand
description 152
modifying default values 152
MODULE entry, user exits 236
MSG operand, OPTION statement 389
multi-file/multi-volume set 96
multi-task sorting 271
multiple sorting
example 371
with SORTZM 222
multiple sorting with SORTZM
example 380

N
NEUTRAL-TRANSLATE user exit 233
normal SORT termination 398
normal termination of sort/merge run 115

O
object module library 92
SORTMODS 114
open system 117
opening
input files 95
merge input files 98
operand REMAINDER-EXPLICIT
remainder fields 368
optimization of sort runs 265
OPTIMIZATION operand
SET-SORT-OPTIONS statement 279
order, change, FORMAT=EBCDIC-DIN,
example 315
output file 99
OUTPUT user exit 233, 345
in 24-bit addressing 246
in 31-bit addressing 244
overflow, extend sum field 325
overlap, sort field 42
overlapping sum fields 58

P
PAM key 32
PARAMETER-MODE
addressing, note 348
example 345
important note 348

passing control information to SORT 209
PHYSICAL-TRANSLATE
format 45
user exit 233, 255
user exit, example 349
PHYSICAL-TRANSLATE user exit
example 349
planning phase 19
PLANNING user exit 233, 238, 265
POSIX 117
file 117
file system 117
input file 94, 97
output file 104
preparatory phase 19
preventing overflow in SUM-RECORDS 325
print mask 60
priority classes 266
product versions, coexistence 283
program information 399

R

RCF area 213, 216, 400
Readme file 13
RECA, SET-RECORD-ATTRIBUTES 404
recoding of sort fields
ETB user exit 255
VIRTUAL-TRANSLATE user exit 256
record format/record length modification 76
using SET-RECORD-ATTRIBUTES statement 77
via defined user exits 77
via undefined user exits 78
record length field
as sort field 41, 118
in sort statement 118, 157, 171, 199, 303
record length, calculation of variable records 295
record selection during input 76
RECORDS-PER-CYCLE operand
MERGE-RECORDS statement 263
records, summation 79
register conventions, user exits 236
remainder fields 54
REMAINDER-EXPLICIT operand 72, 330, 335, 368, 374
REMAINDER-EXPLICIT 72, 330, 335, 368, 374
RESTART-PROGRAM command 264
retrieval address (tag sort) 31
return code indicator 399

S

SDF statements 127
SDF, representation of syntax 126
SELECT-INPUT-RECORDS
example 325
SRT1 macro 363
SELECT-INPUT-RECORDS statement 66, 404
selection of sort criteria 287
selection sort 27, 288, 289
default fixed record length 329
example 332
see SORT-TYPE=COMPOUND-RECORD, example 329
variable file, example 329
SEND-MESSAGE-TO command 261
SET-RECORD-ATTRIBUTES 374
example, selection sort 329
SET-RECORD-ATTRIBUTES, RECA 404
SET-SORT-OPTIONS
SRT1 macro 363, 374
statement 152, 404
SHOW-JV command 400
SHOW-SORT-DEFAULTS operand
description 174
size of work files 106
SORT
and ACS (notes) 119
as a subroutine 206
in an XS environment 80
start 191
TU variant 191
SORT access method SORTZM 220
example 366, 371
SORT call at level 0 359
example 357
SORT call at level 1, example 361

sort criteria 287
sort field 38
  for variable-length records 40
  record length field 41, 118
sort field conversion 43
sort field overlap 42
SORT file assignment 92
SORT files 90
sort functions 24
sort key 38
SORT macros 212
sort run
  execution 287
  optimize 265
  status 399
SORT statement tables 408
SORT termination with error 398
sort types 24
SORT-RECORDS statement 263
SORT-TYPE=COMPOUND-RECORD
  example 329
  see selection sort, example 329
SORT-TYPE=TAG-TRAILER 337
SORT, access method SORTZM
  example 380
SORTCKPT 22
SORTIN 287, 305, 366, 371
sorting 17
sorting ISAM files in ascending order 308
sorting order
  change, FORMAT=EBCDIC-DIN,
    example 315
  change, MODIFY-CODE, example 318
SORTINxx 287
SORTLIB 191, 279, 288
SORTMODS 22, 92, 114
SORTOUT 287, 288, 305, 357, 366, 371
SORTU entry point 206
SORTWK 105
SORTWKEX 109
SORTWKx 105
SORTWKxx 108
SORTZM 220
  example 380
  SORT access method, example 366, 371
SORTZM access method 220
space allocation for work files 106
specify version 192, 200
SRT0 macro 212
  example 357
SRT1 macro 215
  example 361
  note, syntax 364
  syntax 364
SRTCLSE macro 229
  example 367
SRTGET macro 227
  example 367
SRTOPEN macro 223
  example 367
SRTPUT macro 225
  example 367
SRTXKERN 279
statement
  ADD-SYMBOLIC-NAMES 405
  ASSIGN-EXITS 404
  ASSIGN-FILES 23, 94, 296, 405
  ASSIGN-RESOURCES 153, 404
  MERGE-RECORDS 36, 263, 339, 404
  MODIFY-CODE 404
  representation of syntax 126
  SELECT-INPUT-RECORDS 66, 404
  SET-SORT-OPTIONS 152, 404
  SORT-RECORDS 263
  SUM-RECORDS 404
status of SORT run 399
STXIT 212, 407
STXIT facility 216
subcode1 (SC1) 202
subcode2 (SC2) 202
substitute characters 60
subsystem ACS 119
subtask 273
sum field 58
  extend, see overflow 325

SUM-RECORDS
example 325
prevent overflow 325
statement 404
summation
example 325
of records 79
supplying control information to SORT as a subroutine 403
SVB input block 210, 404
symbolic field name,example 313
syntax representation 126
syntax, SRT1 macro 364

T
tables, TRANSLATE-CHARACTER 51
tag records 30, 34
tag sort 29
example 337
retrieval address 31
tape auxiliary files 109
TERMINAL-ABNORMAL entry
user exits 237
transfer control area 212, 406
transfer indicator 406
TRANSLATE-CHARACTER
conversion tables 51
example 321
format 48, 321

U
Unicode 85
Unicode Default Collation Table 53
user exit
EXLST-FOR-INPUT 233, 251
EXLST-FOR-OUTPUT 233, 253
EXTERNAL-COMPARE 233, 258
INPUT 233, 239
INT 233, 261
NEUTRAL-TRANSLATE 233
OUTPUT 233, 244, 246, 345
PHYSICAL-TRANSLATE 233, 255
PLANNING 233, 238, 265
VIRTUAL-TRANSLATE 256, 353
user exits 233
note on 24/31-bit addressing 348

V
variable record format, sort fields 40
variable record length
data byte position 303
version selection, priorities 284
version specification 192, 200
virtual merging 267
VIRTUAL-TRANSLATE format 45
VIRTUAL-TRANSLATE user exit 256, 353

W
work files 105
size 106

X
XHCS (extended host code support) 81
XPG4 117