

All rights reserved. This document contains proprietary and confidential material, and is only for use by licensees of the SyncSort for z/OS proprietary software system.

P R O V E N
performance

SyncSort for z/OS

Programmer's Guide

Release 1.1

SI-4301-4
1.1c

© Syncsort Incorporated, 2003

All rights reserved. This document contains proprietary and confidential material, and is only for use by the licensees of the SyncSort proprietary software system. This publication may not be reproduced in whole or in part, in any form, except with written permission from Syncsort Incorporated.

SyncSort and Visual SyncSort are trademarks of Syncsort Incorporated. No claim is made to the exclusive right to use *Visual* apart from the mark as shown. All other company and product names used herein may be the trademarks of their respective companies.

Table of Contents

Chapter 1.	Introduction	1.1
	An Introduction to SyncSort for z/OS	1.1
	SyncSort's Basic Functions	1.1
	SyncSort's Data Utility and SortWriter Features	1.3
	Sample SortWriter Report	1.4
	Cultural Environment Support	1.5
	DB2 Query Support	1.5
	SyncSort's Operational Features	1.6
	SyncSort's Value-Added Products	1.6
	Structure of the Programmer's Guide	1.7
	Related Reading	1.8
	Online Message Help	1.9
Chapter 2.	SyncSort Control Statements	2.1
	Control Statement Summary Chart	2.2
	Disk Sort, MAXSORT, PARASORT, and Tape Sort Control Statement Requirements	2.5
	Data Utility Processing Sequence	2.6
	Control Statement Examples	2.8
	Rules for Control Statements	2.8
	ALTSEQ Control Statement	2.13
	END Control Statement	2.15
	INCLUDE/OMIT Control Statement	2.16
	INREC Control Statement	2.35
	MERGE Control Statement	2.37
	MODS Control Statement	2.54
	OMIT Control Statement	2.58
	OUTFIL Control Statement	2.59
	OUTREC Control Statement	2.88
	RECORD Control Statement	2.125
	SORT Control Statement	2.129
	SUM Control Statement	2.149
Chapter 3.	How to Use SyncSort's Data Utility Features	3.1
	Introduction	3.1
	Sample Data Utility Applications	3.2

	Selecting Input Records	3.2
	Selecting Relevant Fields from the Input Records	3.6
	Combining Records within a File	3.11
	Making Output Records Printable and Easy to Read	3.14
	Dividing a Report into Sections	3.30
	Writing Headers and Trailers for a Report	3.32
	Totaling and Subtotaling Data	3.41
	Obtaining Maximum, Minimum and Average Data	3.47
	Counting Data Records	3.49
	Creating Multiple Output Files	3.53
Chapter 4.	JCL and Sample JCL/Control Statement Streams.	4.1
	EXEC Statement	4.2
	For MAXSORT, PARASORT, DB2 Query Support, and Tape Sort	4.3
	Coding Conventions for DD Statements	4.3
	STEPLIB/JOBLIB DD Statement	4.4
	SYSOUT DD Statement	4.4
	SORTIN DD Statement	4.5
	SORTINnn or SORTINn DD Statement	4.7
	SORTOUT, SORTOFxx, SORTOFx and SORTXSUM DD Statements	4.8
	SORTWKxx or SORTWKx DD Statement	4.9
	SYSIN DD Statement	4.11
	\$ORTPARM DD Statement	4.11
	SORTCKPT DD Statement	4.14
	For Exit Routines that Require Link-editing at Execution Time	4.15
	DD Statements for MAXSORT, PARASORT, DB2 Query Support, and Tape Sort	4.16
	Sample JCL/Control Statement Streams	4.17
Chapter 5.	PARM Options	5.1
	Additional MAXSORT PARMs	5.1
	PARASORT PARM	5.2
	DB2 Query Support PARM	5.2
	Additional Tape Sort PARMs	5.2
	Precedence Rules	5.2
	PARM Option Summary Chart	5.2
	SyncSort PARM Options	5.7
	PARMs Accepted But Not Processed by Disk Sorts	5.34
Chapter 6.	Invoking SyncSort from a Program	6.1
	Programming Flexibility vs. Performance.	6.1
	DD Statements	6.1
	Invoking the Sort/Merge from an Assembler Program	6.2
	The 24-Bit Parameter List	6.4
	Sample Assembler Invocation Using 24-Bit Parameter List	6.11
	The 31-Bit Extended Parameter List	6.12
	Sample Assembler Invocation Using 31-Bit Parameter List	6.17
Chapter 7.	The Coding and Use of Exit Programs	7.1
	What Is an Exit?	7.1
	Loading the Exit Routines into Main Storage	7.3
	Exit Conventions	7.3
	Register Conventions	7.4
	The Exit Communication Area	7.4
	Exits E11, E21, and E31 - Preparing for Other Exit Routines	7.5
	Exit E32 - Invoked Merge Only: Creating Input Records	7.5
	Exits E14, E15, E25, and E35 - Deleting, Creating, Changing Records	7.7
	Exit E14 - Deleting, Summarizing, Changing Records	7.7

	Exit E15 - Creating, Revising or Analyzing the Input File	7.8
	Coding a COBOL E15 Exit Routine	7.10
	Example 1: Fixed-Length Records	7.11
	Example 2: Variable-Length Records	7.12
	Coding a C E15 Exit Routine	7.19
	Fixed-Length Records - Function Definition	7.20
	Variable-Length Records - Function Definition	7.21
	Exit E25 - Deleting, Changing, Summarizing Records	7.27
	Exit E35 - Adding, Deleting and Changing Records	7.28
	Coding a COBOL E35 Exit Routine	7.30
	Coding a C E35 Exit Routine	7.43
	Fixed-Length Records - Function Definition	7.43
	Variable-Length Records - Function Definition	7.45
	Exit E16-Taking Action on Insufficient Intermediate Storage	7.51
	Exits E17, E27, and E37 - Closing Data Sets	7.52
	Exits E18, E38, and E39 - Checking Labels, Processing Read or Write Errors, End-of-File Routines, Special VSAM Processing	7.52
	Exit E61 - Modifying the Collating Process	7.56
	Coding REXX Exits	7.58
Chapter 8.	The Flow of the Sort	8.1
Chapter 9.	MAXSORT.	9.1
	MAXSORT: A Maximum Capacity Sort	9.1
	MAXSORT's Advantages	9.4
	Job Control Language	9.4
	DD Statements	9.4
	SORTBKPT DD Statement	9.6
	SORTOU00 DD Statement	9.7
	SORTOU _n DD Statements	9.8
	Using Disk for Intermediate Output	9.9
	SORTCKPT DD Statement	9.9
	Control Statements	9.10
	PARM Options	9.10
	Exit Programs	9.13
	Invoking MAXSORT from a Program	9.13
	Restarting MAXSORT	9.14
	MAXSORT's Operator Interface	9.14
	Sample MAXSORT JCL/Control Streams	9.17
Chapter 10.	PARASORT.	10.1
	PARASORT: Parallel Input Processing for Elapsed Time Improvement	10.1
	PARASORT Applicability	10.1
	Job Control Language	10.2
	DD Statements	10.2
	SORTIN DD Statement with PARASORT	10.3
	SORTPAR _n DD Statements	10.5
	Special Channel Separated Esoteric Names	10.7
	Sortwork Considerations	10.8
	Operations Notes	10.9
Chapter 11.	SyncSort DB2 Query Support	11.1
	Restrictions	11.1
	Job Control Language	11.2
	DD Statements	11.2
	SORTDBIN DD Statement	11.3
	Operation	11.4
	Record Description	11.4

	Record Description: Trial Mode Execution	11.5
	Sample SyncSort DB2 Query Application	11.7
Chapter 12.	Tape Sort.	12.1
	When to Use Tape Sort	12.1
	EXEC Statement.	12.2
	DD Statements	12.3
	SORTLIB DD Statement	12.3
	SORTWKxx DD Statement	12.4
	\$ORTPARM DD Statement	12.5
	Optimizing Tape Sort	12.5
	Control Statements.	12.6
	Exit Programs	12.6
	Initiating Tape Sort Through JCL/Control Streams	12.7
	Invoking Tape Sort from a Program	12.9
Chapter 13.	Performance Considerations	13.1
	Disk Sort? MAXSORT? PARASORT? Tape Sort?	13.1
	JCL Sorts vs. Program-Invoked Sorts	13.2
	Control Statement Issues	13.2
	The Efficient Use of PARMs	13.3
	Optimizing System Resources	13.4
	Setting CORE	13.4
	The Incore Sort	13.6
	Disk Space Considerations.	13.7
	The Coding and Use of Checkpoint-Restart	13.10
	Automatic Checkpoint-Restart.	13.12
	Deferred Checkpoint-Restart	13.13
	Optimizing Data Set Placement	13.14
Chapter 14.	The HISTOGRM Utility Program	14.1
	What Is HISTOGRM?	14.1
	Using HISTOGRM to Determine L6 and L7 Values for SyncSort	14.2
	Control Parameters for HISTOGRM	14.2
	Job Control Language	14.4
	Executing HISTOGRM through an E15 Exit	14.4
	HISTOGRM Messages	14.9
Chapter 15.	Value-Added Products	15.1
	Visual SyncSort.	15.1
	SyncSort/COBOL Advantage	15.2
	PROC SYNCSORT - An Accelerator for SAS™ Sorting	15.3
	PipeSort	15.3
Chapter 16.	Messages	16.1
	SyncSort Statistical Record Facility Messages	16.65
	PROC SYNCSORT Messages	16.65
	License Key Messages	16.66
	Troubleshooting Abends.	16.70
	Index	I.1

SyncSort for z/OS Release 1.1 - Summary of Changes

SyncSort for z/OS is a new product for the IBM z/OS operating system and its underlying 64-bit z/Architecture. SyncSort for z/OS is the successor to SyncSort MVS in the same way that z/OS extends the capabilities of MVS.

Release 1.1 of SyncSort for z/OS was preceded by release 1.0. This Summary of Changes identifies changes for both releases as follows:

- Text without a change bar in the left margin applies to SyncSort for z/OS release 1.0 and identifies differences from SyncSort MVS 3.7.
- Text with a solid bar (|) in the left margin applies to SyncSort for z/OS 1.1 only and identifies differences from release 1.0.
- Text with a dashed bar (|) in the left margin applies to changes introduced in the 1.1C/TPF2 level of release 1.1.

Thus, if you are moving from SyncSort MVS 3.7 to SyncSort for z/OS 1.1, you should read the entire Summary of Changes. If you are moving from SyncSort for z/OS 1.0 to release 1.1, you need only read the text indicated by change bars. If you are moving from the TPF0 or TPF1 level to the TPF2 level of release 1.1, you need only read the text indicated by the dashed change bars.

Note that change bars are used in the Summary of Changes only.

Performance Improvements

SyncSort for z/OS exploits the advanced facilities of the zSeries architecture to achieve significant performance improvements:

- Algorithmic improvements have been made to exploit the z/Architecture enhancements of 64-bit central storage support and the elimination of expanded storage. These improvements apply to applications that would have formerly used hiperspace to exploit available expanded storage in a 31-bit OS/390 architecture. A new SyncSort technique, called ZSPACE, allows the native use of the central storage resources without incurring the CPU overhead associated with hiperspace simulation in a 64-bit z/Architecture environment. The technique reduces CPU time and elapsed time. The informational message WER418I indicates if ZSPACE is being used.
- Parallel access volume (PAV) technology, such as on IBM 2105 ESS (SHARK) devices and EMC Symmetrix devices, is exploited to reduce elapsed time.

- Dynamic Storage Management enhancements have been made to exploit the availability of expanded central storage resources found on zSeries servers. Optimization algorithms have been modified to employ additional storage resources when available. This change reduces sorting CPU time, EXCPs, and elapsed time for files that are larger than 600 megabytes.

DB2 Query Support

SyncSort can now directly retrieve data from a DB2 database based on a user-provided query. An SQL SELECT statement is used to specify the criteria of the request. The query of the DB2 database replaces SyncSort's SORTIN or E15 processing. The SORT or COPY functions, but not MERGE, can be used with DB2 queries. All SyncSort features performed after E15 processing are available for use with the DB2 query facility.

This feature improves performance over DB2's DSNTIAUL program by allowing DB2 data to be passed directly into a SORT or COPY operation, without the use of setup steps or the need for user-written E15 exits.

SORTWORK

The maximum number of SORTWK data sets, as specified on the DYNALLOC parameter of the SORT control statement or the DYNALLOC PARM option, has been increased from 100 to 255.

Data Utility Features

The SyncSort for z/OS data utility features have been enhanced with the following:

- INCLUDE/OMIT and OUTFIL INCLUDE/OMIT Statements
 - Fields can now be compared to the date of a SyncSort run or the date of the run with an offset. A variety of forms is available to represent the current date used in the comparison. This allows records to be included or omitted based on whether their dates are equal to, less than, or greater than the run date or the run date with an offset. The forms of the current date constants available for standard comparisons are &DATE_x, &DATE_x(c), &DATE_xP, and Y'DATE_x'.
 - Data fields of the formats Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z can now be compared to a year constant.
 - Data fields that represent the full-date formats Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y are now supported. Previously these data formats were available for use only with the SORT or MERGE control statement.
 - BI (binary) fields can now be compared to positive decimal numbers.

- SORT/MERGE Control Statements
 - The maximum length of an AC or AQ control field has been increased to 4091 bytes (2043 for variable-length records). This is raised from the prior limit of 256 bytes.
- INREC, OUTREC, and OUTFIL OUTREC Statements
 - The date and time of the SyncSort run can now be inserted in different forms in records by the parameters &DATE, &DATE_x, &DATE_x(c), &DATE_xP, &DATE=(m₁m₂m₃m₄), &DATENS=(xyz), &TIME, &TIME_x, &TIME_x(c), &TIME_xP, &TIME=(hp), and &TIMENS=(tt).
 - SMF date and time formats can now be converted to standard date and time formats. The SMF formats are DT1, DT2, DT3, TM1, TM2, TM3, and TM4.
 - The case of EBCDIC letters within a field can now be translated from uppercase to lowercase, from lowercase to uppercase, or the characters in a field can be translated according to an alternate collating sequence (ALTSEQ) table in effect. This is accomplished by using the subparameters TRAN=UTOL, TRAN=LTOU, and TRAN=ALTSEQ, respectively.
 - The conversion of the 2-digit year portion of full-date fields (Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y) to a 4-digit year in printable format is now supported.
 - A new format parameter, f_{y2b}(c), is now supported. Used with p (position) and l (length) in a p,l,f specification, the new format parameter allows conversion of a full-date field to a printable date with or without separator characters.
 - The new edit pattern M26 has been added.
- OUTFIL Control Statement
 - The VLFILL parameter has been added to the OUTFIL control statement. It is used in conjunction with OUTFIL OUTREC or OUTFIL OUTREC CONVERT to specify a fill byte to be used for any missing p,l field bytes. The VLFILL parameter has two functions:
 - It enables a variable-length OUTFIL OUTREC non-CONVERT application to continue processing when there is an input record with missing field bytes in a p,l field specification. If VLFILL has not been specified, the application will terminate with the critical error WER244A.
 - It provides a means to override the default fill byte used in an OUTFIL OUTREC CONVERT application when there are missing bytes in a p,l field specification. By default, spaces will be used for missing field bytes.

- The NULLOFL option has been added to the OUTFIL control statement. The NULLOFL option specifies the action to be taken when any non-SORTOUT OUTFIL data set contains no data records.
- The FTOV parameter has been added to the OUTFIL control statement. The FTOV parameter converts fixed-length input records to variable-length output records.
- The VLTRIM parameter has been added to the OUTFIL control statement. The VLTRIM parameter defines a byte to be deleted from the end of a variable-length record. All prior occurrences of this byte are deleted until a byte that is not equal to the trim byte is found.
- The REMOVECC parameter has been added to the OUTFIL control statement. The REMOVECC parameter generates reports that do not include ANSI carriage control characters that specify printer actions (for example, skipping a line, ejecting a page). REMOVECC omits the carriage control character from all of the report records.
- The &DATENS=(xyz) and &TIMENS=(tt) parameters, which provide additional formats for inserting the date and time of the SyncSort run in headers and trailers, have been added.

PARM Processing

The following are run-time options for sort, merge, or copy applications. The options do not apply to BetterGener.

Null SORTOUT

- The NULLOUT option has been added. The NULLOUT option specifies the action to be taken when SORTOUT contains no data records.

Record Padding and Truncation Control

- The PAD option has been added. The PAD option specifies the action to be taken if the LRECL defined in the JCL for a non-OUTFIL SORTOUT is larger than the SORTIN/SORTINnn LRECL or the internally processed record length when the SORTIN/SORTINnn LRECL is modified by features.
- The TRUNC option has been added. The TRUNC option specifies the action to be taken if the LRECL defined in the JCL for a non-OUTFIL SORTOUT is smaller than the

| SORTIN/SORTINnn LRECL or the internally processed record length when the SORTIN/SORTINnn LRECL is modified by features.

| ***Sum Processing***

- The OVFLO option has been added. The OVFLO option specifies the action to be taken if a summary field overflows or underflows during SUM processing.

| ***Record Validity Checking***

- The OFF4 subparameter has been added to the VLTEST option. OFF4 specifies the action to be taken if an illogical variable-length record segment is found.

Visual SyncSort for z/OS

SyncSort for z/OS incorporates functionality to integrate Visual SyncSort with SyncSort for z/OS mainframe processing. Visual SyncSort is a separately available PC product that is designed to allow programmers and non-programmers alike to easily create and manage SyncSort applications for the mainframe environment. With Visual SyncSort, you can create new sort, merge, and copy applications, or you can import and modify existing ones. Visual SyncSort saves programmer time while taking full advantage of the processing power of SyncSort for z/OS.

Messages

- Message WER219I has been modified to include an SMS return code provided by SMS in the event of DYNALLOC failure.
- Message WER418I has been modified to indicate whether SyncSort has dynamically chosen to use DataSpace, ZSPACE, or hiperspace during the execution of a sort. See the "Performance Improvements" section above for a description of ZSPACE.
- Message WER456I indicates that a file describing your application has been created and written to the VISUALEX DD statement for export to the PC component of Visual SyncSort.
- Message WER457A indicates that the VISUALEX DD statement for export to the PC component of Visual SyncSort is either missing or its data set has been incorrectly defined.
- Message WER458A indicates that the SYSIN data set created by the PC component of Visual SyncSort cannot be processed by SyncSort due to an insufficient level of maintenance on the SyncSort library.
- Message WER459A indicates that only qualified SyncSort applications may be exported to Visual SyncSort for reasons supplied in the message text.

- Messages WER461I, WER462I, and WER462A support the run-time options PAD, TRUNC, and OVFL0.
- Message WER463A indicates that a linear VSAM data set has been specified on input or output. This type of data set is not supported.
- Message WER464I indicates that an invalid spanned record segment has been found and the new OFF4 subparameter of VLTEST is in effect.
- Message WER467I indicates that a report of the record layout produced by the DB2 query contained in the SORTDBIN data set has been successfully produced.
- Message WER468A indicates that the DB2 query operation failed and the sort or copy application will not execute. The message text indicates the condition that caused the failure or the DB2 query requirement that was violated.

Chapter 1. Introduction

An Introduction to SyncSort for z/OS

SyncSort for z/OS is a high performance sort/merge/copy utility. It is designed for the advanced facilities of the zSeries architecture, but also supports the system architectures of IBM System/390 and compatible computers. It exploits the features of the z/OS operating system but will also execute under OS/390.

SyncSort is designed to conserve system resources, provide significant performance benefits, and operate efficiently in 31-bit or 64-bit environments.

SyncSort can be initiated through job control language or invoked from a program written in COBOL, PL/1, or Assembler language. A JCL-initiated sort is more efficient because SyncSort totally controls the sort execution, including I/O management and main storage management. Exit routines may be written in COBOL, C, FORTRAN, REXX, or Assembler language to give a JCL sort additional programming flexibility. Exits may also be in PL/1 when SyncSort is invoked by a PL/1 program.

SyncSort's Basic Functions

SyncSort has three basic functions:

- Sorting - rearranging data set records to produce a specific sequence.
- Merging - combining up to 100 pre-sequenced data sets into one data set which has the same sequence.

- Copying - reproducing a data set *without* going through the sorting process.

Sorting

A sort rearranges the records in a data set to produce a specific sequence, e.g., chronological or alphabetic order. SyncSort provides four sorting techniques:

- Disk Sort, the standard sorting technique. Information in the *Programmer's Guide* refers to the Disk Sort unless otherwise indicated.
- MAXSORT, a maximum capacity sorting technique with an enhanced breakpoint/restart capability. MAXSORT can sort any collection of data - regardless of size - using a limited amount of disk space. MAXSORT is described in the MAXSORT chapter of this guide.
- PARASORT, a sorting technique that significantly reduces elapsed time for sorts whose input is a multi-volume tape data set and/or concatenated tape data sets. PARASORT improves performance by using multiple tape drives in parallel. PARASORT is described in the PARASORT chapter of this guide.
- Tape Sort, an alternative sorting technique used when intermediate storage must be assigned to tape. The Tape Sort is described in the Tape Sort chapter of this guide.

A sort logically consists of four phases that perform the following functions:

- The control statements and JCL information are read and analyzed and the operational parameters for the sort are established.
- The input data is read into main storage and sorted.
- If necessary, intermediate results are written to temporary storage devices.
- The sorting process completes and the sorted data is written to the specified output device(s).

Merging

A merge combines up to 100 pre-sequenced data sets into one data set which has the same sequence. A merge has two phases that perform these functions:

- The control statements and JCL information are read and analyzed and the operational parameters for the merge are established.
- The files are merged and the merged data is written to the specified output device(s).

Copying

A copy reproduces a file, completely bypassing the sorting process. A copy has two phases that perform these functions:

- The control statements and JCL information are read and analyzed and the operational parameters for the copy are established.
- The copied file is written to the specified output device(s).

SyncSort's Data Utility and SortWriter Features

SyncSort is designed to improve programmer productivity by reducing the time the programmer/analyst must spend designing, testing, and debugging applications. With SyncSort's extensive Data Utility and SortWriter features, data processing applications previously requiring several steps can be accomplished in a *single* execution.

SyncSort's Data Utility features include a multiple output facility, a full range of report writing capabilities, and record selection and record reformatting facilities. These options allow the user to design sort/merge/copy applications that can accomplish a host of related tasks.

Generating Multiple Output

The multiple output facility (OUTFIL) allows multiple output files to be generated with just one pass of the sort. Each of these files can have unique specifications that determine which records are to be included, how the records are to be formatted, and which report capabilities are to be used. Moreover, all these files can be written to the same output device, or each can be written to a different device.

Creating Reports

SyncSort's SortWriter feature (OUTFIL) allows the user to design comprehensive reports easily and efficiently. SortWriter options allow output data to be flexibly formatted with headers and trailers, which can include data fields. Various kinds of numeric results can be produced at report, page, and section levels. These include totals, subtotals, minimums, subminimums, maximums, submaximums, averages, subaverages, record counts, and subcounts. Output record fields can be realigned; the records can be padded with blanks, characters, and binary zeros; and numeric data can be converted and edited. Automatic pagination, page numbering, and dating are also provided.

Selecting Records, Reformatting Records, and Summarizing Fields

Record selection, record reformatting, and summing are other important SyncSort Data Utility features. Record selection via the INCLUDE/OMIT feature permits certain records to be included or omitted from an input data set based on comparisons between two data

fields or between a data field and a constant. Date data formats work with the CENTWIN option to ensure that century evaluation is applied to INCLUDE/OMIT comparisons involving 2-digit year data.

Record reformatting after input and/or before output, provided by the INREC/OUTREC capability, allows the user to delete or repeat portions of records; insert spaces, characters, binary zeros, date constants and sequence numbers; realign fields; convert numeric data to its printable format; and convert data to its printable hexadecimal format. The CENTWIN option and date data formats enable conversion of 2-digit year fields to printable or packed decimal 4-digit years of the appropriate century. The ability to delete irrelevant fields before sorting via INREC can provide important performance benefits. Additionally, the OUTREC facility can be used to convert a variable-length record format input file into a fixed-length format output file or to convert a fixed-length record format input file into a variable-length format output file.

The SUM feature allows records with equal sort control fields to be deleted and optionally summarizes numeric fields on those records. The deleted records can optionally be written to a separate data set.

Sample SortWriter Report

The report below illustrates the versatility of SyncSort's Data Utility and SortWriter features. First, irrelevant records are omitted from the input file and the input record is reformatted to eliminate unnecessary data fields. Then the file is sorted by invoice status and invoice date. The output record is reformatted for readability and the numeric fields are converted and edited. The report itself is divided into sections and subsections based on control field breaks. Headers and trailers identify the data fields, provide record counts and section and cumulative totals, and include the date and page number.

INVOICE STATUS: 0

COMPANY NAME	ADDRESS	CO #	INV #	INV DATE	INVOICE		BALANCE	
					PRODUCT	TAX	PRODUCT	TAX
REPUBLIC DATA	NYC NY	2681	86013306	1/17/91	1,100.00	90.75	1,100.00	90.75
RICE FEATURES	CHI IL	2244	86013298	1/17/91	1,500.00	75.00	1,500.00	75.00
SIDNEY COLLEGE	HOU TX	4762	86013297	1/17/91	2,500.00	150.00	2,500.00	150.00
WINIFRED INDUST	WAS DC	1177	86013299	1/17/91	650.00	26.00	650.00	26.00
PIZZUTO LOANS	STL MO	4633	86022200	2/15/91	550.00	22.00	550.00	22.00
RICE FEATURES	CHI IL	2244	86022198	2/15/91	1,500.00	75.00	1,500.00	75.00
SIDNEY COLLEGE	HOU TX	4762	86022197	2/15/91	500.00	30.00	500.00	30.00
REGENCY TRUST CO	BOS MA	4986	85124011	12/15/91	1,500.00	75.00	1,500.00	75.00
SIDNEY COLLEGE	HOU TX	4762	85124016	12/15/91	5,000.00	300.00	5,000.00	300.00
TOTAL NUMBER OF INVOICES:		11	MONTHLY TOTALS:		\$22,850.00	\$1,484.50	\$22,850.00	\$1,484.50
BALTIC AVENUE CORP	CLE OH	0636	86022207	2/15/91	650.00	29.25	650.00	29.25
FASTEROOT EQUIP	BAL MD	4980	86022205	2/15/91	1,700.00	76.50	1,700.00	76.50
FEDERAL FABRICS	SHV LA	5143	86022204	2/15/91	1,750.00	70.00	1,750.00	70.00
PATIO PRODUCTS	MRY CA	3029	86022203	2/15/91	850.00	51.00	850.00	51.00
TURENIUS FOR. EXCH.	DTT MI	8325	86022201	2/15/91	1,600.00	64.00	1,600.00	64.00
WINES ASSOCIATES	SMF CA	1794	86022209	2/15/91	750.00	45.00	750.00	45.00
DESIGN TECHNOLOGIES	LAX CA	2520	85124017	12/15/91	360.00	21.60	360.00	21.60
POLL DATA CORP	LAX CA	0846	85124019	12/15/91	600.00	36.00	600.00	36.00
TOTAL NUMBER OF INVOICES:		8	MONTHLY TOTALS:		\$8,260.00	\$393.35	\$8,260.00	\$393.35

Figure 1. Sample SortWriter Report

Cultural Environment Support

Cultural environment support allows you to choose an alternate set of sort collating rules based on a specified national language. The alternate collating applies to SORT/MERGE and INCLUDE/OMIT processing.

SyncSort employs the callable services of IBM's Language Environment for z/OS to collate data in a way that conforms to the language and conventions of a selected locale. A locale defines single and multi-character collating rules for a cultural environment. Numerous pre-defined locales are available.

For additional information, see "LOCALE" on page 5.20.

DB2 Query Support

SyncSort can directly retrieve data from a DB2 database based upon a user provided query. An SQL SELECT statement is used to specify the criteria of the request, and the query of the DB2 database will be in place of SyncSort's SORTIN or E15 processing. SORT or COPY, but not MERGE, functions can be used with DB2 queries. All SyncSort features that are performed after E15 processing are available for use with the DB2 query facility.

This feature improves performance over DB2's DSNTIAUL program by allowing DB2 data to be passed directly into a SORT or COPY operation, without the use of setup steps or the need for user-written E15 exits. Refer to "Chapter 11. SyncSort DB2 Query Support" for more information.

SyncSort's Operational Features

SyncSort will take advantage of data space and hiperspace to further improve performance. A portion of the address space may be allocated for SyncSort's ZSPACE technique. This technique was created as a replacement for hiperspace. It allows native use of the available central storage resources. This technique eliminates the additional overhead produced when hiperspace is simulated by the z/OS operating system in a z/Architecture environment. It provides superior CPU performance and reduced system overhead compared to a conventional hiperspace application.

SyncSort can also interact with exits and invoking programs such as VS COBOL II, COBOL/370, C370 V2R1 with V2R2 C370 library, SAA AD/Cycle C370 Release 2, and IBM C/C++ V3R2 programs.

SyncSort's PARMEXIT feature permits the dynamic modification of PARM values based on the conditions at execution time. This feature facilitates the passing of additional parameters to specific jobs.

Other operational features include resident, reentrant code, interactive and streamlined installation and maintenance procedures; automatic release or secondary allocation of direct access intermediate storage (SORTWK) and output (SORTOUT) space *without* JCL specification; dynamic allocation of SORTWK space under z/OS (DYNALLOC); and automatic incore sorting.

SyncSort's Value-Added Products

Value-added products available from Syncsort can significantly improve sorting efficiency:

Visual SyncSort for z/OS is a PC-based product designed to allow programmers and non-programmers alike to easily create and manage SyncSort for z/OS applications in the mainframe environment. With Visual SyncSort, you can create new sort, merge, and copy applications, or you can import and modify existing ones. Visual SyncSort saves programmer time while taking full advantage of the mainframe processing power of SyncSort for z/OS.

SyncSort/COBOL Advantage is a fully automatic product which improves the performance of COBOL programs that invoke SyncSort. The COBOL Advantage improves elapsed time by 25 to 40% by enhancing processing of sequential files, including INPUT and OUTPUT procedure files.

PROC SYNCSORT - An Accelerator for SAS® Sorting is a high performance, transparent replacement for the SAS procedure PROC SORT. Compared to PROC SORT, PROC

SYNCSORT reduces the resources required for sorting within SAS applications and cuts sort elapsed time.

PipeSort enables SyncSort to run multiple sorts simultaneously of the same input data. For large input files, PipeSort significantly reduces total elapsed time compared to running separate sort jobs.

For more detailed information regarding each of these products, see “Chapter 15. Value-Added Products”.

Structure of the Programmer’s Guide

The *SyncSort for z/OS Programmer’s Guide* is a reference manual designed for applications programmers who are using SyncSort to sort, merge, or copy sequential data sets. This manual is self-contained and assumes only a basic working knowledge of the operating system and its job control language. It should not be necessary to refer to any other manual to produce an efficient sort.

SyncSort Control Statements describes how to specify and use the SORT/MERGE, INCLUDE/OMIT, INREC/OUTREC, OUTFIL, RECORD, MODS, SUM, ALTSEQ, and END statements. The discussion of a particular control statement includes these topics: the statement’s syntax format, the versatility provided by the various parameters (many of which are unique to SyncSort), and the interaction between the control statement and other statements.

How to Use SyncSort’s Data Utility Features explains and illustrates the Data Utility and SortWriter features through a series of sample applications. Each application is self-contained and provides instructions for specifying both the required JCL and the appropriate control statements.

JCL and Sample JCL/Control Statement Streams analyzes SyncSort’s job control requirements and describes the SyncSort DD statements, each of which is illustrated with an example. JCL and control statement streams for MAXSORT and PARASORT are also described. Numerous examples are provided.

PARM Options describes the operational parameters of SyncSort and identifies the delivered defaults. This chapter explains how to specify such features as dynamic allocation of SORTWK space under z/OS, automatic secondary allocation and release of SORTWK space, the ability to skip a certain number of records or stop after sorting a certain number of records, and message routing.

Invoking SyncSort from a Program describes SyncSort invocation through assembler programs using 24-bit and 31-bit parameter lists. Numerous examples are provided.

The Coding and Use of Exit Programs indicates at which points during sort processing user-written exit routines can be executed. Each exit point is fully documented together

with the appropriate tasks. Examples of COBOL E15 and E35 exit routines for fixed and variable-length records are included.

The Flow of the Sort provides a skeletal view of the flow of control in the standard Disk Sort (including the incore sort), merge and copy. This chapter indicates the order in which the control statements and exit routines are processed, information which is particularly useful at the design stage of an application.

MAXSORT explains when MAXSORT should be used, describes its JCL requirements, control statements and PARM options, and provide examples. The chapter also examines MAXSORT's restart capability and its operator interface.

PARASORT explains the elapsed time advantages of the technique, the type of applications where it can be applied, and the JCL requirements.

SyncSort DB2 Query Support explains how SyncSort can improve performance by allowing DB2 data to be passed directly into a SORT or COPY operation without the use of setup steps or user-written E15 exits.

Tape Sort describes the SyncSort DD statements needed for a tape sort and how to initiate a tape sort from JCL or a program.

Performance Considerations describes how to design the most efficient application. It contrasts the merits of Disk Sort, PARASORT, MAXSORT and Tape Sort, JCL and invoked sorts, the incore sort, and standard SORTWK techniques. Formulas for calculating main storage and SORTWK requirements are provided. Other topics include the efficient use of control statements and PARMs, tuning main storage and SORTWK allocations and the use of the Checkpoint-Restart feature.

The HISTOGRM Utility Program describes how to use the HISTOGRM program to report on the composition of variable-length files. This program indicates the average record length, byte total, record total, block count and record count. Job control requirements, control statements and messages are outlined. Sample job streams illustrate how to run HISTOGRM as a separate job and as an E15 exit during a variable-length sort.

Value-Added Products describes Visual SyncSort for z/OS, SyncSort/COBOL Advantage, PROC SYNCSORT-An Accelerator for SAS[®] Sorting, and PipeSort. This chapter also provides detailed information regarding their functions and special features.

Messages documents all of the WERnnnx messages generated by the SyncSort program. This chapter includes sections describing "Troubleshooting with WER999A UNSUCCESSFUL SORT" and "What to Do before Calling SyncSort for z/OS Product Services."

Related Reading

The following guides supplement the information provided in the *Programmer's Guide*.

Installation Guide

This manual explains how to install and maintain SyncSort and defines the default options.

Reference Guide.

This handbook, intended for quick reference, provides the syntax for SyncSort control statements and briefly describes each parameter.

Exploiting SyncSort: SortWriter Data Utilities Guide.

This two-part user's guide demonstrates how SyncSort's versatile Data Utility features provide an efficient, one-step alternative to writing, testing and debugging programs. Five comprehensive sample applications illustrate how the control statements work together to produce formatted reports.

Exploiting SyncSort: MAXSORT.

This user's guide explains how to use the special MAXSORT feature of SyncSort to sort very large amounts of data with only a limited amount of disk space. MAXSORT's unique restart capability is described and sample job control streams and tuning information are included.

Online Message Help

All SyncSort messages and their explanations can be accessed online through an ISPF/PDF dialog. Contact your system administrator for information about the operation of the message help facility.

Chapter 2. SyncSort Control Statements

The control statements tell SyncSort for z/OS how to process files. There are 12 control statements:

Control Statement	Function
ALTSEQ	Specifies an alternate collating sequence for control fields with an AQ format.
END	Signals the end of control statements.
INCLUDE	Specifies the criteria which determine whether or not records are included in an application.
INREC	Reformats the input record before sort/merge processing.
MERGE	Defines a merge or copy application and specifies merge control fields.
MODS	Specifies user exit(s).
OMIT	Specifies the criteria which determine whether or not records are omitted from an application.
OUTFIL	Describes the output file(s) and specifies SortWriter and processing options.
OUTREC	Reformats the output record after sort/merge processing.

RECORD	Provides record information at various processing stages.
SORT	Defines a sort or copy application and specifies sort control fields.
SUM	Deletes records with equal control fields and summarizes numeric fields on those records.

Control Statement Summary Chart

The following table summarizes the parameters of each control statement and indicates default values.

Control Statement Name	Parameters	Delivered Default
ALTSEQ	CODE=(ccpp ₁ [,ccpp ₂]...)	Standard EBCDIC series
END		
INCLUDE	COND= $\left[\begin{array}{l} \text{ALL} \\ \text{(comparisons)} \\ \text{NONE} \end{array} \right] \text{ [,FORMAT=f]}$	Sort/Merge all records
INREC	FIELDS=(field ₁ [,field ₂]...)	Input records unchanged
MERGE	$\left\{ \begin{array}{l} \text{FIELDS}=(p_1,l_1,f_1,o_1 [,p_2,l_2,f_2,o_2]...) \\ \text{FIELDS}=(p_1,l_1,f_1,o_1 [,p_2,l_2,f_2,o_2]...),\text{FORMAT}=f \\ \text{FIELDS}=\text{COPY} \end{array} \right\}$	
	$\left[\text{,CENTWIN}=\left\{ \begin{array}{l} s \\ f \end{array} \right\} \right]$	Century window starts with current year
	$\left[\begin{array}{l} \text{,CKPT} \\ \text{,CHKPT} \end{array} \right]$	No checkpoint
	$\left[\begin{array}{l} \text{,EQUALS} \\ \text{,NOEQUALS} \end{array} \right]$	NOEQUALS
	$\left[\text{,FILES}=n \right]$	
	$\left[\text{,SKIPREC}=n \right]$	Copy all records
	$\left[\text{,STOPAFT}=n \right]$	Copy all records

Table 1. (Page 1 of 4) Control Statement Summary Chart

Control Statement Name	Parameters	Delivered Default
MODS	exit-name ₁ =(r ₁ ,b ₁ [,d ₁] $\left[\begin{array}{c} \{,N\} \\ \{,S\} \\ \{,C\} \\ \{,E\} \\ \{,X\} \\ \{,T\} \end{array} \right]), \dots, \text{exit-name}_{16}=(\dots)$	No exits
OMIT	COND= $\left[\begin{array}{c} \{ALL \\ (comparisons) \\ NONE\} \end{array} \right] [,FORMAT=f]$	Sort/Merge all records
OUTFIL	[FILES=(fileid ₁ [,fileid ₂]...)]	One output file
	$\left[,FNAMES= \left\{ \begin{array}{c} \text{ddname} \\ (\text{ddname}_1 \text{ [,ddname}_2] \dots) \end{array} \right\} \right]$	Output defined by FILES
	[,HEADER1=(field ₁ [,field ₂]...)]	No report heading
	[,HEADER2=(field ₁ [,field ₂]...)]	No page headings
	$\left[\left\{ \begin{array}{c} ,INCLUDE \\ ,OMIT \end{array} \right\} = \left\{ \begin{array}{c} ALL \\ (comparisons) \\ NONE \end{array} \right\} \right]$	Output all records
	$\left[,LINES= \left\{ \begin{array}{c} n \\ ANSI \\ (ANSI,n) \end{array} \right\} \right]$	60 (if report-writing parameters)
	[,NODETAIL]	Detailed report
	[,REMOVECC]	Produce a report with ANSI control characters
	$\left[,NULLOFL= \left\{ \begin{array}{c} RC0 \\ RC4 \\ RC16 \end{array} \right\} \right]$	Return code of zero
[,STARTREC=n]	Start processing with first record	

Table 1. (Page 2 of 4) Control Statement Summary Chart

Control Statement Name	Parameters	Delivered Default
OUTFIL	[,ENDREC=n]	End processing with last record
	[,SAVE]	Omitted records not saved for output
	[OUTREC=(field ₁ [,field ₂]...)]	Record unchanged
	[,CONVERT]	Record format unchanged
	[,VLFILL=f]	Missing fields will be filled with blanks (x'40') when CONVERT option in use
	[,FTOV]	Output record format the same as input
	[,VLTRIM=b]	Retain all trailing bytes
	[,SPLIT]	No split output
	[,SECTIONS=(field ₁ [,field ₂]...)]	No sections
	[,TRAILER1=(field ₁ [,field ₂]...)]	No report trailer
	[,TRAILER2=(field ₁ [,field ₂]...)]	No page trailers
OUTREC	[,FIELDS=(field ₁ [,field ₂]...)]	Record format unchanged
	[,CONVERT]	Output record format the same as input

Table 1. (Page 3 of 4) Control Statement Summary Chart

Control Statement Name	Parameters	Delivered Default
RECORD	[TYPE=F V]	
	[,LENGTH=(l ₁ ,...,l ₇)]	
SORT	$\left\{ \begin{array}{l} \text{FIELDS}=(p_1,l_1,o_1 [p_2,l_2,o_2]...) \\ \text{FIELDS}=(p_1,l_1,o_1 [p_2,l_2,o_2]...),\text{FORMAT}=f \\ \text{FIELDS}=\text{COPY} \end{array} \right\}$	
SORT	[,CENTWIN= $\left\{ \begin{array}{l} s \\ f \end{array} \right\}$]	Century window starts at current year
	[,CKPT ,CHKPT]	No checkpoint
	[,DYNALLOC=d/(d,n)/OFF]	No dynamic allocation
	[,EQUALS ,NOEQUALS]	NOEQUALS
	[,FILSZ=n] [,SIZE=n]	
	[,SKIPREC=n]	Sort or copy all records
	[,STOPAFT=n]	Sort or copy all records
SUM	$\left\{ \begin{array}{l} \text{FIELDS}=(p_1,l_1,f_1 [p_2,l_2,f_2]...) \\ \text{FIELDS}=(p_1,l_1 [p_2,l_2]...),\text{FORMAT}=f \\ \text{FIELDS}=\text{NONE} \end{array} \right\}$	No summary of fields; no reduction of equal-keyed records
	[,XSUM]	

Table 1. (Page 4 of 4) Control Statement Summary Chart

Disk Sort, MAXSORT, PARASORT, and Tape Sort Control Statement Requirements

The following table summarizes control statement usage for Disk Sort, MAXSORT, PARASORT, and Tape Sort.

Control Statement	Disk Sort	MAXSORT and PARASORT	Tape Sort
ALTSEQ	Optional	Optional	Not supported
END	Required if exits included in input stream	Required for MAXSORT if exits included in input stream; optional for PARASORT	Required if exits included in input stream
INCLUDE/ OMIT	Optional	Optional	Not supported
INREC	Optional	Optional	Not supported
MERGE	Required for merge or copy	Not applicable	Required for merge; copy not supported
MODS	Required for exits	Required for exits	Required for exits; not supported if program-invoked
OUTFIL	Required for multiple output or reports	Not supported for MAXSORT; optional for PARASORT	Not supported
OUTREC	Optional	Optional	Not supported
RECORD	Conditionally required	Conditionally required	Conditionally required
SORT	Required for sort or copy	Required for sort; copy not supported	Required for sort; copy not supported
SUM	Optional; not applicable to copy	Optional	Not supported

Table 2. Control Statement Usage for Disk Sort, MAXSORT, PARASORT, and Tape Sort

Data Utility Processing Sequence

The following figure illustrates the sequence in which SyncSort control statements and parameters are processed. It includes those control statements and parameters that modify the input file (e.g., INCLUDE/OMIT), reposition record fields (e.g., INREC, OUTREC), and create reports (e.g., OUTFIL).

When specifying record fields on any of these SyncSort control statements or parameters, refer to the record as it appears at that stage of SyncSort processing. For example, when specifying SORT fields be sure to take into account any repositioning of fields that may be due to INREC processing.

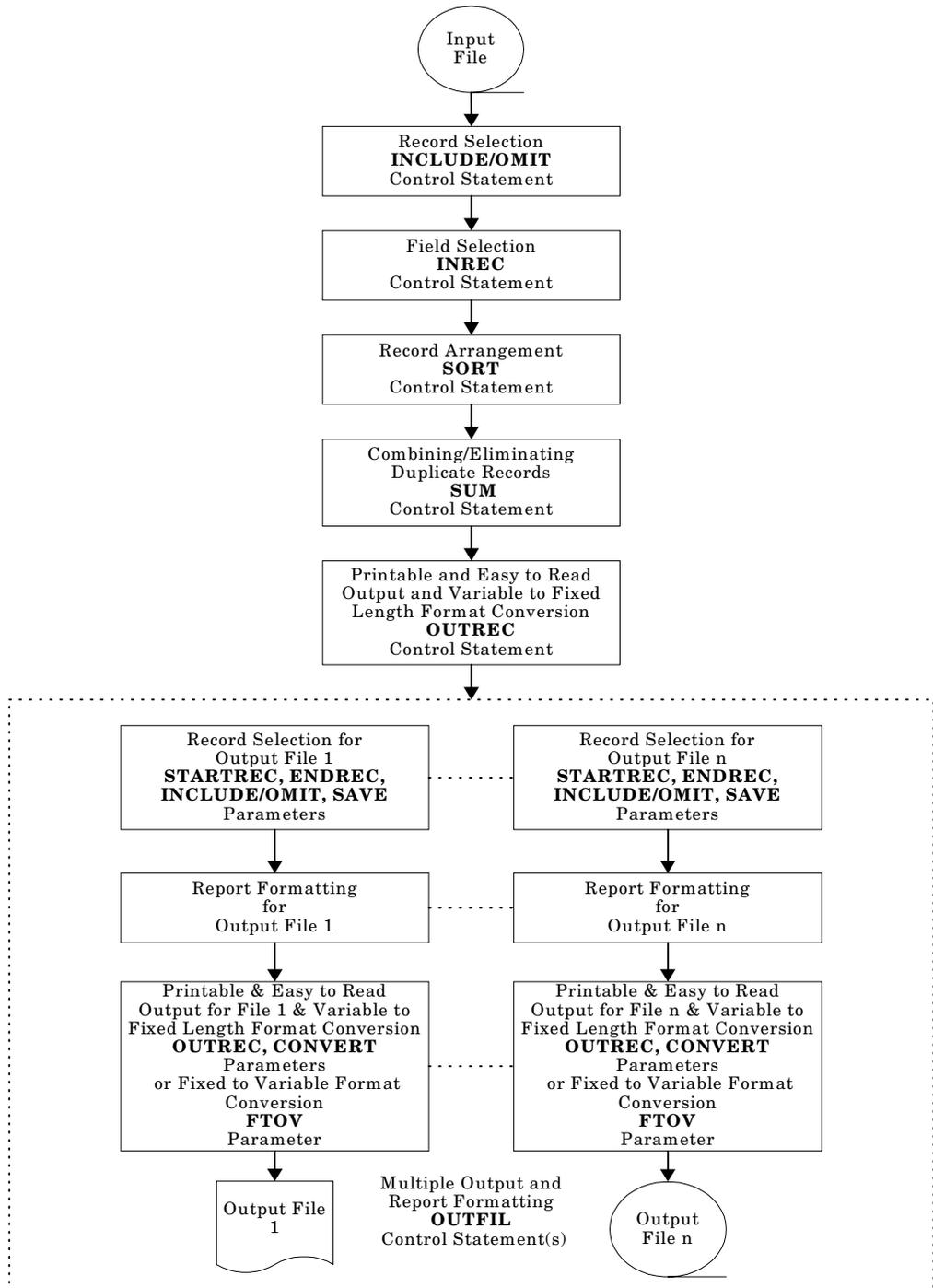


Figure 2. Data Utility Processing Sequence

Control Statement Examples

Simple examples illustrating the syntax of each of the SyncSort for z/OS control statements are included in this chapter. More complex applications are presented in “Chapter 3. How to Use SyncSort’s Data Utility Features”. These applications demonstrate how the INCLUDE/OMIT, INREC, OUTREC, SUM, and OUTFIL control statements can be used to accomplish a variety of tasks, such as selecting input records, selecting input fields, combining records, reformatting output records, writing reports, and creating multiple output.

Rules for Control Statements

The following rules apply to SyncSort for z/OS control statements.

Specifying Control Statements

- Control statements can be in any order, except for the END control statement which, if specified, must be last.
- Each control statement, except for OUTFIL, can be specified only once for a particular application.
- The control statement can begin in column 2 through column 69. If labels are used, the control statement must be separated from the label by at least one blank.
- The control statement name must be the first field (or the first field after a label) of the first card image of the control statement. It cannot be continued on a continuation card image.
- The last operand of each control statement must be followed by at least one blank.

Specifying Parameters

- Parameters can take three forms:
 - Parameter
 - Parameter=value
Parameter=(value)
Parameter(value)
 - Parameter=(value₁,value₂,...,value_n)
Parameter(value₁,value₂,...,value_n)

Note that multiple values must be enclosed in parentheses.

- Parameters can be in any order, but if parameters are present, the first parameter must begin on the first card image of a control statement.
- Parameters must be separated from each other by commas.
- The parameter(s) must be preceded and followed by at least one blank. A blank separates the parameter(s) from the control statement name and also indicates the end of the control statement.
- If the parameter(s) end in column 71, column 72 must contain a blank to signal the end of the control statement.
- With the exception of literal strings and constants, a parameter value cannot exceed 28 alphanumeric characters. Parameter values cannot include commas, equal signs, or parentheses.
- With the exception of literal strings specified as parameter values, blanks are not permitted within parameters.

Specifying Field Positions, Lengths, and Formats

- Control statements reference fields by position p and length l .
- The first byte of every fixed-length record is position 1, the second byte position 2, and so on.
- Bytes 1 through 4 of variable-length records are reserved for the Record Descriptor Word (RDW). For these records, the first byte of the data portion is position 5.
- Some control statements support bit-level processing. This means a binary control field can begin and end on *any* bit of *any* byte. The 8 bits in each byte are numbered 0 through 7. For example, a position value of 7.4 designates a field beginning on the fifth bit of the seventh byte. A length value of 7.4 designates a field 7 bytes, 4 bits long.
- Make sure the position value takes into account any record reformatting and data conversion that may have resulted from SyncSort data utility processing or exit programs. Refer to the *Data Utility Processing Sequence* figure at the beginning of this chapter and to “Chapter 8. The Flow of the Sort”.
- When proper processing depends on data format, the format of the field must be specified.
- The format of the field must be appropriate to the task. For example, only numeric fields can be SUMmed.
- When all the fields have the same format, the format value can be specified just once through the FORMAT=f subparameter. The FORMAT=f subparameter *cannot* be used when the INCLUDE/OMIT parameter is specified on the OUTFIL control statement.

Specifying Comments

- Identify a comment card image by placing an asterisk (*) in column 1. Comments can extend through column 80.
- To add a comment to a control statement card image, leave one or more blanks after the last parameter or comma on the image and follow with the comment, which can extend through column 71.
- Continue a comment that follows a control statement by coding an asterisk (*) in column 1 of the next card image or, if the control statement had ended, by placing a continuation character in column 72.
- Comment lines can be inserted between a control statement and its continuation by coding an asterisk (*) in column one.

Specifying Continuation Card Images

Control statements cannot extend beyond column 71, but they can be continued. To continue a control statement:

- Break after a parameter-comma or parameter-colon combination before column 72. Begin the continuation of the next card image anywhere between columns 2 and 71 if there is no label on the continuation card. If there *is* a label, begin the continuation card in any column from 3-71. No continuation character is required.

--or--

- When the control statement extends through column 71 and cannot be broken at a parameter-comma or parameter-colon combination:
 - If the control statement *does not* contain a literal string that would extend beyond column 71, place a continuation character in column 72 and continue the control statement on the next card image anywhere between columns 2 and 71.
 - If the control statement *does* contain a literal string that would extend beyond column 71, place a continuation character in column 72 and begin the continuation of the literal string in column 16 of the next card image.

The following examples illustrate how card images can be continued.

	COL. 72 ↓
SORT FIELDS=(1,10,A,20,5,A,45,7,A),FORMAT=CH,STOPAFT=100, EQUALS	

Figure 3. Continuing a Control Statement Without Specifying a Continuation Character

In the above example, no continuation character is required. The control statement is interrupted after a parameter-comma combination before column 72.

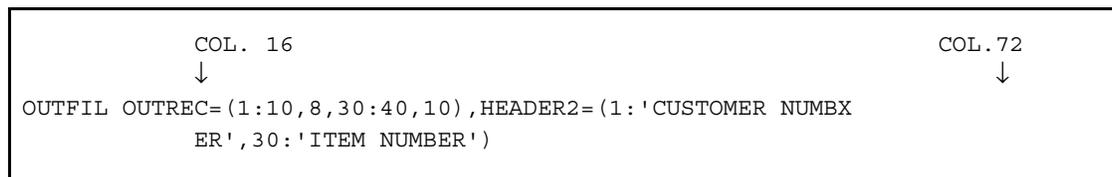


Figure 4. Continuing a Control Statement with a Continuation Character

In this example, a continuation character is necessary because the literal string in the HEADER2 specification would extend beyond column 71. The 'X' in column 72 is the continuation character. The literal string is continued in column 16 of the next card image.

Specifying Labels

SyncSort for z/OS supports labels. If labels are used, the following rules apply:

- Labels are permitted on all SYSIN control statements, including continuation card images, but not on the control statements passed by an invoking program or the \$ORTPARM DD statement.
- Labels must begin in column 1 with an alphabetic character.
- Labels can be any length, provided the other rules which apply to control statements are followed.
- At least one blank must separate the label from the control statement name or parameter that follows it.

Notational Conventions Used in the SyncSort for z/OS Programmer's Guide

- Braces indicate that a choice must be made from the alternatives listed.
- Brackets indicate an optional item. Two or more items in brackets are mutually exclusive options; only one can be chosen for a particular application.
- Defaults are underlined.
- Upper-case letters, numbers, commas, equal signs, and parentheses must be entered exactly as indicated. Lower-case letters represent variables which must be replaced by actual values.
- Subscripts show position in a series, and three dots indicate an ellipsis.

For example, a_1, a_2, \dots, a_5 is equivalent to a_1, a_2, a_3, a_4, a_5 and represents five a items (variables which will be replaced with actual values).

- Examples that are to be entered exactly as shown are presented in the Courier typeface, for instance:

```
ALTSEQ CODE=(F0B7 , F1B8 , F2B9 , F3BA , F4BB , F5BC , F6BD , F7BE , F8BF , F9C0)
```

Figure 5. Examples

ALTSEQ Control Statement

The ALTSEQ control statement constructs an alternate collating sequence for all control fields for which the format code AQ has been specified on the SORT/MERGE control statement, and/or an INCLUDE/OMIT control statement, and/or an INCLUDE/OMIT parameter of the OUTFIL control statement. If an alternate collating sequence has been provided by installation default, AQ fields collate against this sequence, modified by the ALTSEQ control statement. If a default alternate sequence has not been provided, AQ fields collate against the standard EBCDIC sequence, modified by the ALTSEQ control statement. AQ can be specified for one or more control fields so that those control fields all use the same alternate collating sequence.

The ALTSEQ control statement also constructs an alternate collating sequence for all control fields processed by the TRAN parameter of the INREC and OUTREC control statements, as well as the TRAN subparameter of the OUTREC parameter on the OUTFIL control statement.

The ALTSEQ control statement cannot be specified for a Tape Sort.

ALTSEQ Control Statement Format

The format of the ALTSEQ control statement is illustrated below:

ALTSEQ CODE=(ccpp₁ [,ccpp₂]...)

Figure 6. ALTSEQ Control Statement Format

CODE Parameter (Required)

The CODE parameter specifies how the characters of the current collating sequence are to be reordered to create the alternate collating sequence.

The CODE parameter can contain from 1 to 256 entries, each consisting of four hexadecimal digits. These entries must be separated by commas and enclosed in parentheses.

Each CODE entry consists of two parts:

- cc** The *cc* value represents the character that is to be repositioned in the alternate sequence.

- pp** The *pp* value indicates where the character represented by the *cc* value is to be repositioned in the alternate sequence.

The character represented by the *cc* value does not replace the character represented by the *pp* value. If both characters occur as sort control fields, they will be considered equal in the collating process.

ALTSEQ

Each character (*cc* entry) can be moved only one time. However, more than one *cc* entry can be mapped to the same *pp* value.

Sample ALTSEQ Control Statements

```
ALTSEQ CODE=(F0B7,F1B8,F2B9,F3BA,F4BB,F5BC,F6BD,F7BE,F8BF,F9C0)
```

Figure 7. Sample ALTSEQ Control Statement

This sample ALTSEQ control statement shows that the numbers 0 through 9 are to collate before the uppercase alphabet.

```
ALTSEQ CODE=(F040)
```

Figure 8. Sample ALTSEQ Control Statement

This sample ALTSEQ control statement specifies that the number 0 is to collate as equal to a blank (X'40').

END Control Statement

If present, the **END** control statement must be the last control statement. The **END** control statement is required only when the control statements are not followed by `/*` or by a job control statement (i.e., when including exits in the input stream).

The **END** control statement has no parameters, but can contain comments if the comments are preceded by at least one blank.

INCLUDE/OMIT

INCLUDE/OMIT Control Statement

The INCLUDE/OMIT control statement selects records from an input file based on comparisons testing the contents of one or more fields within the record. A field can be compared to a constant or to another field within the record. Furthermore, a binary field may enter into comparisons that involve testing the individual bits in the field. Only one INCLUDE/OMIT control statement can be specified for an application, either as an INCLUDE *or* as an OMIT control statement.

Locale-Based Comparison Processing

SyncSort supports alternative sets of collating rules based on a specified national language. The alternative collating applies to INCLUDE/OMIT (and OUTFIL INCLUDE/OMIT) comparison processing as well as to SORT/MERGE processing. A locale defines single and multi-character collating rules for a cultural environment.

Locale-based INCLUDE/OMIT processing applies only to character (CH) fields and character or hexadecimal constants compared to character fields. When LOCALE is active, a CH to BI (or BI to CH) comparison is not allowed. The illegal comparison will cause SyncSort to terminate with an error message.

For more information on locale-based processing, see “LOCALE” on page 5.20.

INCLUDE/OMIT Control Statement Format

The format of the INCLUDE/OMIT control statement follows.

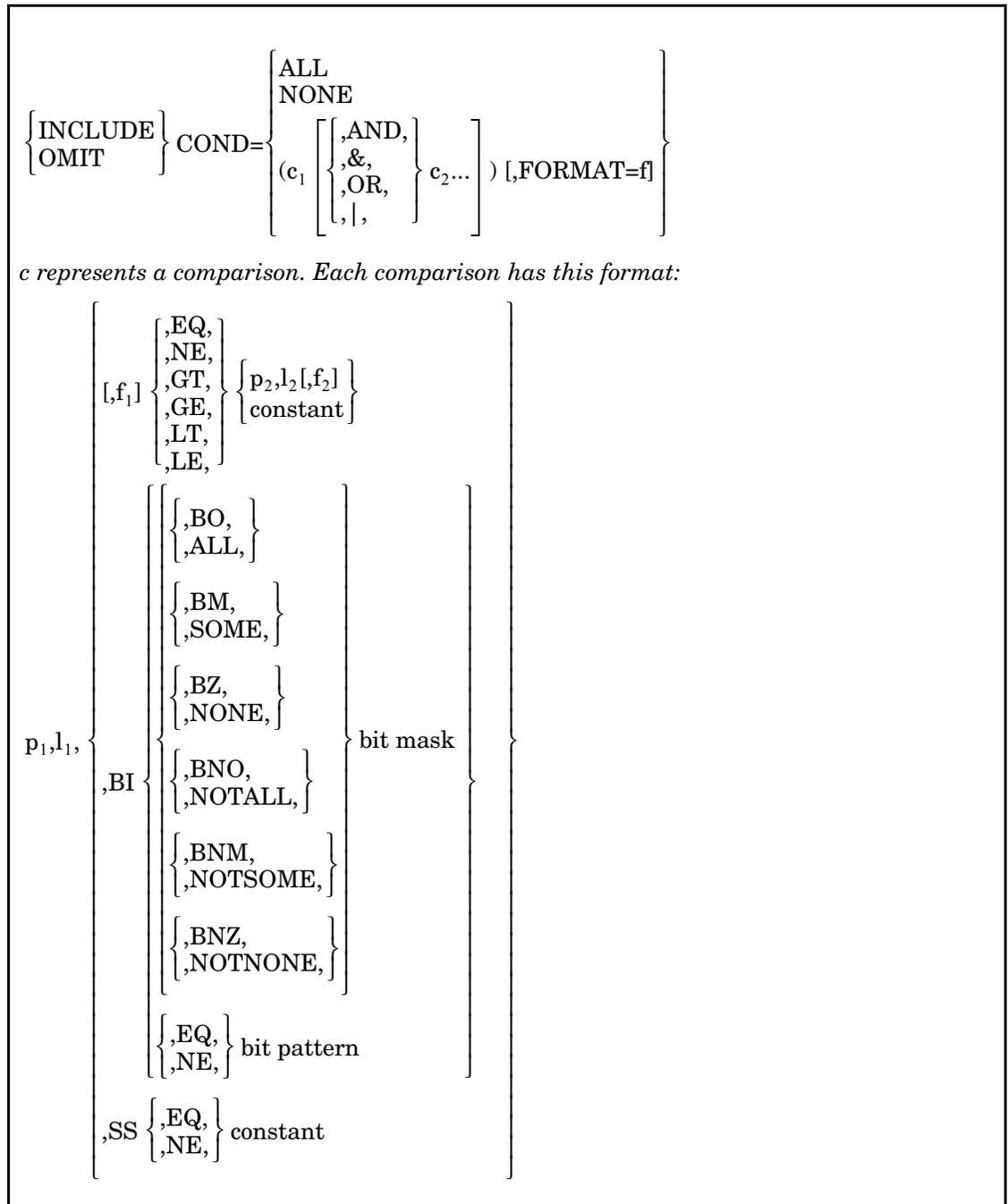


Figure 9. INCLUDE/OMIT Control Statement Format

COND Parameter (Required)

The COND parameter controls how records are included or omitted from an application. There are three forms of the COND parameter:

COND=ALL All of the input records are to be included. This is the default.

INCLUDE/OMIT

- COND=NONE** None of the input records are to be included.
- COND=comparison(s)** Specifies one or more comparisons that determine which records are to be included or omitted. Two types of comparisons are possible:
- A **standard comparison**, between two record fields or between a record field and a constant. A binary input field also allows comparison by bit mask or bit pattern.
 - A **substring comparison**, which allows the search for a constant within a field or for a field value within a constant. Use SS as the format to indicate a substring comparison.

The following several pages describe standard comparisons. For information on substring comparisons, see “Full-Date Comparisons” on page 2.30.

Each field specified in the COND parameter is identified by its position (p), length (l) and format (f). When processing variable-length records, by default all fields specified must be contained within the record. If an application is expected to reference fields not completely contained within the record, refer to “VLTESTI” on page 5.33. VLTESTI provides for processing of records that do not contain all fields.

- p** The position value indicates the first byte of the field relative to the beginning of the input record *after* E15 or E32 processing, if specified, has completed. The field must begin on a byte boundary. (Keep in mind that if a variable-length file is being referenced, the first 4 bytes must be reserved for the Record Descriptor Word.)
- l** The length value indicates the length of the field. The length must be an integer number of bytes. Refer to the table below for permissible field lengths by format.
- f** The format value indicates the format of the field. The permissible formats for standard comparisons are indicated in the table below. If all data fields have the same format, the FORMAT=f subparameter can be specified instead of the individual *f* values. If both are specified, the individual *f* values will be used. (Note that the *f* values must be specified for each compare field).

Data Format	Acceptable Field Length (Bytes)
AC	1 to 256
AQ	1 to 256
ASL	2 to 256
AST	2 to 256
BI	1 to 256
CH	1 to 256
CLO / OL	1 to 256
CSF / FS	1 to 16
CSL / LS	2 to 256
CST / TS	2 to 256
CTO / OT	1 to 256
FI	1 to 256
PD	1 to 256
PD0	2-8
Y2B	1
Y2C / Y2Z	2
Y2D	1
Y2P	2
Y2S	2
Y2T, Y2U, Y2V, Y2W, Y2X, Y2Y	2-6
ZD	1 to 256

Table 3. Valid Formats and Lengths of Include / Omit Fields

For definitions of the field format, see “Valid Formats for Merge Control Fields” on page 2.38.

The constant to which a field can be compared may be one of the following types:

INCLUDE/OMIT

decimal A decimal constant can be any length. It should *not* be enclosed in single quotes. It may or may not include a leading + or - sign. For example, 100 is a valid decimal constant. The following numeric data compare as equal: +0, -0, 0. The &DATE_xP date parameter represents the current date as a decimal number (+n) to which a field can be compared. See page 2.27 for more details.

hexadecimal A hexadecimal constant should be preceded by an X and specified in pairs of valid hexadecimal values which must be enclosed in single quotes: X'hh...hh'. For example, X'ACBF05' is a valid hexadecimal constant. The sign of the field is implicit in the representation.

character A character constant should be preceded by a C and enclosed in single quotes: C'literal'. For example, C'SALES' is a valid character constant.

The &DATE_x and &DATE_x(c) date parameters represent the current date as a character string (C'string') to which a field can be compared. See page 2.27 for more details.

You can also include or omit records based on whether their dates fall within a specified time frame before or after the current date. See page 2.29 for more details.

To include an apostrophe in a character constant, specify it as two apostrophes; for example, C'D"AGOSTINO'. If a character constant must be continued on a second card image, place a continuation character in column 72 and then begin the continuation of the constant in column 16 of the next card image.

There are two methods in which the bit level characteristics of a binary input field can be used to include or omit records. One is to compare the binary field to a bit mask; the other is to compare the binary field to a bit pattern.

bit mask A bit mask is a string of bits, specified in terms of either hexadecimal or binary digits. The bit mask indicates which bits in the input field are to be tested. Each bit in the mask whose value is 1 (ON) is tested against the corresponding bit in the input field. If the value of a mask bit is 0 (OFF), the corresponding bit in the input field is ignored.

The hexadecimal format of a bit mask is X'hh...hh,' where each 'hh' represents any pair of hexadecimal digits.

The binary format of a bit mask is B'bbbbbbbb...bbbbbbbb', where each 'bbbbbbbb' represents 8 bits or a byte. Each bit is 1 or 0. The number of bits in a binary bit mask must be a multiple of 8. The maximum length of a binary bit mask is 256 bytes (2048 bits).

A bit mask is truncated or padded on the right to the byte length of the binary field. The pad character is X'00' or B'00000000'.

bit pattern The binary format of a bit pattern is B'bbbbbbbbb...bbbbbbbbb', where each 'bbbbbbbbb' represents 8 bits or a byte. Each bit is 1, 0, or period (.). If the value of a bit in the bit pattern is 1 or 0, the corresponding bit in the binary input field is compared to 1 or 0. If . (period) occurs in a bit position in the bit pattern, the corresponding bit in the input field is ignored.

The number of bit positions in a bit pattern must be a multiple of 8. The maximum length of a bit pattern is 256 bytes (2048 bits).

A bit pattern is truncated or padded rightward to the byte length of the binary input field. The pad character is B'00000000'.

The comparison operators represent the following conditions:

EQ	Equal to
NE	Not equal to
GT	Greater than
GE	Greater than or equal to
LT	Less than
LE	Less than or equal to
BO (or ALL)	All mask bits are 1s (ON) in the input field
BM (SOME)	Some but not all mask bits are 1s (ON) in the input field
BZ (NONE)	None of the mask bits is 1 (ON) in the input field
BNO (NOTALL)	Some or no mask bits are 1s (ON) in the input field
BNM (NOTSOME)	All or no mask bits are 1s (ON) in the input field
BNZ (NOTNONE)	All or some mask bits are 1s (ON) in the input field

Rules for Multiple Comparisons

Multiple comparisons are separated by ANDs or ORs to form a logical expression. (Alternatively, & and | may be used for AND and OR). When evaluating an expression, each comparison c_n is evaluated first. Then, AND conditions are evaluated before OR conditions.

Parentheses may be used around groups of comparisons to change the default evaluation order. Any number of nested parentheses may be used. Conditions within parentheses are evaluated first, from innermost to outermost parentheses.

INCLUDE/OMIT

For example, if you wanted to select all records from your Paris office for 1995 and 1996, you might incorrectly specify:

```
INCLUDE COND=(1,4,CH,EQ,C'1995',OR,1,4,CH,EQ,C'1996',
              AND,5,5,CH,EQ,C'PARIS')
```

The AND operator in the above statement would be evaluated first, producing unexpected output. The correct statement would be:

```
INCLUDE COND=((1,4,CH,EQ,C'1995',OR,1,4,CH,EQ,C'1996'),
              AND,5,5,CH,EQ,C'PARIS')
```

The added parentheses force the OR operator to be evaluated first, thus producing the expected output.

Specifying Field-to-Field Standard Comparisons for Non-date Fields

The format of a data field determines whether or not it can be compared to another data field. The figure below illustrates which field-to-field comparisons are permitted.

	AC	AQ	ASL	AST	BI	CH	CLO OL	CSF FS	CSL LS	CST TS	CTO OT	FI	PD	PD0	ZD
AC	X														
AQ		X													
ASL			X	X											
AST			X	X											
BI					X	X									
CH					X	X									
CLO OL							X				X				
CSF FS								X	X	X					
CSL LS								X	X	X					
CST TS								X	X	X					
CTO OT							X				X				
FI												X			
PD													X		X
PD0														X	
ZD													X		X

Table 4. Permissible Field-to-Field Comparisons for Non-year Data Formats

Padding of Compared Fields

When two fields are compared, the shorter field is padded to the length of the longer field. Padding takes place as follows:

- The padding characters are blanks when the shorter field is in character format; otherwise, they are zeros of the shorter field's own format.
- Padding is on the right if the shorter field is in BI, CH or PD0 formats. Padding is on the left for all other formats.

INCLUDE/OMIT

Specifying Field-to-Field Standard Comparisons for Year Fields

The year data formats that can be used with INCLUDE/OMIT are Y2B, Y2C, Y2D, Y2P, Y2S and Y2Z. Year data formats can only be compared to other year formats; they cannot be compared to formats in the table above.

The full date formats that can be used with INCLUDE/OMIT are Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y. The full date formats may only be compared to other 2-digit year full date formats with the same number of non-year digits.

The year data formats work with the CENTWIN run-time parameter or installation option to define a 2-digit year value that is to be treated as a 4-digit year. CENTWIN defines a sliding or fixed 100-year window that determines the century to which 2-digit year data belong when processed by INCLUDE/OMIT and other control statements.

The year data formats and CENTWIN ensure that century evaluation is applied to INCLUDE/OMIT comparison conditions involving 2-digit year data. For example, without CENTWIN processing, an INCLUDE/OMIT comparison would treat the year 01 as "less than" the year 98. With CENTWIN processing, the 01 field could be recognized as a twenty-first century date (2001), which would be treated as "greater than" 98 (1998).

For details on the CENTWIN option, see "CENTWIN" on page 5.7. For details on the year data formats, see "CENTWIN Parameter (Optional)" on page 2.134. For an example of an INCLUDE control statement with a condition involving a year data field, see Figure 16 on page 2.33.

For any of the 2-digit year formats, it is valid to compare them with any of the other formats. Specifically, Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z fields can be compared to each other.

The following table summarizes the valid field to field comparisons for Full-Date formats:

Date Form	Length and Data Format Allowed
yyx and xyy	3,Y2T 3,Y2W 2,Y2U 2,Y2X
yyxx and xxyy	4,Y2T 4,Y2W 3,Y2V 3,Y2Y
yyxxx and xxxyy	5,Y2T 5,Y2W 3,Y2U 3,Y2X
yyxxxx and xxxxyy	6,Y2T 6,Y2W 4,Y2V 4,Y2Y

Table 5. Permissible Field-to-Field Comparisons for Full-Date Formats

INCLUDE/OMIT

Specifying Field-to-Constant Standard Comparisons

The format of a data field determines the type of constant to which it can be compared. The figure below illustrates which field-to-constant comparisons are permitted.

Format	Decimal	Hexadecimal	Character	Binary (bit pattern)	Year Constant
AC		X	X		
AQ		X	X		
ASL	X				
AST	X				
BI	X*	X	X	X	
CH		X	X		
CLO / OL	X				
CSF / FS	X				
CSL / LS	X				
CST / TS	X				
CTO / OT	X				
FI	X**				
PD	X				
PD0		X			
Y2B	X				X
Y2C/Y2Z	X				X
Y2D	X				X
Y2P	X				X
Y2S	X				X
Y2T***	X				X
Y2U***	X				X
Y2V***	X				X

Table 6. (Page 1 of 2) Permissible Field-to-Constant Comparisons

Format	Decimal	Hexadecimal	Character	Binary (bit pattern)	Year Constant
Y2W***	X				X
Y2X***	X				X
Y2Y***	X				X
ZD	X				

Notes: * The decimal constant cannot be higher than 4294967295 or lower than 0.
 ** The decimal constant cannot be higher than 2147483647 or lower than -2147483648.
 *** Full-Date formats

Table 6. (Page 2 of 2) Permissible Field-to-Constant Comparisons

A constant will be padded or truncated to the length of the field with which it is compared. Decimal constants are padded or truncated on the left; hexadecimal, binary, and character constants are padded on the right. The padding characters are:

Binary string B'00000000'

Character string X'40'

Hexadecimal string X'00'

Decimal fields Zeros of proper format. Decimal constants for 2-digit year formats are padded or truncated to two decimal digits representing a year. The year constant will then have CENTWIN processing applied to it for comparison to a Y2 field. These are only for the two digit year fields, not for full date constants.

The constants for PD0 comparison should not include the first digit and trailing sign of the PD0 data that will be ignored. Thus, a PD0 field of *n* bytes will be compared to a constant of *n*-1 bytes.

Current Date Constant Specification

You can compare fields to the date of a SyncSort run or the date of the run with an offset in addition to decimal fields and binary, character, and hexadecimal strings. Thus, records can more easily be included or omitted based on whether their dates are equal to, less than, or greater than the run date or the run date with an offset.

INCLUDE/OMIT

The format of a current date constant is:

current date constant $\left[\left\{ \begin{array}{c} + \\ - \end{array} \right\} \text{nnnn} \right]$

where:

- 'current date constant' is in the form of one of the &DATE_x, &DATE_x(c), &DATE_xP, or Y'DATE_x' parameters where x is 1, 2, or 3 and depends on date comparison compatibility.
- '+' indicates a date *after* the current date, and '-' indicates a date *before* the current date.
- 'nnnn' can have a maximum of 15 digits with the leftmost zeroes truncated. When the x in &DATE_x, &DATE_x(c), &DATE_xP, or Y'DATE_x' is 1 or 3, 'nnnn' can be from 0-9999 and represents offset days. When the x in &DATE_x, &DATE_x(c), &DATE_xP, or Y'DATE_x' is 2, 'nnnn' can be from 0-999 and represents offset months.

For an example of an INCLUDE control statement that uses a date range based on a date constant, see Figure 17 on page 2.34.

The forms of current date constants available for standard comparisons are:

- &DATE_x and &DATE_x(c) represent the current date as a character string (C'string') to which a field can be compared.
- &DATE_xP represents the current date as a decimal number (+n) to which a field can be compared.
- Y'DATE_x' represents the current date with a Y constant (Y'string') to which a field can be compared.

The following table shows the current date constants and the format produced by each. The c character in &DATE_x(c) represents a non-blank separator character, except open and close parentheses.

Current Date Constant	Generated Constant
&DATE1	C'yyyymmdd'
&DATE1(c)	C'yyyymmdd'
&DATE1P	+yyyymmdd
&DATE2	C'yyyymm'
&DATE2(c)	C'yyyymm'
&DATE2P	+yyyymm
&DATE3	C'yyyddd'
&DATE3(c)	C'yyyddd'
&DATE3P	+yyyddd
Y'DATE1'	Y'yymmdd'
Y'DATE2'	Y'yymm'
Y'DATE3'	Y'yddd'

Table 7. Current Date Constant Formats

Full-Date Format Constant Specifications

Constants used for full-date comparisons should have the same number of digits in the constant as in the full-date field that has been specified. Leading zeros must be specified when needed. The constant is constructed from two items; the first is a 2-digit year and the second is a value representing the months or days that comprise the remainder of the full date format. For example, if a 5-byte Y2W field were to be compared for a value greater than the 20th day of 1996, 96020 should be the code for the constant.

Constants can be coded to represent special values, such as those found in header or trailer records. All zeros or nines may be used with Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y. The same number of digits must be present as in the field that is being compared. The constant string Y'LOW' (representing binary zeros), Y'HIGH' (representing binary ones), or Y'BLANKS' (representing blanks) may be coded with the fields Y2T, Y2W, and Y2S. Y'DATEX' (representing the current date) may be coded with certain full-date formats specifically (see Table 8).

INCLUDE/OMIT

Y Constant	Date Form	Length and Data Format Allowed
Y'DATE1'	yyxxxx and xxxxyy	6,Y2T 6,Y2W 4,Y2V 4,Y2Y
Y'DATE2'	yyxx and xxyy	4,Y2T 4,Y2W 3,Y2V 3,Y2Y
Y'DATE3'	yyxxx and xxxyy	5,Y2T 5,Y2W 3,Y2U 3,Y2X

Table 8. Full-Date Comparisons

Substring Comparisons

Substring comparison (SS format) can be based on either of the following searches:

- Match occurrence of a constant within a record field
- Match occurrence of a record field within a constant.

In the first form, the length of the constant is **less** than the length of a specified field. Records will be searched for the occurrence of the constant anywhere within the field. The condition will be true if an EQ operator is specified and the constant is found or if a NE operator was specified and the constant is **not** found. For example, consider the constant "ANYTOWN" and a 60-byte field that contains an address. Records will be searched for the occurrence of the literal "ANYTOWN" anywhere within the 60-byte address field. If a match is found and the logical operator is EQ, then the logical result is "true." The logical result is also "true" if the literal does not appear within the 60 bytes and the logical operator is NE.

In the second form, the length of a constant is **greater** than the length of a specified field. Records will be searched for an occurrence of the field within the constant. For example, the constant 'A02,A05,A06,A09', which is composed of substrings separated by commas, can be compared against the contents of a 3-byte field within the record. If the 3-byte field matches any 3-byte character string in the constant, the logical result is "true" if the logical operator is EQ.

The character used to separate elements of the constant should be a character that does not appear in the field being compared. The comparison is then equivalent to a standard comparison with ORed conditions. That is, the condition is true if 'A02' OR 'A05' OR 'A06' OR

'A09' is found in the field being compared. The substring comparison is a much more compact expression than multiple OR conditions in a standard comparison.

For both forms of substring comparison, constants and fields in the record can be from 1 to 256 bytes in length. Constants can be in either character or hexadecimal format. (Refer to the description of constants just after Table 6 on page 2.26.)

See Figure 15 on page 2.33 for an example of how to use the substring comparison.

Sample INCLUDE/OMIT Control Statements

Example 1

```
INCLUDE COND=(24,4,PD,LT,28,4,PD,OR,10,2,CH,EQ,C'NY')
```

Figure 10. Sample INCLUDE Control Statement

In this example, records will be included in the application if the numeric value in the field beginning in byte 24 is less than the numeric value in the field beginning in byte 28 *or* if the character value in the field beginning in byte 10 is equal to NY.

Example 2

```
OMIT COND=(1,3,ZD,EQ,100,AND,20,1,CH,NE,X'40')
```

Figure 11. Sample OMIT Control Statement

In this example, records will be omitted from the application if the numeric value in the field beginning in byte 1 is equal to 100 *and* if the character value in byte 20 is not equal to a blank (X'40').

The next set of control statements exemplifies record selection using bit level logic. The first two examples involve a comparison between a bit mask (shown coded in binary and hexadecimal format) and a binary input field. The third example is a comparison between a bit pattern and a binary field.

Example 3

```
INCLUDE COND=(10,1,BI,ALL,B'01001000')
or INCLUDE COND=(10,1,BI,ALL,X'48')
```

Figure 12. Sample INCLUDE Control Statement Using a Bit Mask

The record selection condition has the following elements (from left to right): a binary field (BI) of length 1 byte that starts at column 10 of the record, a comparison operator (ALL), and a bit mask (B'01001000' in binary, X'48' in hexadecimal). Counting from the left, the

INCLUDE/OMIT

second and fifth bits of the bit mask are ON (1). For the selection condition to be true, the same bits must be ON in the binary input field. Therefore, if the input field contains, for example, 01001000, 01111000 or 11111111, the condition for the inclusion of records is satisfied. However, if the input field contains a bit string where *both* mask bits are not ON (e.g., 01000000, in which the fifth bit is not ON), the condition fails and the records are omitted.

Example 4

```
INCLUDE COND=(10,1,BI,NOTNONE,B'01001000')
or INCLUDE COND=(10,1,BI,NOTNONE,X'48')
```

Figure 13. Sample INCLUDE Control Statement Using a Bit Mask

The condition for the inclusion of records is met if at least one of the mask bits is ON in the input field. Therefore, the condition would evaluate as true, if the bit string in the binary field were 01000000 (the second bit is ON), 000010000 (the fifth bit is ON), 01001000 (both the second and fifth bit are ON). However, with the string 10000111, for instance, in the input field, the specified condition would evaluate as false (resulting in the omission of records), since neither mask bit is ON.

The above method of comparing a binary input field to a bit mask is useful for testing the contents of a "flag" byte where each bit has a different meaning.

Example 5

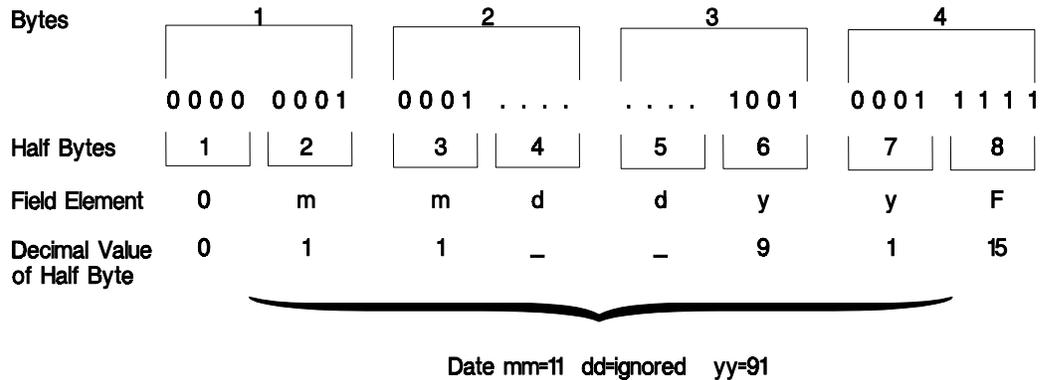
```
INCLUDE COND=(21,4,BI,EQ,B'000000010001.....100100011111')
```

Figure 14. Sample INCLUDE Control Statement Using a Bit Pattern

The condition specifies a 4-byte long binary input field (BI) in column 21, a logical relationship (EQ), and a bit pattern. The bit pattern describes the required sequence of 1s and 0s in the first and last twelve bit positions. The row of periods in the pattern represents the part of the string that is irrelevant to the definition of the condition. The condition is true, if the sequence of 1s and 0s in the input field is identical to that described in the bit pattern.

The method of comparing a binary input field to a bit pattern is useful when testing for numeric digits that are one half byte each, as in the packed data format. For example, assume that the binary input field specified in the condition above is a date field in the PD format X'0mmdyyF'. Each date element is split across a byte boundary. The second half-byte of each byte (except the last) represents the first of the two digits that form a date element (**mm,dd,yy**). (In the last byte, the second half-byte--1111 in binary and F in hexadecimal--stands for the fact that the bit pattern encodes a packed decimal.) The first half-byte of each byte (except the first) represents the second digit of a date element (**mm,dd,yy**). (The first half-byte, i.e. 0000, of the bit pattern gives it the length specified for the binary

field at column 21.) Mapping this scheme onto the bit pattern in the control statement results in the following.



That is, the above control statement is an instruction to select just those records in whose date field *mm* and *yy* equal 11 and 91, respectively, while *dd* can have any value. In other words, the records thus selected are those from November 1991.

Example 6

The following example illustrates substring comparisons.

```
INCLUDE COND=(11,60,EQ,C'ANYTOWN',
              OR,121,3,EQ,C'A01,A05,A06,A09'),FORMAT=SS
```

Figure 15. Sample INCLUDE Control Statement Using Substring Compares

In this example, a record will be included in the application if either of the following conditions is true:

- The literal 'ANYTOWN' is found in the 60-byte field starting at position 11 in the record.
- The contents of the 3-byte field starting at position 121 matches one of the four substrings ('A01', 'A05', 'A06', or 'A09') in the constant.

Example 7

The following example illustrates an INCLUDE comparison based on CENTWIN processing.

```
INCLUDE COND=(20,2,Y2C,GT,96)
```

Figure 16. Sample INCLUDE Control Statement with CENTWIN Processing

INCLUDE/OMIT

In this example only records whose data are from the years greater than 1996 will be included in the application. If the CENTWIN parameter were set to 1980, representing a century window of 1980 to 2079, the records would be processed in the following manner:

Contents of Positions 20 and 21	Record Disposition
84	Omitted - represents 1984
99	Included - represents 1999
37	Included - represents 2037

Example 8

The following INCLUDE control statement illustrates the use of the current date constant and the current date with an offset to include records with dates starting with the current date and spanning through the two week period prior to the current date.

```
INCLUDE COND=( 5 , 8 , ZD , LE , &DATE1P , AND , 5 , 8 , ZD , GT , &DATE1P-14 )
```

Figure 17. Sample INCLUDE Control Statement Using Current Date Constant and Current Date Constant With an Offset Comparison

If the application were run on April 25, 2002, the records included would have dates in the 8-bytes field starting at position 5 from April 12, 2002 through and including April 25, 2002.

Applications using the INCLUDE/OMIT control statement are illustrated in “Chapter 3. How to Use SyncSort’s Data Utility Features”.

INREC Control Statement

The INREC control statement reformats the input records. Use the INREC control statement to add, delete, or reformat fields *before* the records are sorted or merged. Use the OUTREC control statement or the OUTREC parameter of the OUTFIL control statement to delete or reformat fields *after* the records are sorted or merged. Note that INREC is performed after E15 exit processing and INCLUDE/OMIT control statement processing.

Using the INREC control statement to delete data fields improves sort performance by reducing the number of bytes SyncSort for z/OS must process. The same result may be achieved in some cases by changing the data format of certain fields. For example, if you need to change the format of a ZD field to PD, which reduces the number of bytes for the field, it is more efficient to use INREC rather than OUTREC for the conversion. Additionally, for SORT/MERGE processing PD fields are processed more efficiently than ZD fields.

Except for CONVERT, all the functions performed by the OUTREC control statement, such as inserting character strings or changing the data format of a numeric field, can also be performed by the INREC control statement. (See “OUTREC Control Statement” on page 2.88 for an explanation of these functions.) For example, you can use the INREC control statement to insert zeros of the proper format to expand a numeric field before SUM processing to prevent arithmetic overflow. However, you will usually want to use the OUTREC control statement rather than the INREC control statement to expand the record because OUTREC processing takes place *after* records are sorted or merged.

If you use the INREC control statement to reformat the input record, remember to use the post-INREC field positions when you specify the SORT, MERGE, SUM, OUTREC, and/or OUTFIL control statements.

If the SEQNUM function is used in a SORT application to insert a sequence number field in the record, this field will reflect the order of the records prior to sorting. In a MERGE application, the field will reflect the order of the records as they were read from each input in the merge.

INREC Control Statement Format

The format of the INREC control statement is illustrated below:

```
INREC FIELDS=(...)
```

Figure 18. INREC Control Statement Format

FIELDS Parameter (Required)

The FIELDS parameter specifies the data fields to be included in the application. See “OUTREC Control Statement” on page 2.88 for a complete description of the FIELDS parameter.

INREC

Sample INREC Control Statement

```
INREC FIELDS=(1:1,20,21:40,15,ZD,PD,29:60,5)
```

Figure 19. Sample INREC Control Statement

This INREC control statement specifies three data fields from an 80-byte record:

- The first field begins in byte 1 of the input record and is 20 bytes long.
- The second field begins in byte 40 of the input record and is a 15-byte ZD field. The data format is to be converted to PD. Since the input field contains 15 decimal digits, the converted PD output field created by SyncSort will be 8 bytes long.
- The third field begins in byte 60 of the input record and is 5 bytes long.

These three fields have been positioned to begin in bytes 1, 21, and 29, as indicated by their column prefixes.

The reformatted input record is now just 33 bytes long.

For comprehensive examples that illustrate the INREC control statement see “Chapter 3. How to Use SyncSort’s Data Utility Features”.

MERGE Control Statement

The MERGE control statement is required for every merge application. The MERGE control statement can also define a copy application.

Cultural Environment Support

Cultural environment support allows you to choose an alternative set of collating rules based on a specified national language. The alternative collating applies to SORT/MERGE and INCLUDE/OMIT processing.

For additional detail, see “LOCALE” on page 5.20.

MERGE Control Statement Format

The format of the MERGE control statement is illustrated below:

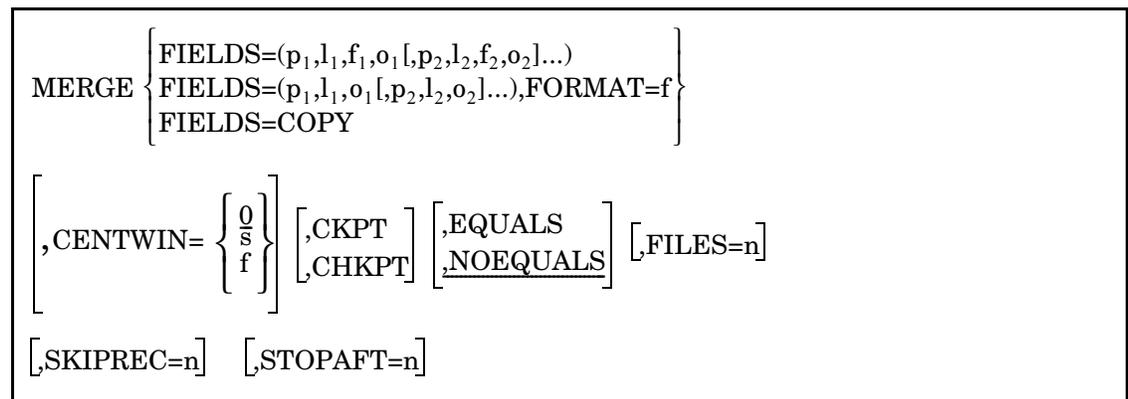


Figure 20. MERGE Control Statement Format

FIELDS Parameter (Required for a Merge)

The FIELDS parameter is required for a merge. It describes the control fields.

List the control fields in order of greatest to least priority, with the primary control field listed first, followed by progressively less significant fields. You can specify up to 128 control fields; however, if fields require complex internal processing, the limit for a particular execution may be less than 128.

Each field specified in the FIELDS parameter is identified by its position p , length l , format f and order o .

- p** The position value indicates the first byte of the field relative to the beginning of the input record *after* INREC and/or E32 processing, if specified, have completed.

MERGE

Binary control fields can begin on any bit of a byte. When a binary field does not begin on a byte boundary, you must specify the bit number (0-7). For example, a position value of 21.3 refers to the 4th bit of the 21st byte of the record.

- l** The length value indicates the length of the control field. The length value must be an integer number of bytes, except for the length of a binary control field which can be specified in bits. For example, a length value of 0.5 refers to a binary control field 5 bits long.

For signed fields, the length value must include the area occupied by the sign.

- f** The format value indicates the data format. For a list of valid formats, refer to the Format Code Chart in the next section, "Valid Formats for Merge Control Fields." If all the control fields have the same format, you can specify the format value once by using the `FORMAT=f` subparameter. If you specify both the individual *f* values and the `FORMAT` subparameter, the individual *f* values will be used. (Note that the *f* values must be specified for each control field).

- o** The order value indicates how the field is to be collated:

A=Ascending order

D=Descending order

E=As modified by an E61 exit.

Valid Formats for Merge Control Fields

The following table lists the valid formats for merge control fields.

Code	Data Format	Field Length (bytes)
AC*	EBCDIC characters are translated to their ASCII equivalents before sorting.	1 to 4091†
AQ*	Character. Records are sorted according to an alternate sequence specified either in the <code>ALTSEQ</code> control statement or as an installation default.	1 to 4091†
ASL*	Leading separate sign. An ASCII + or - precedes numeric field. One digit per byte.	2 to 256
AST*	Trailing separate sign. An ASCII + or - trails numeric field. One digit per byte.	2 to 256

Table 9. (Page 1 of 3) Format Code Chart

Code	Data Format	Field Length (bytes)
BI	Binary. Unsigned.	1 bit to 4092**
CH	Character. Unsigned.	1 to 4092**
CLO* OL*	Leading overpunch sign. Hexadecimal F,C,E, or A in the first 4 bits of your field indicates a positive number. Hexadecimal D or B in the first 4 bits indicates a negative number. One digit per byte. CMP=CLC is forced.	1 to 256
CSF FS	Floating sign format. An optional leading sign may be specified immediately to the left of the digits. If the sign is a -, the number is treated as negative. For other characters, the number is treated as positive. Characters to the left of the sign are ignored.	1 to 16
CSL* LS*	Leading separate sign. An EBCDIC + or - precedes numeric field. One digit per byte. CMP=CLC is forced.	2 to 256
CST* TS*	Trailing separate sign. An EBCDIC + or - follows numeric field. One digit per byte. CMP=CLC is forced.	2 to 256
FI	Fixed point. Signed. (Equivalent to Signed Binary.)	1 to 256
FL	Floating point. Normalized. Signed.	2 to 16
PD	Packed decimal. Signed.	1 to 256
PD0*	Packed decimal. 2-8-byte packed decimal data with the first digit and trailing sign ignored. The remaining bytes are treated as packed decimal digits. Typically PD0 is used with century window processing and Y2P format; Y2P processes the year, while PD0 processes month and day.	2-8
Y2B*	Binary. 2-digit, 1-byte binary year data treated as a 4-digit year by CENTWIN (century window) processing.	1
Y2C*	Character. 2-digit character year data treated as a 4-digit year by CENTWIN (century window) processing. Processing is identical to Y2Z fields.	2

Table 9. (Page 2 of 3) Format Code Chart

MERGE

Code	Data Format	Field Length (bytes)
Y2D*	Packed decimal. 2-digit, 1-byte packed decimal year data treated as a 4-digit year by CENTWIN (century window) processing.	1
Y2P*	Packed decimal. 2-digit, 2-byte packed decimal year data. Of the four packed digits contained in the 2 bytes, the first digit and trailing sign are ignored; the two inner digits are treated as a 4-digit year by CENTWIN processing.	2
Y2S*	Character or zoned decimal. 2-digit, 2-byte valid numeric data treated as a 4-digit year by CENTWIN (century window) processing, as for Y2C and Y2Z. However, certain data are not treated as year data. Data with binary zeros (X'00') or a blank (X'40') in the first byte will be collated before valid numeric year data for ascending order (after year data for descending order). Data with all binary ones (X'FF') in the first byte will be collated after valid numeric year data for ascending order (before year data for descending order). Zones are ignored, as for Y2C and Y2Z, except for data where the first byte begins with X'00', X'40' or X'FF'.	2
Y2T* Y2U* Y2V* Y2W* Y2X* Y2Y*	Full-date, character, binary, or packed decimal formats. Full-date data formats can be used to sort or merge a variety of date fields. They can process dates ending or starting with year digits (x...xyy or yyx...x). They can also process non-date data commonly used with dates. For details, see page 2.140.	2-6
Y2Z*	Zoned decimal. 2-digit, 2-byte zoned decimal year data treated as a 4-digit year by CENTWIN (century window) processing. The zones are ignored. Processing is identical to Y2C fields.	2
ZD CTO* OT*	Zoned decimal. Trailing overpunch in the first 4 bits of the rightmost byte gives the sign. Hexadecimal F,C,E, or A indicates a positive number. Hexadecimal D or B indicates a negative number. One digit per byte. CTO forces CMP=CLC.	1 to 256
<p>Notes: * Cannot be used with Tape Sort. ** 4084 for variable-length records. † 2043 for variable-length records.</p>		

Table 9. (Page 3 of 3) Format Code Chart

For information on the year data formats (Y2B, Y2C, Y2D, Y2P, Y2S and Y2Z) plus the related data format PD0, see “CENTWIN Parameter (Optional)” on page 2.41 and “Converting Year Data with Century Window Processing on INREC, OUTREC, or OUTFIL OUTREC” on page 2.100. Also see "Specifying Field-to-Field Standard Comparisons for Year Fields" in the INCLUDE/OMIT Control Statement section of this chapter.

Rules for Specifying Merge Control Fields

- For fixed-length records, the sum of the lengths of all control fields cannot exceed 32752 bytes. When EQUALS is in effect, the sum of their lengths cannot exceed 4088 bytes.
- For variable-length records, all control fields must be located within the first 4084 bytes and the sum of their lengths cannot exceed 4084 bytes. When EQUALS is in effect, all control fields must be located within the first 4080 bytes and the sum of their lengths cannot exceed 4080 bytes.
- Control fields can be in contiguous or non-contiguous locations in the record.
- Remember that for variable-length records, the first 4 bytes are reserved for the Record Descriptor Word, so the first byte of the *data* portion of the record is byte 5.
- If the output file is a key-sequenced VSAM cluster, the VSAM key must be the first control field specified.

Comparing PD and ZD Control Fields

The CMP PARM determines how PD and ZD control fields will be compared. When CMP=CPD is in effect, the Compare Decimal (CP) instruction is used for the compare. ZD fields are packed and then compared. This method has performance advantages. However, invalid PD data may cause a system 0C7 abend and program termination. Moreover, the integrity of ZD fields is only guaranteed when they contain valid ZD data. The CMP=CPD method cannot be used for control fields that exceed 16 bytes, for variable-length merges when an even value (0, 2, 4, or 6) is specified for the VLTEST PARM, or for a Tape Sort.

When CMP=CLC is in effect, no data validation is performed and the integrity of the output is maintained, even if the sign for a PD or ZD field is invalid. This method is always used if any control field exceeds 16 bytes, for variable-length merges when an even value is specified for the VLTEST PARM, and for a Tape Sort.

CENTWIN Parameter (Optional)

The CENTWIN run-time or installation option acts on 2-digit year data. At run-time, CENTWIN can be specified as either a PARM option or a SORT/MERGE control statement parameter. CENTWIN generates a century window (for example, 1950 through 2049) that determines the century to which a 2-digit year belongs. CENTWIN ensures that year data spanning centuries will be sequenced correctly. Without CENTWIN processing, an ascending collation would sequence the year 01 before the year 98. With CENTWIN

MERGE

processing, the 01 field could be recognized as a twenty-first century date (2001) and would thus be sequenced after 98 (1998).

For more information on specifying the CENTWIN option, see “CENTWIN” on page 5.7.

CENTWIN processing only applies to data defined as year data formats (Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z) and the full-date formats (Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y). These data formats enable SyncSort to process 2-digit year fields as 4-digit years. A related data format, PD0, can be used to process the month and day portions of packed decimal date fields. To correctly specify date fields for CENTWIN MERGE processing, you should be familiar with the CENTWIN-related data formats.

The following describes each of the year data formats and provides MERGE control statement examples:

The Y2B Format

This format is used to sequence 2-digit, 1-byte binary year data with CENTWIN processing. The binary values are converted to decimal, and the two low order digits are used as year data. Thus, while binary and decimal values range from 00 to 255, year values range from 00 to 99. The relationship between binary, decimal and year values is shown in the following table:

Binary Value	Decimal Value	Year Value
X'00' to X'63'	00 to 99	00-99
X'64' to X'C7'	100 to 199	00-99
X'C8' to X'FF'	200 to 255	00-55

Table 10. Possible Values Representing Year Data with Y2B

The Y2C and Y2Z Formats

These formats represent 2-digit, 2-byte year data in either character (Y2C) or zoned decimal (Y2Z) format. Either Y2C and Y2Z formats can be used with data of the form

X'xyxy'

where *y* is a hexadecimal year digit 0-9 and *x* is hexadecimal 0 through F. Y2C and Y2Z ignore the *x* digits, leaving *yy*, the 2-digit unsigned year representation.

Suppose you have a character or zoned decimal date field *mmddy* that begins at byte 20. You can use either Y2C or Y2Z to process the *yy* field. As the following example indicates, you could specify three sort keys to correctly sort this date:

```

MERGE FIELDS=(24,2,Y2C,A, * Collates yy field as 4-digit year
              20,2,CH,A, * Collates mm field
              22,2,CH,A) * Collates dd field

```

The yy field (24,2) will be processed according to the century window setting. For example, if CENTWIN=1945, the field yy=45 will be sequenced as if it were 1945, and yy=44 would be sequenced as if it were 2044. Thus, for an ascending sort, 44 would **follow** 45.

The Y2D Format

This format is used to sequence 2-digit, 1-byte packed decimal year data with CENTWIN processing. Use Y2D to extract the year data yy from packed decimal date fields. For example, consider a 3-byte packed decimal data field defined as

```
X'yyddd'
```

This field has the year yy in the first byte and the day ddd in bytes 2 and 3. The packed decimal sign s would be in the last digit (half byte) of the third byte. To sort this date field, which begins at byte 20, with 4-digit year processing, use the following MERGE control statement:

```

MERGE FIELDS=(20,1,Y2D,A, * Collates 2-digit year as 4-digit year
              21,2,PD,A) * Collates ddds as 3 digits (ddd)

```

The Y2P Format

This format is used to sequence 2-digit, 2-byte packed decimal year data with CENTWIN processing. Use Y2P to extract the year data yy from packed decimal date fields spanning 2 bytes. For example, a packed decimal date of the form *yymmdd* would be stored as 4 bytes:

```
yymmdd = X'0yymmddC'
```

where the trailing C (sometimes F) is a positive sign and the leading 0 pads the field on the left to make an even number of digits.

Notice that the components of the date span bytes:

```
0y ym md dC
```

Y2P handles this condition by ignoring the first and last half bytes of the 2-byte field specification. Thus, Y2P processes 0yym as yy, ignoring the leading digit (0) and the trailing digit m that is part of the month.

MERGE

The following example uses Y2P to collate the year portion of the date field, which begins at byte 20:

```
MERGE FIELDS=(20,2,Y2P,A) * Collates yy field as 4-digit year
```

The field specification 20,2,Y2P treats X'0yym' as X'yy', and CENTWIN processing sorts yy as a 4-digit year yyyy.

The PD0 format, described below, can assist Y2P by processing month and day data that overlap year data in the original field.

The Y2S Format

This format is used to sequence 2-digit, 2-byte character or zoned decimal data. The Y2S format is identical to Y2C and Y2Z for valid numeric data, but Y2S treats data that begin with X'00', X'40' or X'FF' as non-year data. Thus, the Y2S format can distinguish records that have non-year data in the first byte of the year field, allowing such records to be collated differently from other records.

Y2S treats non-year data as follows:

- Data with binary zeros (X'00') or a blank (X'40') in the first byte will not have century window processing applied to it. Instead, such data will be collated in sequence, **before** valid numeric year data for ascending order or **after** the year data for descending order.
- Data with all binary ones (X'FF') in the first byte will also not have century window processing applied to it. Instead, such data will be collated **after** valid year numeric data for ascending order or **before** the year data for descending order.

Zones are ignored, as for Y2C and Y2Z, except for data where the first byte begins with X'00', X'40' or X'FF'.

As an example, suppose you want to preserve the input order of header and trailer records at the start or end of the file, and your header/trailer records are identified by binary zeros (X'00'), a blank (X'40') or binary ones (X'FF') in the first byte of the date field. The Y2S format allows CENTWIN to identify the header/trailer records and treat them differently from other records.

The PD0 Format

This format is used to sequence 2-8 byte packed decimal data. PD0 ignores the first digit and trailing sign during processing. PD0 is normally used in conjunction with the Y2P data format. The Y2P format is used to process the 2-digit year portion of a packed decimal date field, while the PD0 format is used to process the month and day portion of the field.

Although PD0 is typically used with Y2P, CENTWIN processing is not applied to PD0.

Consider the packed decimal date field used in the Y2P example above:

```
yymmdd = X'0yymmddC'
```

where the trailing C (sometimes F) is a positive sign and the leading 0 pads the field on the left to make an even number of digits.

Notice that the components of the date span bytes:

```
0y ym md dC
```

The date can be processed as follows:

- Y2P processes the year component X'0yym' as X'yy'.
- PD0 processes the month and day components X'yymmddC' as X'mmdd'.

The following MERGE control statement can be used to collate the entire date with CENTWIN processing:

```
MERGE FIELDS=(20,2,Y2P,A, * Treats X'0yym' as X'yy'; collates yy as YYYY
                21,3,PD0,A) * Treats X'yymmddC' as X'mmdd'
```

Full-Date Formats

Full-date formats can be used to sort or merge various date fields, processing dates ending or starting with year digits. They also process non-date data that are used with dates. For a full description of full-date formats, see the following section.

Using Full-Date Formats with CENTWIN

SyncSort's full-date data formats enable you to sort or merge a variety of date fields. The full-date formats are Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y. These date formats can process dates ending or starting with year digits:

- x...xyy (for example: qyy, mmyy, dddy, or mmdyy)
- yyx...x (for example: yyq, yymm, yyddd, or yymmdd)

The full-date formats also process non-date data commonly used with the dates. SyncSort interprets two-digit years (yy) according to the century window specified by the CENTWIN option. CENTWIN processing does not apply to non-date data.

In most cases, for CH, ZD, and PD date fields the full-date data formats are easier to use than the 2-digit date formats. The 2-digit formats can be more difficult because you must divide the date into its components. This requires care, particularly for PD dates, where

MERGE

date components (q, dd, mm, or yy) may span bytes or occupy only part of a byte. The full-date formats, on the other hand, process such dates automatically.

The table below describes the full-date formats. For date forms not in the table, use the 2-digit year formats or the non-year formats.

Note the following symbols used in the table:

- y** year digit (0-9)
- x** non-year digit (0-9)
- s** sign (hexadecimal A-F)
- 0** unused digit

Full-Date Format	Data Format	Date Form	Example Date Form	Length (bytes)
Y2T	CH, BI	yyx	yyq	3
		yyxx	yymm	4
		yyxxx	yyddd	5
		yyxxxx	yymmdd	6
Y2U	PD	yyx (X'yyxs')	yyq	2
		yyxxx (X'yyxxxs')	yyddd	3
Y2V	PD	yyxx (X'0yyxxs')	yymm	3
		yyxxxx (X'0yyxxxxs')	yymmdd	4
Y2W	CH, BI	xyy	qyy	3
		xxyy	mmyy	4
		xxxxy	dddy	5
		xxxxyy	mmddy	6
Y2X	PD	xyy (X'xyys')	qyy	2
		xxxxy (X'xxxxyys')	dddy	3
Y2Y	PD	xxyy (X'0xxxyys')	mmyy	3
		xxxxyy (X'0xxxxxyys')	mmddy	4

Table 11. Full-Date Formats

MERGE

The table indicates the full-date formats that can be used with character (CH), binary (BI), or packed decimal (PD) data. Note the recognized non-date values:

Character or binary (Y2T and Y2W full-date formats)

C'0...0' (CH zeros)
C'9...9' (CH nines)
Z'0...0' (ZD zeros)
Z'9...9' (ZD nines)
X'00...00' (BI zeros)
X'40...40' (blanks)
X'FF...FF' (BI ones)

Packed (Y2U, Y2V, Y2X, and Y2Y full-date formats)

P'0...0' (PD zeros)
P'9...9' (PD nines)

The following two examples illustrate how you might use the *Full-Date Formats* table:

- Suppose you have a packed decimal (PD) date field of the form mmyy. To sort this field correctly, you would use the Y2Y 3-byte format from the table. Thus, if the field starts in position 30, you would specify the following SORT control statement to sort in descending order:

```
SORT FIELDS=(30,3,Y2Y,D)
```

Any PD fields of all PD zeros or all PD nines will be processed automatically as non-date data.

- Suppose you have a character (CH) date field of the form yymmdd. To sort this field correctly, you would use the Y2T 6-byte format from the table. Thus, if the field starts in byte 40, you would specify the following SORT control statement to sort in ascending order:

```
SORT FIELDS=(40,6,Y2T,A)
```

Any CH zeros, CH nines, BI zeros, blanks, and BI ones will be processed automatically as non-date data.

Collating Sequence with Full-Date Formats

For full-date formats, the yy component is always sorted first (treated as primary key). This is so even when the yy is physically at the rightmost end of the field, as for Y2W, Y2X, and Y2Y. For example, a 6-byte Y2W field has the form xxxxyy. This is collated with the yy as the primary key and xxxx as the secondary key. Because SyncSort automatically collates the year character first, you don't have to deal with yy manually, for example by using PD0 and Y2D.

It is important to understand that the xxxx component of a full-date format must be designed to collate as a unit. Suppose you have the 6-byte Y2T field yyxxxx. If you collate this field in ascending order, then yy collates first (the primary key) with xxxx collating second (secondary key). Consider two possibilities:

- If yyxxxx is actually yymmdd, you will be sorting first by year, then month, then day.
- If yyxxxx is actually yyddmm, you will sorting by year, then day, then month. In most cases, sorting in this way would not be what you intended.

To correctly collate a date, the date components must be in an order suitable for collating. For example, mmddyy and yymmdd will collate correctly, but ddmmyy or yyddmm will not. For date forms that will not collate correctly, you must use one of the 2-digit year formats (Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z).

The following table shows the order for ascending collation when using full-date formats with the CENTWIN option:

Full-Date Format	Date Format	Ascending Sort Sequence
Y2T Y2W	CH, BI	BI zeros Blanks CH/ZD zeros Lower century dates (e.g. 1980) Higher century dates (e.g. 2010) CH/ZD nines BI ones
Y2U Y2V Y2X Y2Y	PD	PD zeros Lower century dates (e.g. 1980) Higher century dates (e.g. 2010) PD nines

Table 12. Ascending Sort Sequences

For a descending sort, the collation order is reversed.

Other date formats (non-full-date), with the exception of Y2S, do not process non-date data; their sort sequence for ascending sorts is simply lower century dates than higher century dates.

MERGE

Examples Using Full-Date Formats

Example 1 (Y2W)

The following SORT control statement sorts a C'mmddy' date field in ascending order, with the previously set fixed century window 1984-2083:

```
SORT FIELDS=(10,6,Y2W,A)      * Sort C'mmddy' in ascending order
                                * with Y2W
                                * and previously set century window 1984-2083
```

The *Full-Date Formats* table above indicates that the 6-byte Y2W form is appropriate for a CH input field of the form xxxxyy. As shown in the following table, the output will be collated as C'yyyymmdd', with the non-date data (zeros) appearing correctly at the beginning of the sorted output.

SORTIN Input mmddy	Record Order after Sorting mmddy	Actual Date after Sorting yyyy/mm/dd
021783	000000	non-date data
092206	070484	1984/07/04
081395	081395	1995/08/13
110210	092206	2006/09/22
000000	110210	2010/11/02
070484	043060	2060/04/30
043060	021783	2083/02/17

Example 2 (Y2T)

The following SORT control statement sorts a Z'yyddd' date field in descending order, with the previously set fixed century window 1921-2020:

```
SORT FIELDS=(20,5,Y2T,D)      * Sort Z'yyddd' in descending order
                                * with Y2T
                                * and previously set century window 1921-2020
```

The *Full-Date Formats* table above indicates that the 5-byte Y2T form is appropriate for a ZD input field of the form yyddd. As shown in the following table, the output will be collated as Z'yyyddd', with the non-date data (nines and zeros) appearing correctly at the beginning and end of the sorted output.

SORTIN Input yyddd	Record Order after Sorting yyddd	Actual Date after Sorting yyyy/ddd
00000	99999	non-date data
50237	20153	2020/153
99999	20047	2020/047
20047	01223	2001/223
94001	94001	1994/001
01223	50237	1950/237
20153	21148	1921/148
21148	00000	non-date data

Example 3 (Y2Y)

The following SORT control statement sorts a P'mmddy' (X'0mmddy') date field in ascending order, with the previously set fixed century window 1921-2020:

```
SORT FIELDS=(26,4,Y2Y,A)      * Sort P'mmddy' in ascending order
                                * with Y2Y
                                * and previously set century window 1921-2020
```

The *Full-Date Formats* table above indicates that the 4-byte Y2Y form is appropriate for a PD input field of the form xxxxyy. As shown in the following table, the output will be collated as P'yyyymmdd', with the non-date data (zeros and nines) appearing correctly at the beginning of the sorted output. Note that the first two columns are in hexadecimal.

SORTIN Input mmddy	Record Order after Sorting mmddy	Actual Date after Sorting yyyy/mm/dd
0999999C	0000000C	non-date data
0102250C	0080321C	1921/08/03
0032120C	0102250C	1950/10/22
0010194C	0010194C	1994/01/01
0000000C	0111501C	2001/11/15
0111501C	0032120C	2020/03/21
0080321C	0999999C	non-date data

FIELDS=COPY (Required for a Copy)

Use FIELDS=COPY to copy one or more input files. (Multiple files can be copied if they are concatenated on the SORTIN DD specification.) Other control statements such as INCLUDE/OMIT, INREC, OUTREC and OUTFIL may be specified in conjunction with a copy application, allowing you to edit and reformat the file(s) without any collation processing

The SUM control statement and an E32 exit cannot be specified with FIELDS=COPY. All Phase 3 exits can be used.

MERGE

The SORTIN DD statement defines the input to be copied. (SORTINnn DD statements are not processed when FIELDS=COPY is specified.)

CKPT/CHKPT Parameter (Optional)

The CKPT/CHKPT parameter instructs SyncSort to take a checkpoint at every end-of-volume of a SORTOUT data set when OUTFIL is not used. Either spelling is accepted.

This parameter requires a SORTCKPT DD statement. It cannot be specified in conjunction with a user-issued STIMER macro or an incore sort. Checkpoints cannot be taken within a user exit routine.

Refer to “Chapter 13. Performance Considerations” for an explanation of the Checkpoint/Restart feature.

EQUALS/NOEQUALS Parameter (Optional)

The EQUALS parameter insures that equal-keyed records are merged in the order of their respective files. Equal-keyed records from the lowest numbered SORTINnn file are written before those from the second input file, etc. NOEQUALS, the default, specifies that equal-keyed records from different files be written in random order.

The order of equal-keyed records *within* each input file is always preserved during a merge, whether or not the EQUALS parameter is specified.

When the EQUALS parameter is used with the SUM control statement, the first of the equal-keyed records is retained with the sum; all other records are deleted after the specified field(s) have been summed.

EQUALS/NOEQUALS can also be specified as a PARM option on the EXEC statement. If this option is specified both on the MERGE control statement and as a PARM option, the MERGE specification takes precedence.

FILES Parameter (Optional)

The FILES=n parameter specifies the number of input files that an E32 exit will supply to the merge. The *n* value can be any number up to 100.

Specifying the FILES parameter both on the MERGE control statement and in the 24-bit parameter list will cause SyncSort to terminate with a critical error.

The FILES parameter *cannot* be specified as a PARM on the EXEC statement or in a \$ORTPARM data set.

SKIPREC Parameter (Optional)

The SKIPREC=*n* parameter instructs SyncSort to skip a decimal number of records before the input file is copied. The *n* records skipped are deleted from the input file before INCLUDE/OMIT processing, if specified, takes place.

The SKIPREC parameter can only be specified for a MERGE FIELDS=COPY operation. SKIPREC cannot be specified for a merge of multiple SORTIN*nn* data sets.

If SKIPREC is specified as a PARM option as well as on the MERGE control statement, the PARM specification takes precedence.

STOPAFT Parameter (Optional)

The STOPAFT=*n* parameter specifies the number of records to be copied. These will be the first *n* records after INCLUDE/OMIT and SKIPREC processing, if specified, have completed.

The STOPAFT parameter can only be specified for a MERGE FIELDS=COPY operation. STOPAFT cannot be specified for a merge of multiple SORTIN*nn* data sets.

If STOPAFT is specified as a PARM option as well as on the MERGE control statement, the PARM specification takes precedence.

Sample MERGE Control Statements

```
MERGE FIELDS=(1,5,CH,A,10,2,PD,D,30,4,BI,A)
```

Figure 21. Sample MERGE Control Statement

This sample MERGE control statement specifies three merge control fields:

- The first, or primary, control field begins in byte 1, is 5 bytes long, is in character format and is to be merged in ascending order.
- The second control field begins in byte 10, is 2 bytes long, is in packed decimal format and is to be merged in descending order.
- The third control field begins in the third bit of byte 30, is 4 bytes long, is in binary format and is to be merged in ascending order.

```
MERGE FIELDS=COPY,STOPAFT=200
```

Figure 22. Sample MERGE Control Statement

This MERGE statement specifies a copy operation. Only the first 200 records will be copied.

	Sort Phase 1	Sort Phase 2	Sort or Merge Phase 3	Copy
Exit	E11 E14 E15 E16 E17 E18	E21 E25 E27		E15
Name	E61		E31 E32 (merge only) E35 E37 E38 E39 E61	E31 E35 E37 E38 E39

Table 13. Phases and Permissible Exits

The exit-name parameter also provides the following information about the exit.

r The *r* value specifies the name of the user exit routine. Any valid name is acceptable. If the exit routine resides in a library, specify the member name or alias name for the *r* value. For an exit coded in REXX, *r* represents the REXX exec name.

b The *b* value specifies the exact or estimated decimal number of bytes the exit routine requires in main storage. This number should include any additional main storage required by the exit (e.g., buffers, GETMAINS, etc.). Specify an estimate (*without* an E before the value) if the exact number is not known. This number should only include storage requirements *below* the 16-megabyte line.

REXX exits have some additional storage requirements. REXX system modules and control blocks need 26K, and each EXEC that is called will require 12K of storage. In addition to any variables that the EXEC uses, all special SyncSort variables will require storage (including space for a record).

d The *d* value identifies the DD statement name that specifies the library in which the exit routine resides. The JCL must include a DD statement specifying each library in which an exit routine resides. If the exit routine is to be placed in the input job stream, specify SYSIN for the *d* value. (If more than one

MODS

exit routine is included in SYSIN, the exit routines must be specified in ascending numerical order by exit name.)

For a Disk Sort, MAXSORT or PARASORT, an exit routine that is a load module residing in a library identified in a LINKLIB, STEPLIB or JOBLIB DD statement does *not* require a *d* value specification or a DD statement defining a module library in the JCL. If the *d* value is omitted, insert a positional comma to indicate the missing value. For a Tape Sort, it *is* necessary to specify the LINKLIB, STEPLIB or JOBLIB DD statement and to include a DD statement defining the library.

The exit-name parameter also specifies link-editing codes: N, S, C, E, X, or T. If the link-editing code is omitted, the installation setting determines whether or not the exit will be link-edited. The delivered default is T; however, it may have been reset to N at installation.

Ideally, exit routines should be designed so that they do not require link-editing each time they are used. Link-editing consumes system resources and increases sort/merge execution time.

When a link-editing code is specified, the name E10 is reserved and no Phase 1 exit or E61 exit can use this name as a CSECT or ENTRY name. Similarly, the names E20 and E30 are reserved and cannot be used by Phase 2 or Phase 3 exits.

- | | |
|----------|---|
| N | The N value specifies that link-editing is not required. Link-editing has already taken place and SyncSort can directly invoke the routine. |
| S | The S value specifies that link-editing is required. This value can <i>only</i> be used for E11, E21 and E31 exits. The S value also indicates that the exit routine can be link-edited separately from other exit routines specified for the same phase. |
| C | The C value identifies a COBOL exit routine. COBOL exits must be link-edited before execution time. Only COBOL E15 and E35 exits can be specified, and COBOL exits cannot be specified for a Tape Sort. |
| E | The E value identifies a C exit routine. C exits must be link-edited before execution time. Only C E15 and/or E35 exits can be specified, and C exits cannot be specified for a tape sort. |
| X | The X value identifies a REXX exit routine. Only REXX E15 and E35 exits can be specified, and REXX exits cannot be specified for a Tape Sort. |
| T | The T value specifies that SyncSort will dynamically link-edit the exit routine along with other routines specified for the same sort/merge phase. |

Sample MODS Control Statement

```
MODS E15=(ADDREC1,600,MODLIB,N),E25=(ALTREC,500,SY SIN),  
E35=(ADDREC2,600,MODLIB,C)
```

Figure 24. Sample MODS Control Statement

This sample MODS control statement specifies the following information:

- An E15 exit is the first exit routine. ADDREC1 is the member name of the routine, which requires 600 bytes in main storage and resides in a library referenced by the DD statement named MODLIB. The routine does not require link-editing.
- An E25 exit is the second exit routine. ALTREC is the member name of the routine which requires 500 bytes in main storage. The exit is included in the SYSIN input stream. Because N is not specified, this routine will be link-edited.
- An E35 exit is the third exit routine. ADDREC2 is the member name of the routine, which requires 600 bytes in main storage and resides in a library referenced by the DD statement named MODLIB. This routine is a COBOL exit which has been link-edited before execution time.

Examples of JCL-initiated applications with exit routines are illustrated in “Chapter 4. JCL and Sample JCL/Control Statement Streams”.

OMIT

OMIT Control Statement

Refer to “INCLUDE/OMIT Control Statement” on page 2.16 for an explanation of the OMIT control statement.

OUTFIL Control Statement

The OUTFIL control statement describes the output file(s). It is required to accomplish these three tasks:

- Create multiple output files. The OUTFIL parameters associated with this task are FILES, FNAMES, INCLUDE/OMIT, STARTREC, ENDREC, SAVE, OUTREC, CONVERT, VLFILL, FTOV, VLTRIM, NULLOFL, and SPLIT.
- Use the SortWriter facility. The OUTFIL parameters associated with this task are HEADER1, HEADER2, LINES, NODETAIL, REMOVECC, SECTIONS, TRAILER1, and TRAILER2.
- Reformat records after E35 processing. The OUTFIL parameter associated with this task is OUTREC.

The OUTFIL control statement cannot be used with MAXSORT.

OUTFIL Control Statement Format

The format for the OUTFIL control statement is illustrated below.

OUTFIL

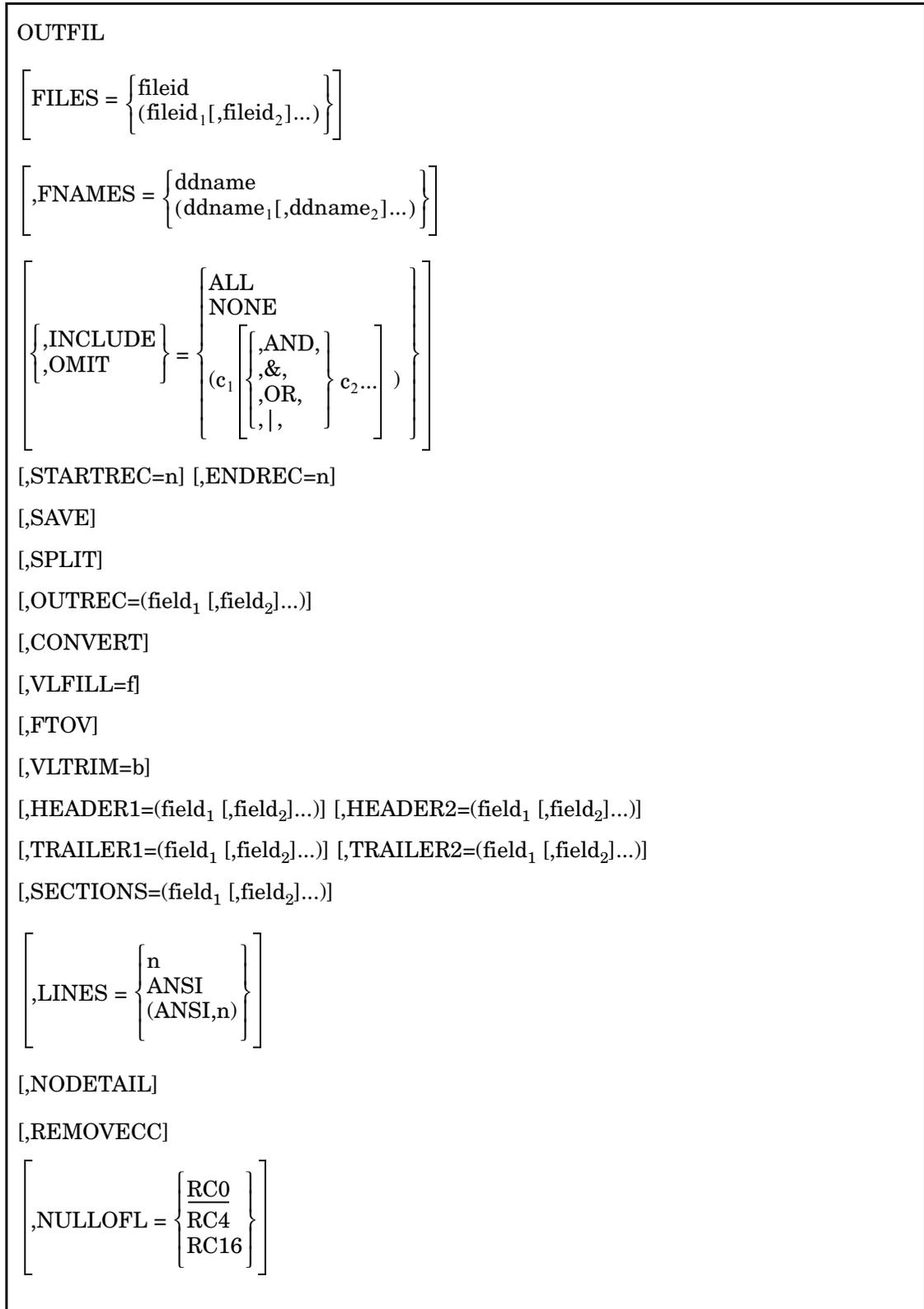


Figure 25. OUTFIL Control Statement Format

The Multiple Output Capability

Use the OUTFIL control statement to create multiple files *without* making multiple passes through the input data. The output files can be treated the same or differently:

- The output files can contain the same or different records.
- The records in the output files can be identically or differently formatted.
- If the input file(s) are variable-length, the output files may be either variable-length or fixed-length.

Note that all the output files will be sequenced in the same way, as specified on the SORT or MERGE control statement. If you need to sort the output files differently, you should use PipeSort, a Syncsort product that works with SyncSort for z/OS to reduce total elapsed time by generating multiple, differently sequenced output files from a single read of the input data.

The SortWriter Capability

The SortWriter capability of OUTFIL can produce completely formatted reports. The report writing features, which can be specified differently for each output file, can accomplish these tasks:

- Arrange the report into pages.
- Divide the report into sections.
- Format headers and trailers for sections, pages, and the complete report.
- Create multiple lines of output from each input record.
- Convert and edit numeric data.
- Provide TOTAL and SUBTOTAL capabilities for data fields in a specific part of a report.
- Provide MIN, MAX, AVERAGE, SUBMIN, SUBMAX, and SUBAVG capabilities for data fields in a specific part of a report.
- Provide COUNT and SUBCOUNT capabilities for records in a specific part of a report.

Once formatted, output files can be assigned to any tape, disk, or unit record device for subsequent printing.

OUTFIL

FILES Parameter (Optional)

The FILES parameter connects the OUTFIL control statement with one or more output files. The files specified on this parameter, along with any specified on the F NAMES parameter, will constitute the ddnames to receive output for this OUTFIL specification.

The format of the FILES parameter is illustrated in the following figure.

$$\text{FILES} = \left\{ \begin{array}{l} \text{fileid} \\ (\text{fileid}_1, [\text{fileid}_2] \dots) \end{array} \right\}$$

where:

$$\text{fileid} = \left\{ \begin{array}{l} \text{OUT} \\ \text{x} \\ \text{xx} \end{array} \right\}$$

Figure 26. FILES Parameter Format

The fileid identifies the output file and connects the OUTFIL control statement with the corresponding SORTOUT, SORTOFx, or SORTOFxx DD statement. For example, FILES=OUT connects the OUTFIL control statement with the SORTOUT DD statement. Similarly, FILES=1 connects the OUTFIL control statement with the SORTOF1 DD statement, and FILES=01 connects the OUTFIL control statement with the SORTOF01 DD statement. The *x* can be any alphanumeric character or special character allowed by JCL DD statements.

If multiple output files have identical specifications (that is, identical record selection, record reformatting, and report writing specifications), the FILES and/or F NAMES parameter can connect the OUTFIL control statement with more than one DD statement. For example, FILES=(OUT,02,03) connects the OUTFIL control statement with the SORTOUT, SORTOF02, and SORTOF03 DD statements. Such a set of output files is termed an *OUTFIL group*.

If multiple output files have different specifications, then each file is specified on a separate OUTFIL control statement with one FILES and/or F NAMES parameter on each control statement.

If a SORTOUT ddname is defined in the JCL and does not appear in any FILES or F NAMES specification, it will be written to without any OUTFIL processing. If an inline E35 exit has been specified, OUTFIL is ignored.

If neither a FILES nor F NAMES parameter is specified on an OUTFIL control statement, the default ddname of SORTOUT will be used. If a 4-byte ddname prefix is in effect, the default SORTOUT ddname will be ppppOUT, where pppp is the prefix; adding FILES=xx would connect to the ppppOFxx DD statement.

FNAMES Parameter (Optional)

The FNAMES parameter connects the OUTFIL control statement with one or more output files. The files specified on this parameter, along with any specified on the FILES parameter, will constitute the ddnames to receive output for this OUTFIL specification.

The format of the FNAMES parameter is illustrated in the following figure.

$$\text{FNAMES}=\left\{ \begin{array}{l} \text{ddname} \\ (\text{ddname}_1 [\text{,ddname}_2]\dots) \end{array} \right\}$$

Figure 27. FNAMES Parameter Format

ddname is a 1 to 8-character ddname that corresponds to a DD statement provided in the JCL.

If multiple output files have identical specifications (that is, identical record selection, record reformatting, and report writing specifications), the FNAMES and/or FILES parameter can connect the OUTFIL control statement with more than one DD statement. For example, FNAMES=(FILE1OUT,FILE2OUT,FILE3OUT) connects the OUTFIL control statement with the three listed DD statements. Such a set of output files is termed an *OUTFIL group*.

If multiple output files have different specifications, then each file is specified on a separate OUTFIL control statement with one FNAMES and/or FILES parameter on each control statement.

If a SORTOUT ddname is defined in the JCL and does not appear in any FILES or FNAMES specification, it will be written to without any OUTFIL processing. If an inline E35 exit has been specified, OUTFIL is ignored.

If neither a FILES nor FNAMES parameter is specified on an OUTFIL control statement, the default ddname of SORTOUT will be used. If a 4-byte ddname prefix is in effect, the default SORTOUT ddname will be ppppOUT, where pppp is the prefix.

INCLUDE/OMIT Parameter (Optional)

Specify the INCLUDE or OMIT parameter to indicate which records are to be included in or omitted from each output file. These parameters let you create multiple output files which contain different records. The default is to include all sorted or merged records in the output file.

The format for the INCLUDE/OMIT parameter is illustrated below:

OUTFIL

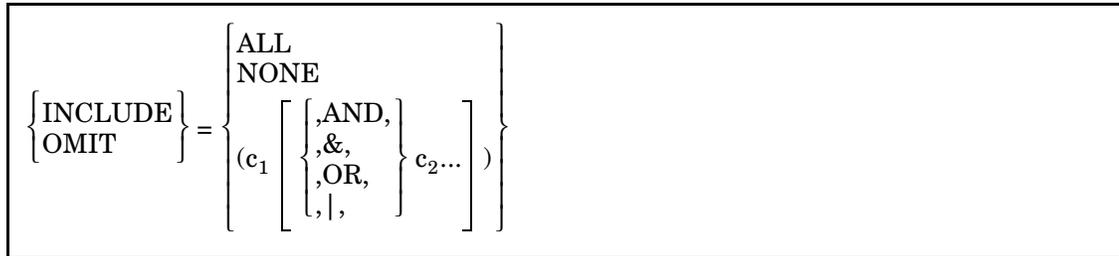


Figure 28. INCLUDE/OMIT Parameter Format

See “INCLUDE/OMIT Control Statement” on page 2.16 for the detailed format of a comparison. The FORMAT=f parameter, which is permitted for the INCLUDE/OMIT control statement, is not permitted for the INCLUDE/OMIT parameter. Field formats must be specified on a field-by-field basis.

The comparison determines which records are included or omitted. When *no* data records are to be included in the output file(s) (when running a test, for example), specify either INCLUDE=NONE or OMIT=ALL.

Note: The location within the data records of the fields specified in the INCLUDE/OMIT parameter will be based on the formatting of the record *after* processing by an E15/E32 exit, the INREC control statement, the OUTREC control statement, and an E35 exit, but *before* processing due to the OUTREC and/or report writing parameters of the OUTFIL control statement.

The following four parameters (STARTREC, ENDREC, SAVE, and SPLIT) are related to the previous parameter (INCLUDE/OMIT) in that they specify records to be included for OUTFIL processing. However, these four options specify records in bulk rather than through a comparison condition.

STARTREC Parameter (Optional)

Use the STARTREC=n parameter to specify the record number *n* of the first record to be processed by the OUTFIL specification in effect. All records prior to the specified record will be ignored for the OUTFIL group. The record number is determined by the sequence of records presented for OUTFIL processing.

ENDREC Parameter (Optional)

Use the ENDREC=n parameter to specify the record number *n* of the last record to be processed by the OUTFIL specification in effect. All records after the specified record will be ignored for the OUTFIL group. The record number is determined by the sequence of records presented for OUTFIL processing.

SAVE Parameter (Optional)

Use SAVE to include records for OUTFIL processing that have not been included in any other OUTFIL group.

If SAVE is specified on more than one OUTFIL group, then each of these OUTFIL groups get the records that were discarded from all other OUTFIL groups that do not have SAVE.

The OUTFIL INCLUDE/OMIT parameter is mutually exclusive with the SAVE parameter. Only one of these parameters can be specified for an OUTFIL group.

Note that if the SORTOUT data set has not been associated with any OUTFIL control statement but is present in the JCL, the SORTOUT data set will receive a copy of all records prior to OUTFIL processing. This does not affect the SAVE operation, since SAVE is only pertinent to other OUTFIL group specifications.

OUTREC Parameter (Optional)

The OUTREC parameter indicates how the records are to be formatted in each output file. This parameter lets you create multiple output files which contain differently formatted records.

When the records in all multiple output files are formatted and edited identically, it is more efficient to specify a single OUTREC control statement rather than several OUTREC parameters.

The OUTREC parameter reformats the records that are to be included in the output file(s) after E35 processing, if specified. If no additional reformatting is required, omit this parameter.

All references to field positions specified in the OUTREC parameter refer to the record *after* processing by an E15 exit, the INREC control statement, the OUTREC control statement, and an E35 exit but *before* insertion of ANSI control characters.

The format of the OUTREC parameter is illustrated below.

```
OUTREC=(field1[,field2]...)
```

Figure 29. OUTREC Parameter Format

The format of the OUTFIL OUTREC parameter is generally identical to the format of the FIELDS parameter of the OUTREC control statement. (See the subsections dealing with the FIELDS parameter in “OUTREC Control Statement” on page 2.88.) Note, however, that FIELDS= is not used with OUTFIL OUTREC. In addition, OUTFIL OUTREC accepts two subparameters that cannot be specified on the OUTREC control statement:

OUTFIL

[n]/ The / subparameter indicates the end of a line and can be used to create multiple output lines from a single input record. Multiple slashes (coded // .../ or n/) can be used to specify leading, trailing, or embedded blank lines. At the beginning or end of the OUTREC parameter, n/ produces n blank lines. Embedded within the OUTREC parameter, n/ produces n-1 blank lines.

The / subparameter is most useful for its ability to accommodate records whose lengths exceed the width of the physical page. For an example of the / subparameter, see “Printing Input Records on Multiple Output Lines” on page 3.28.

The / subparameter may not be used when LINES=ANSI or LINES=(ANSI,n) has also been specified on the OUTFIL control statement.

VLFILL=f The VLFILL parameter is used in conjunction with OUTREC or OUTREC CONVERT to specify a fill byte to be used for any missing p,l field bytes.

The VLFILL parameter has two functions:

- It enables a variable-length OUTFIL OUTREC non-CONVERT application to continue processing when there is an input record with missing field bytes in a p,l field specification.
- It provides a means to override the default fill byte used in an OUTFIL OUTREC CONVERT application when there are missing bytes in a p,l field specification.

In the first instance, if VLFILL has not been specified the application will terminate with the critical error WER244A. In the second case, by default, spaces will be used for missing field bytes.

f specifies a byte to be used for missing field bytes. *f* can be specified as either a character or hexadecimal value. Specify either C'*x*' where *x* is a single EBCDIC character or X'*hh*' where *hh* represents a hexadecimal digit pair (00-FF).

Note: If VLFILL is specified, the OUTREC parameter must also be specified. VLFILL is ignored when the FTOV parameter is used.

FTOV Parameter (Optional)

The FTOV parameter converts fixed-length input records to variable-length output records.

FTOV can be used both with and without the OUTREC parameter. When FTOV is used with the OUTREC parameter, the variable-length record is created from the specified fields

of the fixed-length record. When FTOV is not used with the OUTREC parameter, the variable-length record is created from the whole fixed-length record.

Notes: FTOV cannot be used with CONVERT. If the input record is variable-length, FTOV, if specified, will be ignored. FTOV can be used with the VLTRIM parameter to delete pad bytes at the end of a record.

For an example of an OUTFIL control statement that uses the FTOV parameter, see Figure 41 on page 2.87.

VLTRIM Parameter (Optional)

The VLTRIM parameter defines a byte to be deleted from the end of a variable-length record. All prior occurrences of this byte will also be deleted until a byte that is not equal to the trim byte is found. The resulting records are decreased in record length. However, VLTRIM will not delete the first data byte, the ANSI carriage control character, or the Record Descriptor Word (RDW).

The format of the VLTRIM parameter is illustrated below.

VLTRIM=b

b specifies the byte to be deleted from the end of the record. *b* can be specified as either a character or hexadecimal value. Specify either C'*x*' where *x* is a single EBCDIC character or X'*hh*' where *hh* represents a hexadecimal digit pair (00-FF).

Note: VLTRIM is ignored if used with fixed-length output records.

For an example of an OUTFIL control statement that uses the VLTRIM parameter, see Figure 41 on page 2.87.

CONVERT Parameter (Optional)

The CONVERT parameter is used in conjunction with the OUTREC parameter to convert variable-length records to fixed-length records.

The records do not require an RDW and will be written to the output file(s) with a RECFM of F or FB. When using CONVERT, you no longer need to apply the rules for “Specifying the FIELDS parameter for Variable-Length Records” found in the description of the OUTREC control statement.

You cannot specify the variable portion of the input records (position without length) when using CONVERT. All other p,l data fields that are not present will be filled with blanks by default. The OUTFIL VLFILL parameter can be used to specify a different fill byte for any missing fields (see above description).

OUTFIL

Notes: If CONVERT is specified, the OUTREC parameter must also be specified. CONVERT cannot be used with the FTOV parameter.

SPLIT Parameter (Optional)

The SPLIT parameter of the OUTFIL control statement causes output records to be distributed in rotation among files in an OUTFIL group.

In the normal case, when the SPLIT parameter is not used, the output files in the group will contain the same records. SPLIT distributes the output records. The following OUTFIL control statement will distribute records among three output files:

```
OUTFIL FILES=(01,02,03),SPLIT
```

Figure 30. Sample OUTFIL Control Statement with SPLIT

For the above example, the first record will be written to the SORTOF01 data set; the second, to SORTOF02; the third, to SORTOF03. The fourth record will be written to SORTOF01 again, and so on in round-robin fashion.

The OUTFIL control statement can contain an INCLUDE/OMIT and an OUTREC parameter, in which case the selected and reformatted subset of records will be distributed among the output files.

Note that the SPLIT parameter cannot be used with any report writing (SortWriter) functions. Specifically, report writing parameters (HEADER_n, TRAILER_n, SECTIONS, LINES, NODETAIL) cannot be specified on the OUTFIL control statement that defines the output group.

SPLIT can be used with BatchPipes/MVS; that is, the output records can be distributed among BatchPipes/MVS data sets.

HEADER1/HEADER2 Parameters (Optional)

The SortWriter facility provides three types of headers:

- HEADER1, the report header
- HEADER2, the page header
- HEADER3, the section header.

HEADER1 and HEADER2 are parameters of the OUTFIL control statement. HEADER3 is a subparameter of OUTFIL's SECTIONS parameter. Refer to "SECTIONS Parameter (Optional)" on page 2.80 for an explanation of how to specify HEADER3.

The three types of headers function independently of each other. Each serves a different purpose.

- **HEADER1** provides a header or a possible title page for the entire report. It appears only once at the beginning of the report on its own page.
- **HEADER2** provides a page header or a running head for each page defined by the **LINES** parameter. It appears at the beginning or top of each page.
- **HEADER3** provides a section header that appears at the beginning of each specified section and, optionally, at the top of each page (or directly below any **HEADER2**).

The chart below illustrates the format for **HEADERS**. The field entries represent the subparameters that can be specified for each **HEADER** entry.

```

HEADER1=(field1[,field2]...)
HEADER2=(field1[,field2]...)
HEADER3=(field1[,field2]...)
    
```

Figure 31. HEADER Parameter Format

The following *HEADER Subparameters Format* chart illustrates and defines the available subparameters. Each subparameter constitutes a separate field of the **HEADER**.

```

[ c: ] {
    [n] X
    [n] 'literal string'
    [n] /
    p,l
    &DATE
    &DATE=(m1m2m3m4)
    &DATENS=(abc)
    &TIME
    &TIME=(hp)
    &TIMENS=(tt)
    &PAGE
}
    
```

Figure 32. HEADER Subparameter Format

- c:** Use the *c:* subparameter to define the column in which the specified field should begin.
- n** Used in conjunction with the **X**, 'literal string', and / subparameters, the *n* value defines the number (1-4095) of repetitions for each entry.
- X** Use the **X** subparameter to define the number of spaces. It must be coded to the immediate right of the *n* value, if specified. For

OUTFIL

more than 4095 spaces, two or more nX values should be specified.

'literal string'

Use the 'literal string' subparameter to define a literal string. Specify the number of repetitions by coding *n* immediately before it. An apostrophe within a literal string must be specified as a double apostrophe; for example, C 'O"Leary'.

/

Use the / subparameter to indicate the end of a line, force a carriage return, and separate text lines of a header. Multiple slashes (//.../ or *n/*) can be used to specify leading, trailing, or embedded blank lines. At the beginning or end of a header, *n/* produces *n* blank lines. Within a header, *n/* produces *n*-1 blank lines.

p,l

Use the *p* and *l* subparameters to include a field (or fields) within a record in the header. For a HEADER1, the field(s) will be extracted from the first record in a file; for a HEADER2, the field(s) will be extracted from the first record on a page; for a HEADER3, the field(s) will be extracted from the first record in a section. *p* is the starting position of the field in the record; *l* is the length in bytes (1-255) of the field. Any number of fields can be specified. (Contiguous fields within a record can be specified with a single *p,l* entry, but their combined length cannot exceed 255 bytes.) The specified field(s) should be a character or alphanumeric string or a number in printable format, and the field(s) cannot be converted or edited.

&DATE

The &DATE subparameter specifies the current system date and requires 8 bytes to display *mm/dd/yy*.

&DATE=(*m*₁*m*₂*m*₃*m*₄)

This form of the &DATE subparameter generates the current system date and controls the formatting of the date. You can specify the position of the year, month, and date, specify a separator character, and choose between 2-digit and 4-digit year representation.

The positions *m*₁ through *m*₄ represent masks used to format the date. To specify the position of the month, day, and year, replace the *m*₁, *m*₂, and *m*₃ positions, in any order, with M for the month (01-12), D for the day (01-31), and either Y or 4 for the year (where Y is a 2-digit year and 4 is a 4-digit year). Replace the *m*₄ position with a separator character.

For example, to print the date with the form *yy-mm-dd*, specify &DATE=(YMD-). For December 31, 1999, the date would appear as "99-12-31".

A blank used as the separator character must be enclosed in apostrophes. An apostrophe used as the separator character must be specified as two apostrophes enclosed within apostrophes ("").

The field for this form of &DATE requires 8 bytes for a 2-digit year representation and 10 bytes for a 4-digit year. The M, D, and Y or 4 may only appear once in the mask. All four positions must be specified.

&DATENS=(xyz)

specifies that the current date is to appear in the report record in the form 'xyz', where x, y, and z indicate the order in which the month, day, and year are to appear and whether the year is to appear as two or four digits. For x, y, and z, use M to represent the month (01-12), D to represent the day (01-31), Y to represent the last two digits of the year (for example, 02), or 4 to represent the four digits of the year (for example, 2002). M, D, and Y or 4 can each be specified only once.

For example, &DATENS=(DMY) would produce a date of the form 'ddmmyy' which on March 29, 2002, would appear as '290302'. &DATENS=(4MD) would produce a date of the form 'yyyymmdd' which on March 29, 2002, would appear as '20020329'. x, y, and z must be specified.

&TIME

The &TIME subparameter specifies the current time of day and requires 8 bytes to display *hh:mm:ss*, where *hh* is in 24-hour format.

&TIME=(hp)

This form of the &TIME subparameter generates the current system time of day and controls the formatting of the time. You can print the time in 24-hour or 12-hour formats and specify the separator character between the hours, minutes, and seconds.

The format for 24-hour time is *hhpmpss*, where *hh* represents the hour (00-23), *mm* represents minutes (00-59), *ss* represents seconds (00-59), and *p* represents the separator character as specified by *p* in the &TIME=hp subparameter.

The format for 12-hour time is *hhpmpss nn*, where *hh* represents the hour (01-12), *mm* represents minutes (00-59), *ss* represents seconds (00-59), and *p* represents the separator character as specified by *p* in the &TIME=hp subparameter. The *nn* is "am" or "pm" as appropriate.

To select 12-hour mode specify *h* as 12; to select 24-hour mode specify *h* as 24. The *p* specification represents the character to use as a separator.

For example, to display the time in a 12-hour format with a period as a separator, specify &TIME=(12.). At 22:43:23 hours, the time would appear as "10.43.23 pm".

A blank used as the separator character must be enclosed in apostrophes. An apostrophe used as the separator character

OUTFIL

must be specified as two apostrophes enclosed within apostrophes ('').

The field for this form of the &TIME subparameter requires 8 bytes for the 24-hour format and 11 bytes for the 12-hour format.

&TIMENS=(tt)

specifies that the current time is to appear in the report record in the form 'hhmmss' (24-hour time) or 'hhmmss xx' (12-hour time). If tt is 24, the time is to appear in the form 'hhmmss' (24-hour time) where hh represents the hour (00-23), mm represents the minutes (00-59), and ss represents the seconds (00-59).

For example, &TIMENS=(24) would produce a time of the form 'hhmmss' which at 08:25:13 pm would appear as '202513'. If tt is 12, the time is to appear in the form 'hhmmss xx' (12-hour time) where hh represents the hour (01-12), mm represents the minutes (00-59), ss represents the seconds (00-59), and xx is either 'am' or 'pm'.

For a second example, &TIMENS=(12) would produce a time of the form 'hhmmss xx' which at 08:25:13 pm would appear as '082513 pm'.

&PAGE

The &PAGE subparameter sequentially numbers logical pages of the output report and requires 6 bytes. It produces a 6-digit sequential page number, right justified with leading zeros suppressed. &PAGE is ignored for HEADER1.

Rules for Specifying HEADER Subparameters

Observe the following guidelines when you specify HEADER subparameters:

- Separate subparameters with commas, except between *c*: and another subparameter. Commas are optional for the / subparameter.
- Enclose literals in single quotes.
- Specify blank fields of *n* bytes as *nX*.
- Headings specified with fewer blanks than the logical record length (LRECL) of the output record are automatically padded on the right with blanks.
- If a heading exceeds the logical record length (LRECL) of the output record, use the OUTREC control statement or the OUTREC parameter to expand the output record length so that it is at least as long as the longest header. For example, if the longest header is 115 characters and the output record length is 80 bytes, use the OUTREC

control statement or the OUTREC parameter to insert a blank in position 115 of the output record. This will cause bytes 81 through 115 to be padded with blanks.

TRAILER Parameters (Optional)

The SortWriter facility provides three types of trailers:

- TRAILER1, the report trailer
- TRAILER2, the page trailer
- TRAILER3, the section trailer.

TRAILER1 and TRAILER2 are parameters of the OUTFIL control statement; TRAILER3 is a subparameter of OUTFIL's SECTIONS parameter. Refer to "SECTIONS Parameter (Optional)" on page 2.80 for an explanation of how to specify TRAILER3.

The three types of trailers function independently of each other. Each serves a different purpose:

- TRAILER1 provides a trailer or a possible summary for the entire report. It appears only once at the end of the report on its own page.
- TRAILER2 provides a page trailer for each page defined by the LINES parameter. It appears at the end of each page.
- TRAILER3 provides a section trailer that appears at the end of each specified section and serves as a conclusion or summary for that section.

TRAILER1, TRAILER2, and TRAILER3 also provide TOTAL, SUBTOTAL, MIN, SUBMIN, MAX, SUBMAX, AVG, SUBAVG, COUNT, SUBCOUNT, COUNT15, and SUBCOUNT15 capabilities at report, page, and section levels.

The chart below illustrates the format for TRAILERs. Its field entries represent the subparameters that can be specified for each TRAILER entry.

```
TRAILER1=(field1[,field2]...)
TRAILER2=(field1[,field2]...)
TRAILER3=(field1[,field2]...)
```

Figure 33. TRAILER Parameter Format

The following *TRAILER Subparameters Format* chart illustrates and defines the available subparameters. Each subparameter constitutes a separate field of the TRAILER.

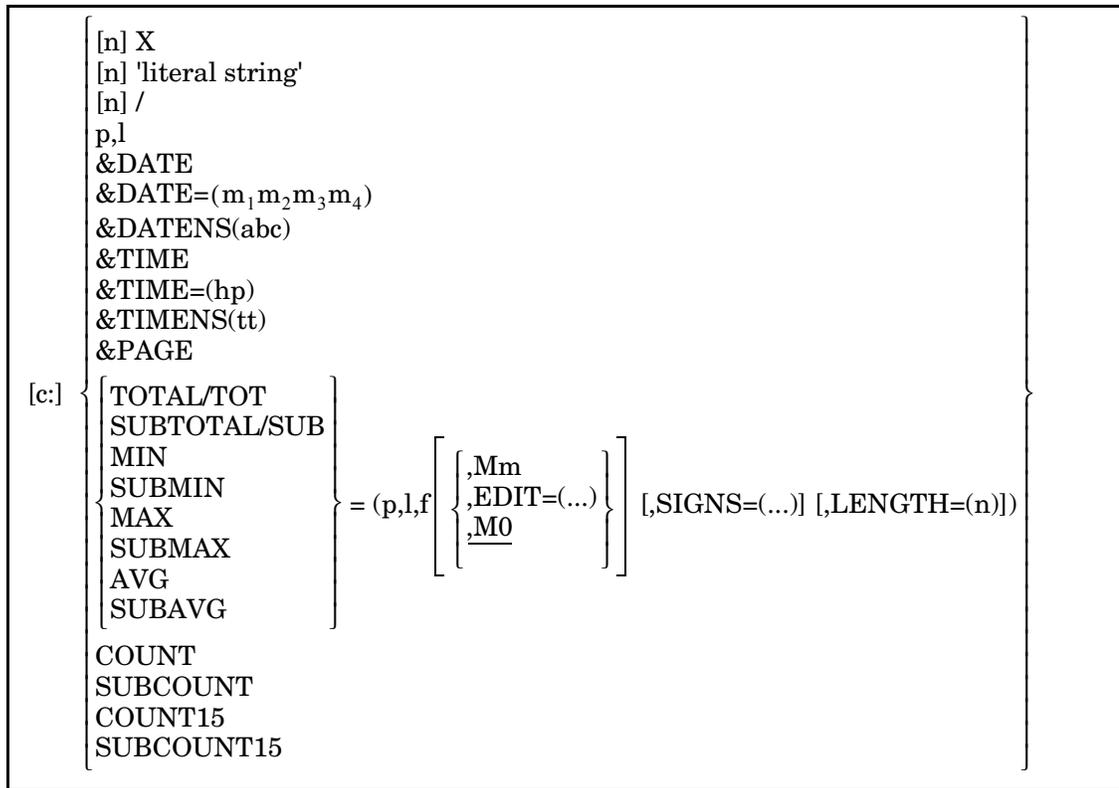


Figure 34. TRAILER Subparameters Format

- c:** Use the *c:* subparameter to define the column in which the specified field should begin.
- n** Used in conjunction with the X, 'literal string', and / subparameters, the *n* value defines the number (1-4095) of repetitions for each entry.
- X** Use the X subparameter to define the number of spaces. It must be coded to the immediate right of the *n* value, if specified. For more than 4095 spaces, two or more nX values should be specified.
- 'literal string'** Use the 'literal string' subparameter to define a literal string. Specify the number of repetitions by specifying *n* immediately before it. An apostrophe within a literal string must be specified as a double apostrophe; for example, C 'O"Leary'.
- /** Use the / subparameter to indicate the end of a line, force a carriage return, and separate text lines of a trailer. Multiple slashes (coded //.../ or *n/*) can be used to specify leading, trailing, or embedded blank lines. At the beginning or ending of a trailer,

n/ produces *n* blank lines. Within a trailer, *n/* produces *n*-1 blank lines.

p,l

Use the *p* and *l* subparameters to include a field (or fields) within a record in the trailer. For a TRAILER1, the field(s) will be extracted from the last record in a file; for a TRAILER2, the field(s) will be extracted from the last record on a page; for a TRAILER3, the field(s) will be extracted from the last record in a section. *p* is the starting position of the field in the record; *l* is the length in bytes (1-255) of the field. Any number of fields can be specified. (Contiguous fields within a record may be specified with a single *p,l* entry, but their combined length may not exceed 255 bytes.) The specified field(s) should be a character or alphanumeric string, or a number in printable format, and the field cannot be converted or edited.

If any variable-length record contains only a portion of the bytes in a specified field, those bytes will be included in the trailer and blanks will be substituted for the missing bytes.

&DATE

The &DATE subparameter specifies the current system date and requires 8 bytes to display *mm/dd/yy*.

&DATE=(m₁m₂m₃m₄)

This form of the &DATE subparameter generates the current system date and controls the formatting of the date. You can specify the position of the year, month, and date, specify a separator character, and choose between 2-digit and 4-digit year representation.

The positions *m*₁ through *m*₄ represent masks used to format the date. To specify the position of the month, day, and year, replace the *m*₁, *m*₂, and *m*₃ positions, in any order, with M for the month (01-12), D for the day (01-31), and either Y or 4 for the year (where Y is a 2-digit year and 4 is a 4-digit year). Replace the *m*₄ position with a separator character.

For example, to print the date with the form *yy-mm-dd*, specify &DATE=(YMD-). For December 31, 1999, the date would appear as "99-12-31".

The field for this form of &DATE requires 8 bytes for a 2-digit year representation and 10 bytes for a 4-digit year. The M, D, and Y or 4 may only appear once in the mask. All four positions must be specified.

&DATENS=(xyz)

specifies that the current date is to appear in the report record in the form 'xyz', where x, y, and z indicate the order in which

OUTFIL

the month, day, and year are to appear and whether the year is to appear as two or four digits. For *x*, *y*, and *z*, use *M* to represent the month (01-12), *D* to represent the day (01-31), *Y* to represent the last two digits of the year (for example, 02), or 4 to represent the four digits of the year (for example, 2002). *M*, *D*, and *Y* or 4 can each be specified only once.

For example, `&DATENS=(DMY)` would produce a date of the form 'ddmmyy' which on March 29, 2002, would appear as '290302'. `&DATENS=(4MD)` would produce a date of the form 'yyyymmdd' which on March 29, 2002, would appear as '20020329'. *x*, *y*, and *z* must be specified.

&TIME

The `&TIME` subparameter specifies the current time of day and requires 8 bytes to display *hh:mm:ss*, where *hh* is in 24-hour format.

&TIME=(hp)

This form of the `&TIME` subparameter generates the current time of day and controls the formatting of the time. You can print the time in 24-hour or 12-hour formats and specify the separator character between the hours, minutes, and seconds.

The format for 24-hour time is *hhpmpss*, where *hh* represents the hour (00-23), *mm* represents minutes (00-59), *ss* represents seconds (00-59), and *p* represents the separator character as specified by *p* in the `&TIME=hp` subparameter.

The format for 12-hour time is *hhpmpss nn*, where *hh* represents the hour (01-12), *mm* represents minutes (00-59), *ss* represents seconds (00-59), and *p* represents the separator character as specified by *p* in the `&TIME=hp` subparameter. The *nn* is “am” or “pm” as appropriate.

To select 12-hour mode specify *h* as 12; to select 24-hour mode specify *h* as 24. The *p* specification represents the character to use as a separator.

For example, to display the time in a 12-hour format with a period as a separator, specify `&TIME=(12.)`. At 22:43:23 hours, the time would appear as “10.43.23 pm”.

The field for this form of the `&TIME` subparameter requires 8 bytes for the 24-hour format and 11 bytes for the 12-hour format.

&TIMENS=(tt)

specifies that the current time is to appear in the report record in the form 'hhmmss' (24-hour time) or 'hhmmss xx' (12-hour

time). If `tt` is 24, the time is to appear in the form 'hhmmss' (24-hour time) where `hh` represents the hour (00-23), `mm` represents the minutes (00-59), and `ss` represents the seconds (00-59).

For example, `&TIMENS=(24)` would produce a time of the form 'hhmmss' which at 08:25:13 pm would appear as '202513'. If `tt` is 12, the time is to appear in the form 'hhmmss xx' (12-hour time) where `hh` represents the hour (01-12), `mm` represents the minutes (00-59), `ss` represents the seconds (00-59), and `xx` is either 'am' or 'pm'.

For a second example, `&TIMENS=(12)` would produce a time of the form 'hhmmss xx' which at 08:25:13 pm would appear as '082513 pm'.

&PAGE

The `&PAGE` subparameter sequentially numbers logical pages of the output report and requires 6 bytes. It produces a 6-digit sequential page number, right justified with leading zeros suppressed.

TOTAL/TOT

Use the `TOTAL` subparameter to specify that numeric data are to be accumulated and totaled at the end of a report, logical page, or section.

After including the results in the appropriate trailer, the accumulator resets to zero. `TOTALs` appear in printable format. If a `SyncSort` editing mask is used for totaled data, the length of the output field is determined by the maximum permissible length of the data format, not by the specified length of the input field. This means that the default is to display 10 digits for `BI` and `FI` fields and 15 digits for `PD`, `ZD`, and `CSF` or `FS` fields. Internally, `SyncSort` maintains up to 15 digits for all data formats. Thus, if a `BI` or `FI` field total could exceed 10 digits, you should specify the `LENGTH` and/or `EDIT` subparameters to override the length of the output field.

SUBTOTAL/SUB

Use the `SUBTOTAL` subparameter to generate a running total of a field at the end of a report, logical page, or section. This subparameter functions like the `TOTAL` subparameter except the accumulator does not reset to zero. `SUBTOTALs` appear in printable format. If a `SyncSort` editing mask is used for subtotaled data, the length of the output field is determined by the maximum permissible length of the data format, not by the specified length of the input field. This means that the default is to display 10 digits for `BI` and `FI` fields and 15 digits for `PD`, `ZD`, and `CSF` or `FS` fields. Internally, `SyncSort` maintains up to

OUTFIL

15 digits for all data formats. Thus, if a BI or FI field total could exceed 10 digits, you should specify the LENGTH and/or EDIT subparameters to override the length of the output field.

- MIN** Use the MIN subparameter to obtain the minimum numeric value of an input field for all records within the report, logical page, or section. This value will be displayed in printable format.
- SUBMIN** Use the SUBMIN subparameter to obtain the running minimum numeric value of an input field for all records within the report up to the point of the TRAILER. This value will be displayed in printable format.
- MAX** Use the MAX subparameter to obtain the maximum numeric value of an input field for all records within the report, logical page, or section. This value will be displayed in printable format.
- SUBMAX** Use the SUBMAX subparameter to obtain the running maximum numeric value of an input field for all records within the report up to the point of the TRAILER. This value will be displayed in printable format.
- AVG** Use the AVG subparameter to obtain the average numeric value of an input field for all records within the report, logical page, or section. This value will be displayed in printable format.
- SUBAVG** Use the SUBAVG subparameter to obtain the running average numeric value of an input field for all records within the report up to the point of the TRAILER. This value will be displayed in printable format.
- p** Use the *p* subparameter to indicate the position of the first byte of the numeric field.
- l** Use the *l* subparameter to indicate the length of the numeric field. Permissible lengths are 1-4 bytes for BI or FI, 1-8 bytes for PD, 1-15 bytes for ZD, and 1-16 for CSF or FS with a 15-digit limit. To determine the length of the output field, refer to “How to Convert Numeric Data” on page 2.97.

For the (SUB)TOTAL and (SUB)AVG functions, fields are totaled internally as 8-byte PD fields. An overflow condition will occur if the positive or negative value of a totaled or subtotaled field exceeds the value that can be represented by such fields, and the execution will terminate with an error message.

f	Use the <i>f</i> subparameter to indicate the format of the numeric field. Replace <i>f</i> with BI, FI, PD, ZD, CSF, or FS.
Mm	Use the <i>Mm</i> subparameter to indicate that one of the 27 SyncSort-supplied masks (M0-M26) should be used to format a field. Replace <i>m</i> with the mask number. The default is M0. For details, refer to “The Mm Subparameter (Editing Masks)” on page 2.109.
EDIT=(pattern)	Use the EDIT=(pattern) subparameter to indicate that a user-provided editing mask should be used to format a field. For details, refer to “Converting SMF Date and Time Formats” on page 2.102.
SIGNS=(...)	Use the SIGNS subparameter to specify leading and/or trailing signs that will appear before or after the edited number. For details, refer to “The SIGNS Subparameter” on page 2.112.
LENGTH=(n)	Use the LENGTH subparameter to alter the length of a field determined by the edit pattern and the internal field format. For details, refer to “The LENGTH=n Subparameter” on page 2.108.
COUNT	Use the COUNT subparameter to obtain a count of the number of records in either the entire report or a specific part of the report. In a TRAILER1, this field will contain a count of the total number of data records in the report. In a TRAILER2, it will contain a count of the number of data records on each page. In a TRAILER3, it will contain a count of the number of data records in each section. The count will be the number of data records <i>before</i> any multi-line OUTREC processing has been done. This number will be a right-justified 8-digit field with leading zeros suppressed. The maximum value is 99999999.
COUNT15	This subparameter is identical to the COUNT subparameter except for the allowable size of the count number. For COUNT15 the number will be a right-justified <i>15-digit</i> field with leading zeros suppressed. The maximum value is 999999999999999.
SUBCOUNT	Use the SUBCOUNT subparameter to obtain a running, or cumulative, count of the number of records throughout a report. In a TRAILER1, this field will contain a count of the total number of data records in the report. In a TRAILER2, it will contain a cumulative count of the number of data records on a page-by-page basis. In a TRAILER3, it will contain a cumulative count of the number of data records on a section-by-section basis. The

OUTFIL

count will be the number of data records *before* any multi-line OUTREC processing has been done. This number will be a right-justified, 8-digit field with leading zeros suppressed. The maximum value is 99999999.

SUBCOUNT15

This subparameter is identical to the SUBCOUNT subparameter except for the allowable size of the count number. For SUBCOUNT15 the number will be a right-justified *15-digit field* with leading zeros suppressed. The maximum value is 999999999999999.

Rules for Specifying TRAILER Subparameters

Observe the following guidelines when you specify TRAILER subparameters:

- Separate fields with commas, except for /, where commas are optional.
- Enclose literals in single quotes.
- Specify blank fields of *n* bytes as nX.
- If a SyncSort editing mask is used for totaled or subtotaled data (either by specification or by default), the length of the generated pattern will be determined by the maximum permissible length supported for that data format, regardless of the actual length of the field being totaled or subtotaled. Use the LENGTH subparameter to override the length of the pattern.
- Trailers specified with fewer blanks than the logical record length (LRECL) of the output record are automatically padded on the right with blanks.
- If a trailer exceeds the logical record length (LRECL) of the output record, use the OUTREC control statement or the OUTREC parameter to expand the output record length so that it is at least as long as the longest header. For example, if the longest trailer is 115 characters and the output record length is 80 bytes, use the OUTREC control statement or the OUTREC parameter to insert a blank in position 115 of the output record. This will cause bytes 81 through 115 to be padded with blanks.

SECTIONS Parameter (Optional)

The SECTIONS parameter allows the output report to be divided into sections.

The format of the SECTIONS parameter is illustrated below.

SECTIONS=(field₁[,field₂]...)

Each field is specified as follows:

p,l [,subparameter1] [,subparameter2] ...

Figure 35. SECTIONS Parameter Format

The SECTIONS parameter identifies the control field(s) that determine or control section breaks. More than one control field can be specified to subdivide a report within sections. However, if more than one control field is specified, the specifications must be made in major to minor order. A major control field break causes all minor control fields to break at the same time.

Each control field is identified by its position *p* and length *l*.

p The position value indicates the first byte of the field relative to the beginning of the record *after* processing by an E15/E32 exit, the INREC control statement, the OUTREC control statement, and an E35 exit, if specified, but *before* processing by the OUTREC parameter and other report writing parameters of the OUTFIL control statement, if specified.

l The length value indicates the length of the field. The length must be an integer number of bytes and cannot exceed 256 bytes.

For each control field, one or more of the following subparameters may be specified: SKIP, HEADER3, or TRAILER3. The SECTIONS subparameters are described below.

$\left[,\text{SKIP} = \left\{ \begin{matrix} \text{P} \\ \text{nL} \end{matrix} \right\} \right] [, \text{TRAILER3}=(...)] [, \text{HEADER3}=(...)] [, \text{PAGEHEAD}]$

Figure 36. Sections Subparameter

SKIP The SKIP subparameter specifies the amount of spacing that should occur after a section is completed. This spacing will follow immediately after the last TRAILER3 for that section, if specified. SKIP=nL specifies that the next line of the report will appear after *n* number of blank lines, with *n* being between 0 and 255. SKIP=P specifies a page break following the completion of a section.

HEADER3 The HEADER3 subparameter specifies a section header or title that will appear at the start of each new section. The HEADER3 format is identical to the format of the HEADER1/HEADER2 parameters. (See *HEADER1/HEADER2 Parameters* for details.)

OUTFIL

TRAILER3 The TRAILER3 subparameter specifies a section trailer that will appear at the end of each section. The TRAILER3 format is identical to the format of the TRAILER1/TRAILER2 parameters. (See *TRAILER1/ TRAILER2 Parameters* for details.)

PAGEHEAD The PAGEHEAD subparameter may be specified in conjunction with the HEADER3 subparameter. The PAGEHEAD subparameter specifies that the HEADER3 appear at the top of each page following any HEADER2, as well as at the start of each new section. PAGEHEAD is ignored if no HEADER3 is specified.

A control field may be specified without any subparameters. This allows multiple non-contiguous control fields to be specified for each SECTIONS break field.

LINES Parameter (Optional)

Use the LINES parameter to define the logical pages constituting a report. The pages can be defined in three ways:

- Using the carriage control characters automatically supplied by SyncSort for z/OS
- Using ANSI control characters supplied by the user
- Using a combination of the above two methods.

Regardless of which method is selected, the number of lines defining a logical page must be equal to or greater than the total number of lines, including blank lines, required for all HEADER2, HEADER3, TRAILER2, and TRAILER3 entries *plus* at least one record. If multi-line OUTREC is used, all lines produced from each input record will be written to the same logical page.

The format of the LINES parameter is illustrated below:

$$\text{LINES} = \left\{ \begin{array}{l} n \\ \text{ANSI} \\ (\text{ANSI},n) \end{array} \right\}$$

Figure 37. LINES Parameter Format

LINES=n

If LINES=n is specified, paging is automatic and carriage control characters are added to the beginning of each record by SyncSort. Because SyncSort requires one byte for a control character, the LRECL specified in the SORTOUT, SORTOFx, or SORTOFxx DD statement must be one byte longer than the number of bytes specified for the output record length.

Specify n as a value from 1 to 255. If report writing parameters are specified for the file(s) (e.g., HEADERS, TRAILERS, SECTIONS), the default is LINES=60.

The LINES= n specification works in conjunction with any HEADERS and TRAILERS you have specified as follows:

- HEADER1, if specified, prints as a preface to the report. Its page is not numbered.
- An automatic page break occurs after HEADER1. Every n th line after the completion of HEADER1 will signal the start of a new page.
- A HEADER2 entry, if present, is the first line(s) on each page, followed by any HEADER3 entries that might be triggered either by control breaks or by PAGEHEAD specifications in the SECTIONS parameter. HEADER2 is part of the logical page.
- A HEADER3 entry, if present, is part of a section of the report. It prints as a header for the separate report sections. HEADER3s appear in major to minor order according to the order of their associated sections.
- If PAGEHEAD is specified, HEADER3 prints immediately below HEADER2, if specified, or at the top of the page if a HEADER2 is not specified. A HEADER3 will not print near the end of a page if there is not sufficient room on that page for at least one data record and a TRAILER2, if specified.
- A TRAILER3 entry, if present, is part of a section of the report. It prints as a conclusion or summary for the separate report sections. TRAILER3s will appear in major to minor order according to the order of their associated sections.
- A TRAILER2 entry, if present, will be the last line(s) on the logical page, preceded by any TRAILER3s triggered by coincidentally occurring control breaks. TRAILER2 is part of the logical page.
- TRAILER1 will be the last page of the entire report. Its page is not numbered.

Therefore, when LINES= n is specified, all HEADER2, HEADER3, TRAILER2, and TRAILER3 entries will be included as part of n (the total number of lines in a logical page) and will print as described above.

LINES=ANSI

If LINES=ANSI is specified, user-provided ANSI control characters define the logical pages. The first byte of each output record must contain an ANSI control character (inserted, for example, by an E35 program) which is valid for the specified output device type. For example, inserting a '0' in byte 1 of the output records produces double-spaced records.

The ANSI control characters which can be used with the LINES=ANSI specification are summarized in the *ANSI Control Character Chart* below.

OUTFIL

If printed output is requested, the ANSI control characters do not print as part of the output record. If, however, the report is routed to a disk or tape device, the control characters are included in the output data.

The `LINES=ANSI` specification works in conjunction with any `HEADERs` or `TRAILERs` you have specified. If you specify `HEADER2`, the ANSI specification affects this header as follows:

- After `HEADER1` is output, the first logical page begins with the first line of `HEADER2`.
- A logical page ends when *data* with a '1' in the first byte are encountered. The printing of a data record beginning with a '1' is delayed until after `TRAILER2` and `HEADER2`, if specified, are output. When record printing resumes, this delayed record will be modified to have a control character '+', which causes it to print over the last line of `HEADER2` (or `HEADER3`, if `HEADER3` appears at the top of the page). To prevent the data record from printing over a text line of a header, the header should end with at least one blank line, specified by a slash (/).
- To print `HEADER2` at the top of a new physical page, the `HEADER2's` first line should begin with a '1'.
- Because you are in complete control of the paging with `LINES=ANSI`, you can permit `HEADER2` to appear between variable numbers of printed records.

`LINES=(ANSI,n)`

If `LINES=(ANSI,n)` is specified, ANSI control characters govern vertical control, and the 'n' specification provides additional automatic paging. Added flexibility is provided because the user can elect to double or triple space the output and still use automatic paging.

When SyncSort encounters a data record with a '1' in the first byte, SyncSort begins a new logical page. If no data record begins with a '1' but the next data record would cause the number of lines on the page to exceed *n*, SyncSort treats the record as if it began with a '1' and begins a new page.

Refer to the `LINES=ANSI` discussion for information on using a `HEADER2` with ANSI control characters.

Multiline `OUTREC` may not be used with `LINES=ANSI` or `LINES=(ANSI,n)`.

Valid ANSI Control Characters

The following chart lists the ANSI control characters accepted by SyncSort.

Code	Interpretation	Code	Interpretation
blank	Space one line before printing	6	Skip to channel 6 before printing
0	Space two lines before printing	7	Skip to channel 7 before printing
-	Space three lines before printing	8	Skip to channel 8 before printing
+	Suppress space before printing	9	Skip to channel 9 before printing
1	Skip to channel 1 before printing	A	Skip to channel 10 before printing
2	Skip to channel 2 before printing	B	Skip to channel 11 before printing
3	Skip to channel 3 before printing	C	Skip to channel 12 before printing
4	Skip to channel 4 before printing	V	Select stacker 1
5	Skip to channel 5 before printing	W	Select stacker 2

Table 14. ANSI Control Character Chart

NODETAIL Parameter (Optional)

The NODETAIL parameter instructs the SortWriter facility to generate an output report consisting *only* of header and trailer entries. Data records are *not* included in the output report when this parameter is specified.

Thus, for example, it is possible to generate a report with section trailers containing totals and record counts without printing *any* data records.

REMOVECC Parameter (Optional)

The REMOVECC parameter generates reports that do not include ANSI carriage control characters that specify printer actions (for example, skipping a line or ejecting a page). The REMOVECC parameter omits the carriage control character from all of the report records. REMOVECC simplifies the removal of printer controls when output is to be displayed online or written to a list data set rather than a printout. When REMOVECC is used, the LRECL does not require an extra byte for the carriage control character, and the RECFM does not require the 'A' (for ANSI); thus you would specify FB, not FBA.

OUTFIL

NULLOFL Parameter (Optional)

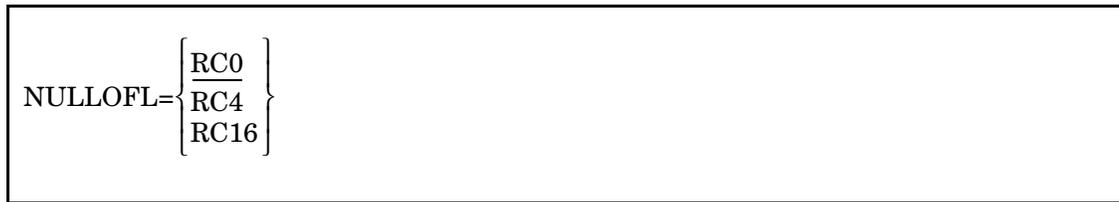


Figure 38. NULLOFL Parameter Format

The NULLOFL parameter specifies the action to be taken when any non-SORTOUT OUTFIL data set contains no data records.

- RC0** The delivered default instructs SyncSort to issue a return code of 0 if not overridden by a higher return code set for another reason.
- RC4** Instructs SyncSort to issue a WER461I warning message and continue processing. A return code of 4 will be issued if not overridden by a higher return code set for another reason.
- RC16** Instructs SyncSort to issue a WER461A message and to terminate processing with a return code of 16.

Sample OUTFIL Control Statements

Example 1

The following example illustrates how to use the OUTFIL control statement to define multiple output files.

```
OUTFIL FILES=1,OUTREC=(10:1,20,40:45,5,50:60,8),  
          INCLUDE=(21,2,CH,EQ,C'NY')  
OUTFIL FILES=2,OUTREC=(20:1,20,50:60,8),  
          INCLUDE=(21,2,CH,EQ,C'MA')
```

Figure 39. Sample OUTFIL Control Statement

The two OUTFIL control statements illustrated above are required to create two different output files.

- The output records in the first file (SORTOF1) contain three fields from the input record. The first input record field begins in byte 1 and is 20 bytes long, the second input record field begins in byte 45 and is 5 bytes long, and the third input record field begins in byte 60 and is 8 bytes long. This file will include only those records with 'NY' in bytes 21 and 22 of the input record. These three fields will begin in bytes 10, 40, and 50 of the output record.

- The output records in the second file (SORTOF2) contain two fields from the input record. The first input record field begins in byte 1 and is 20 bytes long, and the second input field begins in byte 60 and is 8 bytes long. This file will include only those records with ‘MA’ in bytes 21 and 22 of the input record. These two fields will begin in bytes 20 and 50 of the output record.

Example 2

```
OUTFIL FILES=(01,02,03),OUTREC=(1:1,40,50:41,40)
```

Figure 40. Sample OUTFIL Control Statement

This OUTFIL control statement creates three identically formatted output files: SORTOF01, SORTOF02, and SORTOF03. These files may be written to the same output device or to three different output devices.

- The output records contain two input record fields. The first input record field begins in column 1. This field began in position 1 before OUTREC processing and is 40 bytes long. The second input record field begins in column 50. This field began in position 41 before OUTREC processing and is 40 bytes long. The two fields will begin in positions 1 and 50 after OUTREC has been processed.

Example 3

```
OUTFIL FTOV,VLTRIM=C' *',OUTREC=(1,7,9:8,8)
```

Figure 41. Sample OUTFIL Control Statement with FTOV and VLTRIM

This OUTFIL control statement uses FTOV to convert fixed-length records to variable-length records and VLTRIM to remove the specified type of trailing bytes (in this case, asterisks).

The control statement would produce the following output:

Input Records	Output Records	Record Length (with 4-byte RDW)
RECORD1ABC*****	RECORD1 ABC	15
RECORD2ABCDEF**	RECORD2 ABCDEF	18
RECORD3ABC*****Z	RECORD3 ABC*****Z	20

Comprehensive examples illustrating the SortWriter facility and the multiple output capability of the OUTFIL control statement are provided in “Chapter 3. How to Use SyncSort’s Data Utility Features”.

OUTREC

OUTREC Control Statement

The OUTREC control statement reformats the output records. Use the OUTREC control statement to accomplish the following tasks:

- Delete or repeat segments of the input records.
- Insert character strings between data fields.
- Insert binary zeros.
- Create a sequence number field.
- Convert numeric data to printable format or to another numeric data format.
- Perform arithmetic operations (multiply, divide, add, subtract) and minimum and maximum functions with numeric fields and constants. This “horizontal arithmetic” ability complements the “vertical arithmetic” already available with SUM and OUTFIL TOTAL, MIN, MAX, and AVG.
- Convert data to printable hexadecimal format.
- Translate the case of EBCDIC letters from uppercase to lowercase or lowercase to uppercase, or translate a field based on an ALTSEQ table in effect.
- Select, realign, and reorder data fields.
- Convert a variable-length record input file to a fixed-length record output file.

The OUTREC parameter of the OUTFIL control statement can also be used to accomplish any of the above tasks. The INREC control statement can also be used to accomplish any of the above tasks except for converting a variable-length record file to a fixed-length record file.

Consider these guidelines when deciding whether to use the INREC control statement, the OUTREC control statement, or the OUTREC parameter of the OUTFIL control statement:

- Use the INREC control statement to delete irrelevant data fields, reformat numeric fields to a shorter length, or combine numeric fields with arithmetic operations and functions. Reducing the size of the input records before they are sorted or merged usually improves performance.
- Use either the OUTREC control statement or the OUTREC parameter of the OUTFIL control statement to expand the data record, create new numeric fields, realign data fields, convert and edit numeric data, and change from variable-length format to fixed-length format when you are creating *one* output file.

- Use the OUTREC control statement when you are creating *multiple* output files with the *same* output record formatting.
- Use the OUTREC parameter of the OUTFIL control statement when you are creating *multiple* output files with *different* output record formatting.
- Use the OUTREC control statement if you need to convert a numeric field to printable format so it can be displayed in an OUTFIL header.
- Use the OUTREC parameter of the OUTFIL control statement when an E35 exit must process the records first.
- Use the OUTREC parameter of the OUTFIL control statement when you specify the TOTAL and/or SUBTOTAL subparameters of the TRAILER parameter so that the accumulator(s) can sum numeric fields *before* they have been converted to readable format and edited.

OUTREC Control Statement Format

The format for the OUTREC control statement is illustrated below.

OUTREC FIELDS = (field₁ [field₂(| ...) [,CONVERT]])

Fields can be specified as follows:

[c:]	}	<p>[p,l [,subparameters]</p> <p>[n] X</p> <p>[n] X'hhhh...hh'</p> <p>[n] C'literal string'</p> <p>[n] Z</p> <p>'date field'</p> <p>'time field'</p> <p>SEQNUM,i,f [,START = { $\frac{1}{n}$ }] [,INCR = { $\frac{1}{i}$ }]</p>
------	---	--

Figure 42. OUTREC Control Statement Format

Note: The *n/* entry and the VLFILL parameter cannot be used with the INREC or OUTREC control statement. They can only be used with the OUTREC parameter of the OUTFIL control statement. For a description of the *n/* entry and the VLFILL parameter, see page 2.65.

OUTREC

FIELDS Parameter (Required)

The FIELDS parameter specifies fields to be included in the output record. Fields can be data fields, spaces, hexadecimal digits, literal strings, binary zeros, current date and time literals, or sequence numbers.

Each *data field* specified in the FIELDS parameter is identified by its position *p* and length *l*.

- | | |
|-----------|--|
| c: | The column value (optional) specifies the output column in which a field should begin. |
| p | For INREC, the position value indicates the first byte of the field relative to the beginning of the input record after E15 processing, if specified, has completed. For OUTREC, the position value indicates the first byte of the field after <i>both</i> E15 <i>and</i> INREC processing, if specified, have completed. If the OUTREC parameter of the OUTFIL control statement is used, the position value refers to the record after E35 processing as well. The field must begin on a byte boundary. |
| l | The length value indicates the length of the field. The length must be an integer number of bytes. |

Specifying the FIELDS Parameter for Variable-Length Records

If you are *not* using the CONVERT option to convert variable-length records to fixed-length records, you must observe these rules when you specify the FIELDS parameter for variable-length records:

- Remember to specify 4 bytes for the Record Descriptor Word in the first output field. You can include the 4 bytes in the length value of the first field if the first field in the original data record is also the first field specified in the FIELDS parameter.
- To include any portion of the variable part of the input records, specify a position value *without* a length value as the last entry. The only subparameters you can specify after the position value are the HEX and TRAN conversion subparameters. (Refer to the FIELDS subparameters for an explanation of HEX and TRAN conversion.)
- If INREC or OUTREC processing changes the output record length, the contents of the Record Descriptor Word will be automatically revised by the sort.

Field (p,l) Subparameters

Use the FIELDS subparameters to accomplish these tasks:

- Specify the column in which a field should begin.

- Specify halfword, fullword, or doubleword alignment.
- Convert a numeric field to a printable format with editing capabilities.
- Convert numeric data to another numeric data format.
- Perform minimum and maximum functions and arithmetic operations (multiply, divide, add, subtract) with numeric fields and constants.
- Convert a field to its printable hexadecimal representation.
- Translate the case of EBCDIC letters from uppercase to lowercase or lowercase to uppercase, or translate a field based on an ALTSEQ table in effect.

The figure below illustrates how the FIELDS subparameters should be specified and describes their functions. For information on the EDIT, LENGTH, Mm, and SIGNS subparameters, see “How to Convert Numeric Data” on page 2.97.

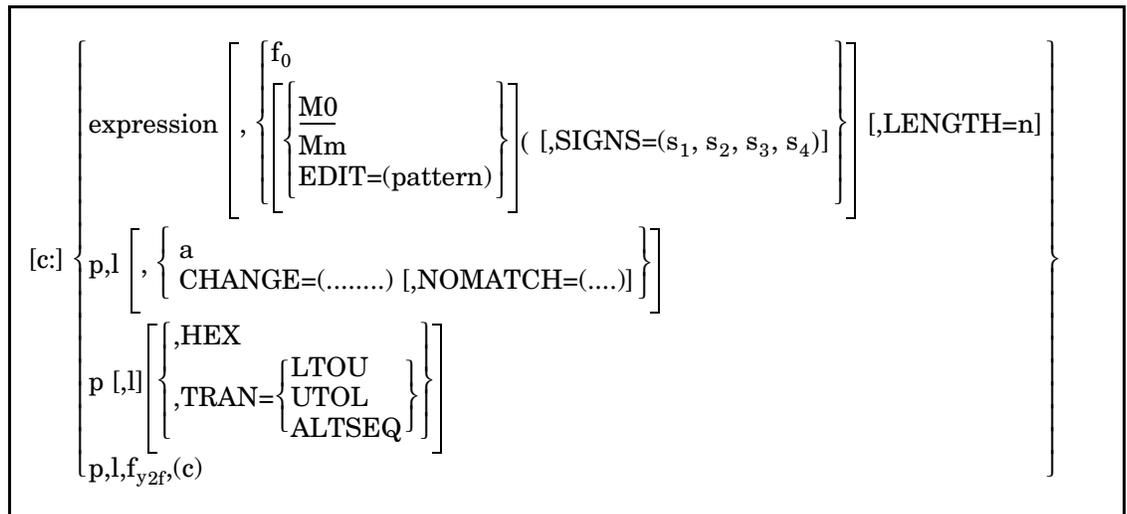


Figure 43. Fields Subparameters Format

The following describes the *c*: subparameter:

- c:** Use the *c*: subparameter to define the column in which the field should begin. SyncSort will add the appropriate number of blanks to achieve the proper alignment. This subparameter can be specified for *all* types of fields.

The term *expression* represents the following syntax:

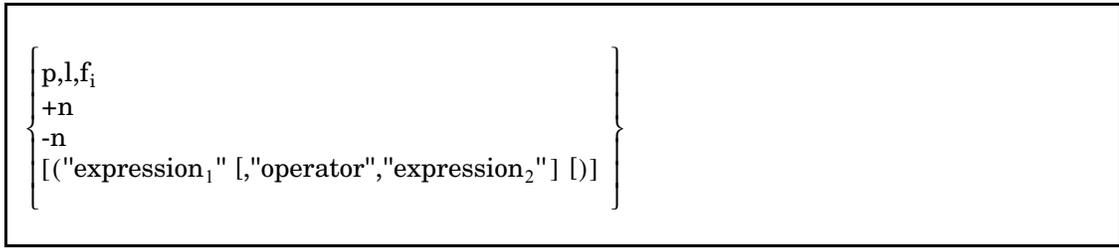


Figure 44. Syntax for expression

The following describes the elements of *expression*:

- p,l,f_i** This specifies the position, length, and format of an input field. (See the description of *f_i* below for details.)

- +n** This represents a positive numerical constant of up to 15 decimal digits. The + sign must be specified.

- n** This represents a negative numerical constant of up to 15 decimal digits. The - sign must be specified.

- expression** An expression defines a numeric value. The simplest forms of an expression consist of a numeric data input field defined either by *p,l,f_i* or a constant defined by *+n* or *-n*. Expressions can also be created by connecting these simple expressions with operators, as shown in the last line of the above syntax illustration. Parentheses may be used to change the default precedence order of the operators. Algebraic equations can thus be represented with an expression.

A maximum value of 15 digits is permitted at all times in evaluating an expression. If this is exceeded, a critical error will be issued. Similarly, an attempted division by zero will also result in a critical error. The results of division will be rounded down to an integer.

Once an expression has been defined, its value can either be converted to a numeric output data format or to a printable numeric format using editing masks. See “How to Convert Numeric Data” on page 2.97. The default is to use the M0 editing mask to create printable output. The number of digits in an expression is defined to be 15 (unless the expression is a simple *p,l,f_i* field), so using the M0 default mask will create a 16-byte output field.

The following are expressions:

```
+10
10, 2, Y2Z
```

```
+10,ADD,10,2,Y2Z
1,4,ZD
10,2,PD
+30
1,4,ZD,ADD,10,2,PD
+30,MUL,(1,4,ZD,ADD,10,2,PD)
+30,MUL,(1,4,ZD,ADD,10,2,PD),MIN,(5,5,ZD,DIV,+100)
(+30,MUL,(1,4,ZD,ADD,10,2,PD)),MIN,(5,5,ZD,DIV,+100)
```

operator

Operations between two numeric fields or constants are performed with *operators*. There are two types of operators: *function operators* and *arithmetic operators*. The following are the function operators:

MIN Generates the minimum arithmetic value of two specified fields.

MAX Generates the maximum arithmetic value of two specified fields.

The following are the arithmetic operators:

MUL multiplication

DIV division

ADD addition

SUB subtraction

The following rules of arithmetic precedence apply in computing an “expression”:

- Conditions within parentheses are evaluated first, from innermost to outermost parentheses.
- The arithmetic functions of minimum and maximum (MIN and MAX) are performed before the arithmetic operators (MUL, DIV, ADD, SUB). Within the arithmetic operators, multiplication (MUL) and division (DIV) are performed before addition (ADD) and subtraction (SUB). Operations within the same precedence level are performed from left to right.

f_i

Use this parameter together with *p,l* to define the *input* format of a numeric field that is part or all of an expression. The expression will then be converted to either another numeric data format or to a printable format. In such cases, indicate the format of the data field that is to be converted by replacing *f_i* with BI, FI, PD, ZD, CSF/FS,

OUTREC

PD0, or one of the year data formats (Y2B, Y2C, Y2D, Y2P, Y2S, Y2Z, Y2T, Y2U, Y2V, Y2W, Y2X, Y2Y).

Also use this parameter when a 2-digit packed decimal year value is to be expanded to a 4-digit packed decimal value. In such cases replace f_i with Y2ID or Y2IP. The Y2ID and Y2IP formats cannot be used to form complex arithmetic expressions and do not allow the specification of mask (Mm), EDIT, SIGNS, or LENGTH.

An l value indicating the length of the field must be specified in accordance with the following allowable values:

- for BI ... 1-4 inclusive
- for CSF or FS ... 1-16 inclusive (15 digit limit)
- for FI ... 1-4 inclusive
- for PD ... 1-8 inclusive
- for PD0 ... 2-8 inclusive
- for Y2B ... 1
- for Y2C ... 2
- for Y2D ... 1
- for Y2ID ... 1
- for Y2IP ... 2
- for Y2P ... 2
- for Y2S ... 2
- for Y2Z ... 2
- for ZD ... 1-15 inclusive
- for Y2T ... 3-6 inclusive
- for Y2U ... 2-3 inclusive
- for Y2V ... 3-4 inclusive
- for Y2W ... 3-6 inclusive
- for Y2X ... 2-3 inclusive
- for Y2Y ... 3-4 inclusive

Field conversion of a single p,l,f_i expression with a format of Y2B, Y2C, Y2D, Y2P, Y2S, Y2Z, Y2ID, or Y2IP does **not** default to the use of the M0 default output mask. The default will convert to a 4-digit 4-byte printable year. However, except for Y2S, Y2ID, and Y2IP, these formats can be used to form expressions with operators. In this case, the default will use the M0 output mask with 15 decimal digits. The specification of an output numeric data format f_o or mask Mm, EDIT, SIGNS, or LENGTH is permitted except when using Y2S, Y2ID, and Y2IP.

Field conversion of a single p,l,f_i expression with a format of Y2T, Y2U, Y2V, Y2W, Y2X, or Y2Y does **not** default to the M0 mask. These fields are converted to a printable year with the 2-digit year portion converted to a 4-digit value. The year portion of the date is

converted to a 4-digit year using the century window defined by the CENTWIN parameter. The century window is not used for the special values, which are only expanded with characters of the proper format. The specification of an output numeric data format f_o or mask Mm , EDIT, SIGNS, or LENGTH is not permitted.

The following describes the other FIELDS subparameters:

- f_o** Use this subparameter to define the *output* numeric data format of an expression. When f_o is specified, mask Mm , EDIT, and SIGNS cannot be specified. Indicate the desired format of the output field by replacing f_o with BI, CSF/FS, FI, PD, or ZD. See “How to Convert Numeric Data” on page 2.97 for the default lengths of these fields. See “The LENGTH= n Subparameter” on page 2.108 for how this default may be changed.
- Mm** Use the Mm subparameter to indicate that one of the 27 SyncSort-provided editing masks, M0-M26, is to be used. Replace 'm' with the mask number. For details, see “The Mm Subparameter (Editing Masks)” on page 2.109.
- EDIT=(pattern)** Use the EDIT subparameter to specify that a user-provided editing mask should be used to format the output fields. For details, see “The EDIT Subparameter” on page 2.107.
- SIGNS=(s_1,s_2,s_3,s_4)** Use the SIGNS subparameter to specify the signs that will appear before or after the edited number. For details, see “The SIGNS Subparameter” on page 2.112.
- LENGTH= n** Use the LENGTH subparameter to alter the length of the output field. This is normally determined by the number of numeric digits d and either the data format or the edit pattern and format of the edited field. For details, see “The LENGTH= n Subparameter” on page 2.108.
- a** Use this subparameter to tell SyncSort how the field should be aligned with respect to the start of the output record. Replace a with H, F, or D to specify halfword (H), fullword (F), or doubleword (D) alignment. The alignment itself actually takes place after the column designation. It will automatically pad any provided field with the number of bytes of binary zeros required to achieve the specified alignment. This subparameter cannot be used in conjunction with data conversion.
- change-parm** This variable represents the CHANGE subparameter. The CHANGE subparameter changes an input field to a *replacement* constant in the reformatted output record if the input field equals a

OUTREC

search constant. For a complete description, see “The CHANGE Subparameter” on page 2.113.

HEX

Use the HEX subparameter to convert a record field to its hexadecimal representation. Specify this subparameter immediately after the position *p* and the length *l* of the field to be converted. Specify *p,l,HEX* for both fixed-length records and the fixed-length portion of variable-length records. Specify *p,HEX* for the variable-length portion of variable-length records. Starting in position *p* of the input record, for a length of *l*, each byte will be converted to its hexadecimal representation. Note that in the reformatted record, the converted field will be twice the length of the original field.

TRAN

Use this subparameter to change the case of EBCDIC letters from lowercase to uppercase, uppercase to lowercase, or based on an alternate collating sequence (ALTSEQ) table in effect. Specify this subparameter immediately after the position *p* and the length *l* of the field to be converted. Specify *p,l,TRAN* for both fixed-length records and the fixed-length portion of variable-length records. Specify *p,TRAN* for the variable-length portion of variable-length records. Starting in position *p* of the input record, for a length of *l*, each byte will be converted as per specification.

$$\text{TRAN} = \left\{ \begin{array}{l} \text{LTOU} \\ \text{UTOL} \\ \text{ALTSEQ} \end{array} \right\}$$

LTOU Instructs SyncSort to translate EBCDIC letters in a specified field from lowercase to uppercase.

UTOL Instructs SyncSort to translate EBCDIC letters in a specified field from uppercase to lowercase.

ALTSEQ Instructs SyncSort to translate characters based on the ALTSEQ table in effect.

For examples of OUTREC control statements that use the TRAN subparameter, see Figure 55 on page 2.120 and Figure 56 on page 2.121.

f_{y2p}(c)

Use this subparameter together with the *p,l* elements to indicate the conversion of a full-date field to a printable date with separator character(s). The “c” represents the separator and can be any character except a blank. The year portion of the date is converted to a 4-digit year using the century window defined by the CENTWIN parameter. The century window is not used for the special values, which are expanded with characters of the proper format.

The following table shows what is produced if (c) is set to a “?”:

Full-Date Format	Date Form	Length (bytes)	Output Format
Y2T	yyx	3	yyyy/x
	yyxx	4	yyyy/xx
	yyxxx	5	yyyy/xxx
	yyxxxx	6	yyyy/xx/xx
Y2U	yyx (X'yyxs')	2	yy/x
	yyxxx (X'yyxxxs')	3	yy/xxx
Y2V	yyxx (X'0yyxxs')	3	yy/xx
	yyxxxx (X'0yyxxxxs')	4	yy/xx/xx
Y2W	xyy	3	x/yy
	xxyy	4	xx/yy
	xxxyy	5	xxx/yy
	xxxxyy	6	xx/xx/yy
Y2X	xyy (X'xyys')	2	x/yy
	xxxyy (X'xxxxyys')	3	xxx/yy
Y2Y	xxyy (X'0xxxyys')	3	xx/yy
	xxxxyy (X'0xxxxxyys')	4	xx/xx/yy

Table 15. Full-Date Field Conversions

How to Convert Numeric Data

One of the most important functions of OUTREC processing is to convert a numeric data field or an expression to either an output numeric data format or a printable format with editing capabilities.

OUTREC

OUTREC processing can also expand a packed decimal 2-digit year to a packed decimal 4-digit year. In such cases, Y2ID or Y2IP formats are used to convert from a 2-digit to a 4-digit year while maintaining a packed format. For details on converting year data, see “Converting Year Data with Century Window Processing on INREC, OUTREC, or OUTFIL OUTREC” on page 2.100.

When a single numeric field defined by p,l,f_i is to be converted to a printable format without editing, the format and length of the field determine the length of the output field, as illustrated in the following two tables.

Data Conversion		
Input Format	Number of Bytes in Input Field	Number of Resulting Digits (d)
ZD	n	n
PD	n	2n-1
BI, FI	1	3
BI, FI	2	5
BI, FI	3	8
BI, FI	4	10
CSF or FS	n	n (to maximum of 15, then truncated)
PD0	n	2n-2 digits
Y2C, Y2P, Y2S, Y2Z	2	4 digits
Y2B, Y2D	1	4 digits
Y2ID	1	2 bytes
Y2IP	2	3 bytes

Table 16. Data Conversion Table

For full-date formats, the number of bytes in the input field can vary. The following table shows input lengths for full-date formats and the resulting output length:

Input Format	Number of Bytes in Input Field	Number of Resulting Digits (d)
Y2T	3	5
	4	6
	5	7
	6	8
Y2U	2	5
	3	7
Y2V	3	6
	4	8
Y2W	3	5
	4	6
	5	7
	6	8
Y2X	2	5
	3	7
Y2Y	3	6
	4	8

Table 17. Data Conversion Table – Full-Date Formats

For any other type of expression (those that are not a simple p,l,f_i), SyncSort internally treats the input field length as 15. Thus, in Table 15 and Table 17, d would be equal to 15. (For more details on expressions, see the description of *expression* on pages 2.87-2.89.)

If you specify no other FIELDS subparameters, the result will be converted to printable output according to the default editing mask, M0. See “The Mm Subparameter (Editing Masks)” on page 2.109. Other forms of printable output can be created by using the EDIT, LENGTH, Mm, and SIGNS subparameters, which allow you to create your own edit patterns, or by using one of the 27 SyncSort-supplied editing masks, which are appropriate for many editing operations.

To convert to a numeric data field, simply specify an output format of BI, CSF/FS, FI, PD, or ZD. The default output field length is determined by the following table, where d in the

OUTREC

second column is obtained from column 3 of Table 16 on page 2.98 for p,l,f_i fields. For any other type of expression (not p,l,f_i), d is equal to 15.

Output Format	Default Output Length (bytes)
BI	4
CSF	$d + 1$
FI	4
FS	$d + 1$
PD	$d/2 + 1$
ZD	d

Table 18. Output Length of Output Formats

These lengths can be overridden by specifying the LENGTH parameter.

The following five sections describe the data conversion capabilities:

- Converting Year Data with century window processing on INREC, OUTREC, or UTFIL OUTREC
- The EDIT Subparameter
- The LENGTH= n Subparameter
- The Mm Subparameter (Editing Masks)
- The SIGNS Subparameter

Converting Year Data with Century Window Processing on INREC, OUTREC, or UTFIL OUTREC

A 2-digit year field, as specified by the Y2B, Y2C, Y2D, Y2P, Y2S, Y2Z, Y2ID, and Y2IP formats, can be converted on output to a 4-digit year. For full-date fields, as specified by the Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y formats, the 2-digit year portion can be expanded on output to a 4-digit year.

The following describes output data conversion for date fields:

- The Y2B format specifies 2-digit, 1-byte binary year data that will be converted to a 4-digit, displayable character format with the appropriate century value. For information on the range of binary values representing year data with Y2B, see Table 24 on page 2.135.
- The Y2C and Y2Z formats specify 2-digit year data that are in displayable (zoned decimal) format. The 2-digit year data will be expanded to a 4-digit field containing the appropriate century value.
- The Y2S format is equivalent to Y2C and Y2Z for valid numeric year data. All three formats will convert such data to a displayable 4-digit year with the appropriate century value. Y2S, however, provides additional functionality. For data with binary zeros (X'00'), a blank (X'40') or binary ones (X'FF') in the first byte, typically to identify header/trailer records, Y2S will expand the data to 4 bytes, padded in the first 2 bytes with the same character as found in the first byte of the input field. The fourth byte of the output field is copied unchanged from the second byte of the input field.

The following symbolic representation shows the treatment in hexadecimal of the three types of data:

SORTIN Input	OUTREC Output
00ab	000000ab
40ab	404040ab
FFab	FFFFFFab

- The Y2D and Y2P formats specify 2-digit year values in packed decimal format. The processing applied to these fields will create a 4-digit year value converted to a displayable character format.
- The Y2ID and Y2IP formats take as input the same 2-digit packed decimal year data as the Y2D and Y2P formats but produce a 4-digit year output that remains in packed decimal format. Y2ID will convert data from X'yy' to X'ccyy', and Y2IP will convert data from X'ayys' to X'accyys', where cc is the correct century. (For a description of Y2D and Y2P formats for SORT or MERGE processing, see Table 24 on page 2.135 or Table 10 on page 2.42, respectively.)
- For full-date fields (Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y), the 2-digit portion will expand to the appropriate 4-digit year based on the CENTWIN setting. The output field length can be determined from Table 17 on page 2.99.

Note that an additional data format, PD0, which is typically used to process the month and day portion of packed decimal data, is not affected by CENTWIN processing and will not convert 2-digit year data to 4-digit years. PD0 can be used with the SyncSort-supplied edit mask M11. The year data formats Y2B, Y2C, Y2D, Y2P and Y2Z can also be used when forming expressions. The 4-digit year will be converted to an integer for arithmetic calculations. Any expression with these formats can also be converted to an output numerical data format f_{\circ} , or to printable output by specifying one or more of the OUTREC FIELDS subparameters (Mm, EDIT, SIGNS or LENGTH). For information on using the year data formats

OUTREC

for SORT or MERGE field specifications, see “CENTWIN Parameter (Optional)” on page 2.134 or “CENTWIN Parameter (Optional)” on page 2.41, respectively. For more information on using the year data formats for INREC or OUTREC processing, see “Example 5” on page 2.120.

For more information on converting full-date formats, see the descriptions of the f_i and $f_{y2B(c)}$ parameters on pages 2.93-2.96, Table 15 on page 2.97, and Table 17 on page 2.99.

Converting SMF Date and Time Formats

You can convert SMF date and time formats to standard date and time formats. The following table shows the SMF formats and the converted output:

SMF Format	Converted Output
DT1	Z'yyyymmdd'
DT2	Z'yyyymm'
DT3	Z'yyyddd'
TM1	Z'hhmmss'
TM2	Z'hhmm'
TM3	Z'hh'
TM4	Z'hhmmssxx'

Table 19. SMF Formats and Converted Output

For DTn, the source is the 4-byte packed SMF date value (P'cyddd'). For TMn, the source is a 4-byte binary SMF time value.

The c in the date source P'cyddd' represents the century. It is converted as follows: 0 is converted to 19, 1 is converted to 20, and 2 or greater is converted to 21.

The converted output is a zoned decimal field, where each character in the table represents a single byte. For TM4, xx represents hundredths of a second.

The SyncSort pre-defined edit masks (M0-M26) or specified edit patterns can be used to edit the converted date and time. The default mask is M11.

Notes: A data exception (0C7 ABEND) or an inaccurate ZD date can occur if an SMF date is not valid. An inaccurate ZD time can occur if an SMF time is not valid. SMF dates and times are processed as positive values.

For an example of an OUTFIL control statement that converts SMF formats, see Figure 53 on page 2.120.

Specifying Literal Fields

Spaces (X), hexadecimal digits (X'hhhh...hh'), literal strings (C'literal string'), and binary zeros (Z) can also be specified in the FIELDS parameter. Each of these entries can be preceded by an 'n' value which indicates that a specified number of spaces, hex digits, literal strings, or binary zeros should be inserted in the output record. Additionally, SEQNUM can be specified to place a sequence number field in the output record.

nX Use the nX entry to specify a number *n* of spaces. The *n* value may be any number from 1 to 4095 inclusive. The X entry represents a space and must be coded to the immediate right of the number specified for *n*. If more than 4095 spaces are desired, two or more nX values should be specified.

nX'hhhh...hh' Use the nX'hhhh...hh' entry to specify that *n* copies of hex digits or hex digit strings should be inserted in the output record. (Each *hh* pair is 1 byte of output.) The repetition factor *n* may be any number from 1 to 4095 inclusive.

nC'literal string' Use the nC'literal string' entry to specify that *n* copies of literal strings should be inserted in the output record. The repetition factor *n* may be any number between 1 and 4095 inclusive. An apostrophe within a literal string must be specified with a double apostrophe (e.g., C'O"LEARY').

nZ Use the nZ entry to define a specified number *n* of binary zeros that will be inserted in the output record. The repetition factor *n* may be any number between 1 and 4095 inclusive. The Z entry must be coded to the immediate right of *n*.

SEQNUM Use SEQNUM to create a sequence number field within the output record. The length of the field can be from 1 to 16 bytes and can be represented in either BI, PD, or ZD formats. In addition, a starting value and an increment can be specified for the field.

The following describes the SEQNUM variables and parameters:

l Represents the length in bytes of the field to be created. A value from 1 to 16 can be specified.

f Indicates the format of the field to be created. BI, PD, or ZD can be specified to create an unsigned binary field, a packed decimal field, or zoned decimal field, respectively.

START Optionally specifies a starting number *n* for the field. The *n* value can be 0 through 2,147,483,647. The default is 1.

OUTREC

INCR Optionally specifies a value *i* that indicates how sequence numbers should be incremented. The *i* value can be 1 through 65,535. The default is 1.

The maximum sequence number generated is limited to 15 decimal digits or the length of the output field. If a number is reached that would exceed the limit, SyncSort will truncate the high-order digit and continue processing. Thus, sequence numbers will cycle within the limit. For example, if the output field is 2 bytes, then 99 will be the highest sequence number. The next number, 100, will have its high-order digit truncated. The resulting number, 00, starts a new sequence number cycle from 00 to 99, regardless of the START value.

Generating Run-Time Date and Time Constants

You can insert the date and time of your SyncSort run into your records. The following table shows the constants generated by the run-time date and time parameters.

A 'C' in the output format denotes a character constant, while a 'P' denotes a packed decimal constant. Packed decimal constants contain a positive sign and a leading zero when padding is necessary.

A '(c)' in the parameter represents a separator character. A blank used as the separator character must be enclosed in apostrophes. An apostrophe used as the separator character must be specified as two apostrophes enclosed within apostrophes ('').

Parameter	Output	Length (Bytes)
&DATE	C'mm/dd/yy'	8
&DATE1	C'yyyymmdd'	8
&DATE1(c)	C'yyyymmdd'	10
&DATE1P	P'yyyymmdd'	5
&DATE2	C'yyyymm'	6
&DATE2(c)	C'yyyymm'	7
&DATE2P	P'yyyymm'	4
&DATE3	C'yyyddd'	7
&DATE3(c)	C'yyyddd'	8
&DATE3P	P'yyyddd'	4
&DATE=(m ₁ m ₂ m ₃ m ₄)	(see description below table)	
&DATENS=(xyz)	(see description below table)	
&TIME	C'hh:mm:ss'	8
&TIME1	C'hhmmss'	6
&TIME1(c)	C'hhmmss'	8
&TIME1P	P'hhmmss'	4
&TIME2	C'hhmm'	4
&TIME2(c)	C'hhmm'	5
&TIME2P	P'hhmm'	3
&TIME3	C'hh'	2
&TIME3P	P'hh'	2
&TIME=(hp)	(see description below table)	
&TIMENS=(tt)	(see description below table)	

Table 20. Run Time Constants

OUTREC

&DATE=(m₁m₂m₃m₄) This form of the &DATE subparameter generates the current system date and controls the formatting of the date. You can specify the position of the year, month, and date; specify a separator character; and choose between 2-digit and 4-digit year representation.

The positions m_1 through m_4 represent masks used to format the date. To specify the positions of the month, day, and year, replace the m_1 , m_2 and m_3 positions, in any order, with M for the month (01-12), D for the day (01-31), and either Y or 4 for the year (where Y is a 2-digit year and 4 is a 4-digit year). Replace the m_4 position with a separator character.

For example, to print the date with the form *yy-mm-dd*, specify &DATE=(YMD-). For December 31, 1997, the date would appear as "97-12-31".

A blank used as the separator character must be enclosed in apostrophes. An apostrophe used as the separator character must be specified as two apostrophes enclosed within apostrophes ('').

The field for this form of &DATE requires 8 bytes for a 2-digit year representation and 10 bytes for a 4-digit year. The M, D, and Y or 4 may only appear once in the mask. All four positions must be specified.

&DATENS=(xyz) specifies that the current date is to appear in the output record in the form 'xyz', where x, y, and z indicate the order in which the month, day, and year are to appear and whether the year is to appear as two or four digits. For x, y, and z, use M to represent the month (01-12), D to represent the day (01-31), Y to represent the last two digits of the year (for example, 02), or 4 to represent the four digits of the year (for example, 2002). M, D, and Y or 4 can each be specified only once.

For example, &DATENS=(DMY) would produce a date of the form 'ddmmyy' which on March 29, 2002, would appear as '290302'. &DATENS=(4MD) would produce a date of the form 'yyyymmdd' which on March 29, 2002, would appear as '20020329'. x, y, and z must be specified.

&TIME=(hp) This form of the &TIME subparameter generates the current system time of day and controls the formatting of the time. You can print the time in 24-hour or 12-hour formats and specify the separator character between the hours, minutes and seconds.

The format for 24-hour time is *hhpmpss*, where *hh* represents the hour (00-23), *mm* represents minutes (00-59), *ss* represents seconds (00-59), and *p* represents the separator character as specified by *p* in the &TIME=(hp) subparameter.

The format for 12-hour time is *hhpmpss nn*, where *hh* represents the hour (01-12), *mm* represents minutes (00-59), *ss* represents seconds (00-59), and *p* represents the separator character as specified by *p* in the &TIME=(hp) subparameter. The *nn* is "am" or "pm" as appropriate.

To select 12-hour mode specify *h* as 12; to select 24-hour mode specify *h* as 24. The *p* specification represents the character to use as a separator.

For example, to display the time in a 12-hour format with a period as a separator, specify `&TIME=(12.)`. At 22:43:23 hours, the time would appear as "10.43.23 pm".

A blank used as the separator character must be enclosed in apostrophes. An apostrophe used as the separator character must be specified as two apostrophes enclosed within apostrophes (``').

The field for this form of the `&TIME` subparameter requires 8 bytes for the 24-hour format and 11 bytes for the 12-hour format.

`&TIMENS=(tt)` specifies that the current time is to appear in the output record in the form 'hhmmss' (24-hour time) or 'hhmmss xx' (12-hour time). If `tt` is 24, the time is to appear in the form 'hhmmss' (24-hour time) where `hh` represents the hour (00-23), `mm` represents the minutes (00-59), and `ss` represents the seconds (00-59).

For example, `&TIMENS=(24)` would produce a time of the form 'hhmmss' which at 08:25:13 pm would appear as '202513'. If `tt` is 12, the time is to appear in the form 'hhmmss xx' (12-hour time) where `hh` represents the hour (01-12), `mm` represents the minutes (00-59), `ss` represents the seconds (00-59), and `xx` is either 'am' or 'pm'.

For a second example, `&TIMENS=(12)` would produce a time of the form 'hhmmss xx' which at 08:25:13 pm would appear as '082513 pm'.

For an example of an `OUTREC` control statement that generates run-time constants, see Figure 54 on page 2.120.

The EDIT Subparameter

The `EDIT` subparameter lets you create your own edit patterns for converted numeric data. An edit pattern can consist of:

- Significant digit selectors.
- Leading insignificant digit selectors.
- Sign replacement characters.
- Any other characters to be printed in the actual output.

The edit pattern can be up to 22 characters in length, with a maximum of 15 leading insignificant and/or significant digits.

The characters used to represent significant or insignificant digit selectors are determined by the keyword `EDIT`. If `EDIT` is specified, the letter `I` represents leading insignificant digits which will print as blanks if the digits are zeros, and the letter `T` represents significant digits (digits that will print in their true form, even as leading zeros).

OUTREC

The keyword **EDIT** can be specified with replacements for the letters **I** and/or **T**. Any printable character can be used as a replacement character. This replacement makes available to the user a pattern which encompasses all printable characters.

The figure below illustrates the concept of replacing the insignificant and significant digit selectors **I** and **T** with other characters.

EDxy=
where:
x = insignificant digit selector
y = insignificant digit selector

Figure 45. Replacing Digit Selector Characters

When a blank, quotation mark or unbalanced parenthesis appears within an **EDIT** pattern, the entire pattern must be enclosed within single quotation marks. Balanced parentheses need not be enclosed within quotation marks. A single quotation mark within the pattern (i.e., an apostrophe) must be specified as two apostrophes.

All other characters are printed as specified in the edit pattern, with the following exceptions:

- Any character specified after the first leading insignificant digit selector and before the first significant digit selector will print as a blank, unless a previously selected digit was non-zero.
- Any character specified after the last significant digit selector will print as a blank if the edited number is positive.
- Any character or character string specified before the first leading insignificant digit selector, including a leading sign character, will print to the immediate left of the first significant digit. The appropriate number of leading blanks will be supplied, assuring that the total number of characters in the printed field corresponds to the total number of characters in the edit pattern.
- Any leading insignificant digit selector specified after the first significant digit selector will be treated as a significant digit selector.
- The sign replacement character appearing as the first and/or last character of the pattern is replaced as per the **SIGNS** subparameter.

The LENGTH=n Subparameter

Use the **LENGTH=n** subparameter to alter the default length of the output field data:

- When an editing mask is used, the default length is determined by the edit pattern and the format of the field. If LENGTH=n is not specified, the length is equal to the number of characters specified in the edit pattern. If LENGTH=n is specified, the edit pattern will either be truncated on the left or padded with blanks on the left so that the length of the pattern equals the *n* value.

The maximum value which can be specified for *n* when editing masks are used is 22.

- When an output data format f_o is used, the default length is 4 for BI and FI formats, and is determined by the number of digits in the input expression for CFS/FS, PD and ZD formats. (The number of digits is 15 for any input expression other than a single p,l,f_i field.) If LENGTH=n is specified, the output data will either be truncated on the left or padded on the left with zeros (or blanks for CSF/FS) of the appropriate format to a length of *n*.

The following are the maximum values that can be specified for *n* when an output data format f_o is used:

Output Format	Maximum Value of n
BI	4
FI	4
CSF	16
FS	16
PD	8
ZD	15

Table 21. Maximum Values of LENGTH=n for Output Data

The Mm Subparameter (Editing Masks)

SyncSort for z/OS provides editing masks to simplify the more common editing operations. If neither *Mm* nor EDIT is specified in the OUTREC control statement, M0 is used to edit BI, FI, PD, PD0, ZD, and CSF/FS fields and M11 is used to edit DT1, DT2, DT3, TM1, TM2, TM3, and TM4 fields.

OUTREC

Mask	Pattern	Signs	Length
M0	IIIIIIIIIIIIIIITS	(, , ' ', -)	d+1
M1	TTTTTTTTTTTTTTTS	(, , ' ', -)	d+1
M2	I, III, III, III, IIT.TTS	(, , ' ', -)	d+1 + [d/3]
M3	I, III, III, III, IIT.TTCR		d+2 + [d/3]
M4	SI, III, III, III, IIT.TT	(+, -)	d+1 + [d/3]
M5	SI, III, III, III, IIT.TTS	(' ', (, ' ',))	d+2 + [d/3]
M6	III-TTT-TTTT		12
M7	TTT-TT-TTTT		11
M8	IT:TT:TT		8
M9	IT/TT/TT		8
M10	IIIIIIIIIIIIIIIT		d
M11	TTTTTTTTTTTTTTTT		d
M12	SIII, III, III, III, IIT	(' ', -)	d+1 + [(d-1)/3]
M13	SIII.III.III.III.IIT	(' ', -)	d+1 + [(d-1)/3]
M14	SIII III III III IITS	(' ', (, ' ',))	d+2 + [(d-1)/3]
M15	III III III III IITS	(, , ' ', -)	d+1 + [(d-1)/3]
M16	SIII III III III IIT	(' ', -)	d+1 + [(d-1)/3]
M17	SIII'III'III'III'IIT	(' ', -)	d+1 + [(d-1)/3]
M18	SI, III, III, III, IIT.TT	(' ', -)	d+1 + [d/3]

Table 22. (Page 1 of 2) Editing Masks

Mask	Pattern	Signs	Length
M19	SI.III.III.III.IIT,TT	(' ','-)	d+1 + [d/3]
M20	SI III III III IIT,TTS	(' ','(',' ','),')	d+2 + [d/3]
M21	I III III III IIT,TTS	(,,' ','-)	d+1 + [d/3]
M22	SI III III III IIT,TT	(' ','-)	d+1 + [d/3]
M23	SI'III'III'III'IIT.TT	(' ','-)	d+1 + [d/3]
M24	SI'III'III'III'IIT,TT	(' ','-)	d+1 + [d/3]
M25	SIIIIIIIIIIIIIIIT	(' ','-)	d+1
M26	STTTTTTTTTTTTTTTT	(+,-)	d+1

Table 22. (Page 2 of 2) Editing Masks

Notes:

- M0 is the default mask
- The letter *d* represents the number of resulting digits after data conversion. The mask patterns in the Pattern column show the maximum number of resulting digits, which is 15. (Refer to Table 16 on page 2.98.)
- The bracket symbols indicate that only the integer part of this division should be retained.

The *Editing Masks* table illustrates the following for each of the available masks.

- Edit pattern.
- Leading or trailing signs, where appropriate.
- Length. If a SyncSort editing mask is used for totaled or subtotaled data, the length of the output field is determined by the maximum permissible length of the data format, not by the specified length of the input field. The subparameter LENGTH can be used to override the length of the output field.

The edit patterns use the same symbolic letters used in the EDIT subparameter. Leading insignificant digits are represented by the letter I; significant digits are represented by the

OUTREC

letter T. Leading or trailing sign replacement characters are represented by the letter S. All other characters print as they appear in the pattern.

The SIGNS illustrated for each mask follow the format requirements of the SIGNS subparameter. You can specify the SIGNS subparameter to selectively override the signs for a particular mask. For example, if you specify mask M4 and also specify SIGNS=(' '), a leading blank will print instead of a plus sign if the number is positive. However, a leading minus sign will print if the number is negative because the leading negative sign specified in the editing mask has not been overridden.

The lengths in the table represent the length, in bytes, of the mask. The lengths of masks M0-M5 and M10-M26 are determined, in part, by the number of digits *d*. Refer to Table 16 on page 2.98 to determine the number of digits for each type of numeric field.

The SIGNS Subparameter

The SIGNS subparameter specifies the sign(s) that will appear before or after the edited number.

The sign replacement character, normally 'S', has special meaning if it appears as the first or last character in an edit pattern. In these positions, the sign replacement character will be replaced, as appropriate, by the characters specified by the SIGNS subparameter.

The format of the SIGNS subparameter is illustrated below.

SIGNS=(s₁,s₂,s₃,s₄)

where:

s₁= leading positive sign indicator

s₂= leading negative sign indicator

s₃= trailing positive sign indicator

s₄= trailing negative sign indicator

Because the SIGNS subparameter contains four positional values, commas must be used to indicate embedded, unspecified values. Each of the four values can contain one, and *only* one, character; specified characters must be separated by commas.

A blank, comma, quotation mark and unbalanced parenthesis used as a SIGNS character must be enclosed within apostrophes. An apostrophe used as a SIGNS character must be specified as two apostrophes enclosed within apostrophes ("").

When the SIGNS subparameter is specified, the letter 'S' is normally used as the sign replacement character in the user-supplied edit pattern. The user can change the last letter of the keyword SIGNS in order to specify another character as the sign replacement character. For example, if the user specifies SIGNX instead of SIGNS, the letter 'X' becomes the sign replacement character in the user-provided edit pattern.

If the user specifies a sign replacement character in the edit pattern but does not specify a value in the corresponding position in the SIGNS parameter, a blank will be assumed. For example, if the user specifies the following:

```
EDIT=(IITT.TTS) , SIGNS=( , , , -)
```

then a trailing minus sign will print if the number is negative and a trailing blank will print if the number is positive.

The SIGNS subparameter can also be used to override the sign values in SyncSort-provided editing masks.

The CHANGE Subparameter

The CHANGE subparameter changes an input field to a *replacement* constant in the reformatted output record if the input field equals a *search* constant. The input field remains unchanged on the input side.

The format of the CHANGE subparameter is shown below:

```
[c:] p,l, CHANGE=(o,srch1,repl1 [,srch2,repl2,...srchn, repln])
[ ,NOMATCH=( { nmrepl } ) ]
```

Figure 46. Change Subparameter

Multiple search-replacement paired constants, with different data formats, can be specified on a CHANGE subparameter. Note the following rules for mixing data formats:

- Search constants are character, hexadecimal, or binary strings. Multiple search constants on a CHANGE subparameter can be a mixture of character and hexadecimal formats. Binary search constants cannot be mixed with search constants of other formats; thus, if one search constant on a CHANGE subparameter is binary, all other search constants on that subparameter must also be binary.

OUTREC

- Replacement constants are character or hexadecimal strings. Multiple replacement constants on a CHANGE subparameter can be a mixture of character and hexadecimal data formats.
- The constants of a search-replacement pair can be of different data format. For example, a hexadecimal or binary search constant could be paired with a character replacement constant, or a character search constant could be paired with a hexadecimal replacement constant. Thus, you could change a hexadecimal or binary input field to a character output field, or you could change a character input field to a hexadecimal output field.

The following describes the elements of the CHANGE subparameter:

p,l The normal SyncSort position-length designation that specifies the input field. When this field matches a search constant, the field will be changed in the output to a replacement constant.

For character or hexadecimal search constants, the input field can be 1 to 64 bytes long. For binary search constants, the input field must be one byte.

o The length of the output replacement field. Permissible length is 1 to 64 bytes.

srch The search constant to which the input field is compared. Permissible formats are character string (C'x...x'), hexadecimal string (X'x...x'), or a binary byte (B'bbbbbbb'). When the search constant matches the input field, the input field will be changed to an output replacement constant.

If one of the search constants is binary in a set of search-replacement pairs on a CHANGE subparameter, then all the search constants on that CHANGE subparameter must be binary. (For additional information on using binary fields in INCLUDE/OMIT processing, see “INCLUDE/OMIT Control Statement” on page 2.16.)

If the search constant is longer than the length *l* of the input field, the constant will be truncated to length *l*. If the search constant is shorter than *l*, the constant will be padded on the right to length *l*. Character strings are padded with blanks (X'40'). Hexadecimal strings are padded with zeros (X'00'). Binary strings are neither truncated nor padded since only one-byte strings are permissible.

repl The replacement constant to which the input field is changed in the reformatted output record when the input field matches a search constant. Permissible formats are character string (C'x...x') and hexadecimal string (X'x...x').

If the replacement constant is longer than the length *o* of the output field, the constant will be truncated to length *o*. If the replacement constant is

shorter than *o*, the constant will be padded on the right to length *o*. Character strings are padded with blanks (X'40'). Hexadecimal strings are padded with zeros (X'00').

NOMATCH Indicates how SyncSort should respond if the input field does not match a search constant. If NOMATCH is not specified and no search constant matches the input field, sort processing will terminate with an error message.

nmrepl A replacement constant to which the input field is changed in the reformatted output record when the input field *p,l* fails to match a search constant. For details, see the description of the repl variable above.

r,n The position *r* and length *n* of an input field that will be inserted in the output record when the CHANGE input field *p,l* fails to match a search constant.

n must be at least 1. If *n* is greater than the length *o* specified for the output replacement field, the output field *r,n* will be truncated on the right to length *o*. If *n* is less than *o*, the field *r,n* will be padded on the right with blanks (X'40') to the length *o*.

The following example illustrates the use of the CHANGE subparameter:

```

OUTREC FIELDS=(16,2,
               CHANGE=(13,C'NJ',C'NEW JERSEY',
                       C'NY',C'NEW YORK',
                       C'PA',C'PENNSYLVANIA'),
               NOMATCH=(C'NOT SUPPORTED'),
               8X,
               24,1,
               CHANGE=(10,B'1.....',C'EAST COAST',
                       B'0.....',C'WEST COAST'))
    
```

Figure 47. Sample OUTREC Parameter with CHANGE Subparameter

In the above example, the FIELDS parameter contains two CHANGE subparameters. The first CHANGE subparameter changes the input field 16,2 to a state name in the reformatted output record when the input field matches a state code. If no matches are found, the output field will be 'NOT SUPPORTED.' The second change subparameter changes the one-byte input field 24,1 to 'EAST COAST' or 'WEST COAST' in the reformatted output record, depending on the binary contents of the input field.

The following example illustrates a situation that can arise when using binary search constants. In such cases, more than one search constant may match an input field:

OUTREC

```
OUTREC FIELDS=(24,1,
               CHANGE=(6,B'.....11.',C'SHARE',
                       B'.....1.',C'UNIQUE'))
```

Figure 48. CHANGE Subparameter with Binary Search Constants

Note that in the above example, the input field X'06' would match both binary search constants. In such cases, the first search constant is used, thus the output would be the character string 'SHARE'. If the input field were X'02', the output would be the character string 'UNIQUE'.

CONVERT Parameter (Optional)

The CONVERT parameter enables you to convert variable-length records into fixed-length records.

These records do not require an RDW and will be written to the output file(s) with a RECFM of F or FB. When using CONVERT, you no longer need to apply the rules for "Specifying the FIELDS parameter for Variable-Length Records."

You may create multiple output files with different record formats when specifying CONVERT in conjunction with the OUTREC parameter on the OUTFIL control statement. Refer to the explanation of CONVERT in the OUTFIL control statement description for facilities available when using CONVERT with OUTFIL.

You cannot specify the variable portion of the input records (position without length) when using CONVERT. However, all data fields need not be present in each record being CONVERTed, unless a numeric or year data field is specified. That is, blanks will be used as a default for any missing *p,l* field bytes, while all *p,l,f* fields must be present. See VLFILL for how to change the default character if you use the OUTREC parameter of the OUTFIL control statement.

Sample OUTREC Control Statements

Example 1

The following example illustrates how the OUTREC control statement can be used to insert binary zeros and blanks into the record.

```
OUTREC FIELDS=(1:4Z,5:20,10,23:44,28,10X)
```

Figure 49. Example 1, Sample OUTREC Control Statement

This OUTREC control statement defines a 60-byte record as follows:

- Four binary zeros are inserted in the first 4 bytes of the record (4Z).
- The next field begins in position 5. This field began in position 20 before OUTREC processing and is 10 bytes long (5:20,10).
- Eight blanks are inserted before the next field, which is positioned at byte 23. SyncSort for z/OS automatically inserts blanks in the unused positions between fields.
- The next field begins in position 23. This field began in position 44 before OUTREC processing and is 28 bytes long (23:44,28).
- Ten blanks are inserted in the last 10 bytes of the record (10X).

Example 2

The following example illustrates how the OUTREC control statement can be used to convert and edit numeric fields.

```
OUTREC FIELDS=(1,50,64,4,PD,M2,68,6,ZD,
              EDIT=($I,IIT.TTS),SIGNS=(,+, -))
```

Figure 50. Example 2, Sample OUTREC Control Statement

This OUTREC control statement defines a 70-byte output record as follows:

- The first field (1,50) begins in position 1. This field began in position 1 before OUTREC processing and is 50 bytes long.
- The next field (64,4) begins in position 51. This packed decimal field began in position 64 before OUTREC processing and is 4 bytes long. After being converted and edited by editing mask M2 (64,4,PD,M2) the resulting field will be 10 bytes long. However, the number of digits that will actually print will depend on the number of leading zeros, if any, because this mask specifies that only three digits must print whether or not they are leading zeros. Moreover, this mask specifies that a minus sign print after the number if it is negative and a blank print after the number if it is positive.
- The last field (68,6) begins in position 61. This zoned decimal field began in position 68 before OUTREC processing and is 6 bytes long. The EDIT and SIGNS subparameters (EDIT=(\$I,IIT.TTS),SIGNS=(,+, -)) specify a 10-byte field because 4 additional bytes are needed for the dollar sign, the comma, the decimal point and the trailing plus or minus sign. Note that if the first three digits are leading zeros, they will be suppressed.

OUTREC

Example 3

This example uses the OUTREC control statement to convert numeric data from one format to another.

```
OUTREC FIELDS=(1,10,ZD,PD,
               11,4,FI,ZD,LENGTH=8)
```

Figure 51. Example 3, Sample OUTREC Control Statement

This OUTREC control statement defines a 14-byte output record as follows:

- The first field (1,10,ZD,PD) begins in position 1. This field was a 10-byte ZD field that began in position 1 before OUTREC processing. It will be converted to a 6-byte PD field in the output record, because 6 bytes are required to contain 10 decimal digits as a PD field.
- The next field (11,4,FI,ZD) begins in position 7. This field was a 4-byte FI field that began in position 11 before OUTREC processing. It will be converted to an 8-byte ZD field in the output record. Normally 10 ZD bytes would be required to contain the 10 decimal digits that may be represented by a 4-byte FI field, but the LENGTH=8 parameter overrode the output length. If there are more than 8 decimal digits in any of the 11,4,FI fields, those digits will be truncated on the left in the output record.

Note that ZD output is not the same as printable output using editing masks. High order zeros will appear as zeros in a ZD field, while they appear as blanks when using the default M0 mask, as well as most other masks. The sign indicator in a ZD field is placed in the first 4 bits of the rightmost byte, and not as a separate printable sign.

Example 4

This OUTREC example uses arithmetic and function operators to do algebraic calculations.

New 8-byte PD fields are required in each record containing the maximum and average of fields A, B and C. Another new 5-byte printable field is required containing field D as a percentage of field E. The field definitions are:

Field A:	1,4,PD
Field B:	5,8,ZD
Field C:	13,4,FI
Field D:	25,4,PD
Field E:	29,4,PD

The OUTREC control statement to accomplish this would be:

OUTREC FIELDS=(1,36,	Retain existing fields
40:(01,4,PD,ADD,	Field A plus
05,8,ZD,ADD,	Field B plus
13,4,FI),	Field C
DIV,+3,	divide by 3 to get average
PD,	output as 8-byte PD field
*	
50:01,4,PD,MAX,	Determine maximum of Field A and
05,8,ZD,MAX,	Field B and
13,4,FI,	Field C
PD,	output as 8-byte PD field
*	
60:+100,MUL,	100 times
25,4,PD,DIV,	Field D divided by
29,4,PD,	Field E
LENGTH=5)	output as printable 5-byte field
*	using default M0 mask

Figure 52. Example 4, Sample OUTREC Control Statement

This OUTREC control statement defines a 64-byte output record as follows:

- The first field (1,36) retains the complete contents of the input record.
- The second output field begins in position 40. An arithmetic calculation is done using three different numeric input fields and the constant +3 to compute the arithmetic average. This is an expression that is considered to contain 15 decimal digits. The output is requested as a PD field. The length of this field will be 8 bytes, since that is the length required to contain 15 decimal digits.
- The third output field begins in position 50. Multiplying numeric Field D by 100 before dividing by numeric Field E gives the desired percentage number, which is considered to contain 15 decimal digits. No output format or editing mask is specified, so the default mask M0 is used to create printable output. LENGTH=5 is specified to reduce the default length of the output field from 16 to 5, since it is known that the percentage number will not be large.

OUTREC

Example 5

This OUTREC control statement uses DT1, TM1, and edit masks to convert SMF date and time values to appropriate formats.

```
OUTREC FIELDS=(1,4,DT1,EDIT=(TTTT/TT/TT),
                3X,5,4,TM1,EDIT=(TT:TT:TT))
```

Figure 53. Sample OUTREC Control Statement

The following shows how the output would be formatted:

```
2002/07/04 07:22:12
2002/07/04 05:15:25
2002/07/05 11:37:39
2002/07/05 16:42:28
```

Example 6

This OUTREC control statement illustrates the use of the &DATE1(c) and &TIME1(c) parameters in a SyncSort run on June 9, 2002 at 04:16:29 p.m.

```
OUTREC FIELDS=(8,20,24:&DATE1(' '),X,&TIME1(:))
```

Figure 54. Sample OUTREC Control Statement

The output would include data from the input record in the first twenty columns followed by the run-time date and time starting in column 24. The date and time would appear as '2002 06 09 16:16:29'.

Example 7

The following control statements illustrate two of the options of the TRAN subparameter.

This OUTREC control statement uses TRAN=LTOU to translate the letters in positions 1-5 of each output record from lowercase to uppercase.

```
OUTREC FIELDS=(1,5,TRAN=LTOU)
```

Figure 55. Sample OUTREC Control Statement

For example, 'Ab,Cd' would translate to 'AB,CD'.

This OUTREC control statement uses TRAN=ALTSEQ to translate each binary zero (X'00') in columns 1-5 to an asterisk (X'5C') in positions 1-5.

```
ALTSEQ CODE=(005C)
OUTREC FIELDS=(1,5,TRAN=ALTSEQ)
```

Figure 56. Sample OUTREC Control Statement

Comprehensive examples illustrating the OUTREC control statement and the OUTREC parameter of the OUTFIL control statement are provided in “Chapter 3. How to Use SyncSort’s Data Utility Features”.

Sample OUTREC Control Statements with CENTWIN Processing

For century window processing, data conversion is determined by the century window defined by the CENTWIN parameter.

The following provides examples of data conversion with CENTWIN:

Example 1

A 2-digit year field in character format at position 20 in the input record could be expanded with the following specification:

```
OUTREC FIELDS=(1,19,      * Copies first 19 bytes of record
                20,2,Y2C, * Converts 2-digit year data to 4-digit year
                22,59)    * Copies remaining 59 bytes
```

Figure 57. Example 1, OUTREC Control Statement with Year Data

Note that the expansion of the year data from 2 to 4 digits increases the output record length by 2 bytes compared to the input record length.

The CENTWIN setting determines the century of the 2-digit year field. If CENTWIN=1980, then a year field in the input record would be converted as follows:

SORTIN Input	OUTREC Output
13	2013
79	2079
80	1980
92	1992

Example 2

Consider the following packed decimal date field at position 20 in the input record:

```
yymmdd = X'0yymmddC'
```

OUTREC

Suppose you want to output a displayable 4-digit year in character format in the form

mm/dd/yyyy

To accomplish this, specify the following OUTREC control statement:

```
OUTREC FIELDS=(1,19,          * Copies first portion of record
                  21,2,PD0,M11, * Converts X'yymm' to X'mm' then C'mm'
                  C'/',        * Inserts slash
                  22,2,PD0,M11, * Converts X'mddC' to X'dd'then C'dd'
                  C'/',        * Inserts slash
                  20,2,Y2P,     * Converts X'0yym' to X'yy' then C'yyyy'
                  24,76)        * Copies rest of record
```

Figure 58. Example 2, OUTREC Control Statement with Year Data

The 4-digit year output from the input year field (20,2,Y2P) depends on the CENTWIN setting. The following sample of input and output data shows the case for CENTWIN=1980:

SORTIN Input Date Field	OUTREC Output Date Field
X'0800329C'	03/29/1980
X'0790603C'	06/03/2079

Example 3

To expand a 3-byte packed decimal date field of the form X'yyddd's', at position 20 in the input record, to a 4-byte packed field of the form X'yyyyddd's' that contains a prefixed century value, specify an OUTREC control statement such as the following:

```
OUTREC FIELDS=(1,19,          * Copies first portion of record
                  20,1,Y2ID,   * Converts X'yy' to X'yyyy'
                  21,60)        * Copies rest of record starting with
*                               the X'ddd's' of the date field
```

Figure 59. Example 3, OUTREC Control Statement with Year Data

Note that in the above example the output record length will be 1 byte larger than the input record length. The following sample of input and output data shows the effect for CENTWIN=1980:

SORTIN Input Date Field	OUTREC Output Date Field
X'79'	X'2079'
X'80'	X'1980'

Example 4

To expand a 4-byte packed decimal date field of the form X'0yymmdds', at position 20 in the input record, to a 5-byte field of the form X'0yyyyymmdds' that contains a prefixed century value, specify an OUTREC control statement such as the following:

```

OUTREC FIELDS=(1,19,          * Copies first portion of record
                20,2,Y2IP,    * Converts X'0yym' to X'0yyyyym'
                22,59)        * Copies rest of record starting with
*                               * the X'mdds' of the date field

```

Figure 60. Example 4, OUTREC Control Statement with Year Data

As with Y2ID conversion, the output record length will be 1 byte larger than the input length. The following sample of input and output data shows the effect for CENTWIN=1980:

SORTIN Input Date Field	OUTREC Output Date Field
X'0790'	X'020790'
X'0801'	X'019801'

Example 5

Consider a 2-byte character or zoned decimal field that may contain either valid numeric year data or characters that identify the record as a header or trailer. Header records in the example are identified by zeros (X'00') or a blank (X'40') in the first byte of the year field, while trailer records are identified by binary ones (X'FF') in the first byte of the field. The Y2S format will treat the valid year data normally, in the same way as the Y2C or Y2Z formats would treat the data, but the year fields of header and trailer records will be converted to a 4-digit form padded on the left with data identical to the data in the first byte of the input field.

Typically this type of conversion is needed when a Y2S SORT or MERGE field is used to collate the records so that header/trailer records in the output remain at the start or end of the file. An OUTREC control statement such as the following could be used.

```

OUTREC FIELDS=(1,19,          * Copies first portion of record
                20,2,Y2S,    * Converts C'yy' to C'yyyy' and pads
*                               * fields that identify header/trailer records
                22,59)        * Copies the remaining fields

```

Figure 61. Example 5, OUTREC Control Statement with Year Data

As with Y2C or Y2Z, the output record length will be 2 bytes larger than the input record length.

For CENTWIN=1990, the sorted Y2S field would be converted as follows:

OUTREC

SORTIN Input Date Field	OUTREC Output Date Field
X'4001'	X'00000000' (from 4th input record)
X'F9F8'	X'40404001' (from 1st input record)
X'F0F3'	X'F1F9F9F8' (from 2nd input record)
X'0000'	X'F2F0F0F3' (from 3rd input record)
X'FFFF'	X'FFFFFFFF' (from 5th input record)

RECORD Control Statement

The RECORD control statement provides record length and format information. It is required in the following situations:

- SyncSort is invoked by a program passing either a 24-bit or 31-bit extended parameter list and using an in-memory E15 or E32 exit routine.
- An E15 or E35 exit routine changes the record length.

RECORD Control Statement Format

The format of the RECORD control statement is illustrated below:

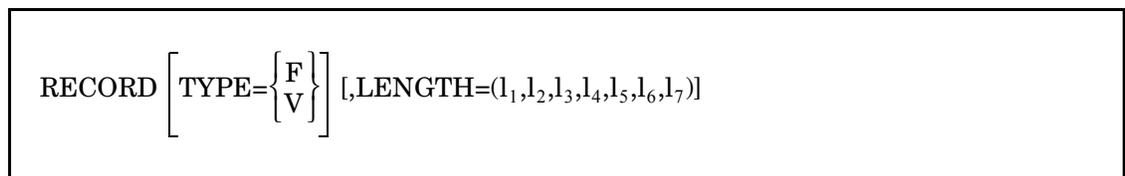


Figure 62. RECORD Control Statement Format

TYPE Parameter (Optional)

The TYPE parameter can be used to indicate the record format. TYPE=F indicates fixed-length records; TYPE=V indicates variable-length records. TYPE=FB or TYPE=VB can be specified but the 'B' is ignored.

TYPE should be specified if SORTIN is VSAM. If TYPE is not provided, the SORTOUT RECFM will be examined to determine the SORTIN TYPE. If no SORTOUT RECFM is found, TYPE=V will be assumed if the SORTOUT is VSAM and TYPE=F if the SORTOUT is non-VSAM.

Note: If the TYPE specification differs from the RECFM DCB parameter for the SORTIN/SORTINnn DD statement, the latter takes precedence.

LENGTH Parameter (Conditionally Required)

The LENGTH parameter, usually optional, is required whenever the RECORD control statement is required.

The LENGTH parameter specifies the length of the record at various points during the processing of the application.

The number of length values can vary from 1 to 7. Only the l1, l2 and l3 values should be specified for fixed-length records and for merge or copy applications. All seven length values can be specified for variable-length sorts. If l1 is the only value specified, parentheses

RECORD

are optional. If l1 and additional length values are specified, they all must be enclosed in parentheses.

The length values are positionally dependent. An extra comma must indicate a missing length value between any two that are specified. Commas need not follow the final length value specified. For example, if LENGTH=(l1,,,l4) is specified, the omitted values are understood to be l2 and l3.

The l1,...,l7 variables specify the following:

l1 The maximum record input length of the logical records. For variable-length records, this is the length of the longest logical record plus the 4-byte Record Descriptor Word. The 4-byte RDW must be included, even if the input is a VSAM file. The maximum record length cannot exceed 32,760 for fixed-length records and 32,767 for variable-length records. An LRECL value specified on the SORTIN/SORTINnn DD statement or the data set label will override the l1 value for fixed-length records. For variable-length records, the higher value (LRECL or l1) is used.

l2 The maximum length of the logical records *after* E15 processing. An omitted l2 value defaults to the l1 value and indicates that the maximum record length has not been changed by an E15 exit. If there is no E15 exit, an l2 value which is smaller than the l1 value or the LRECL specified on the SORTIN/SORTINnn DD statement or data set label will truncate the records. This truncation will occur after the record is read from SORTIN.

l3 The maximum length of the logical records after E35 processing. If the l3 value is omitted, the default is either the l2 value, or, if an INREC and/or OUTREC control statement is specified, the record length after INREC/OUTREC processing. Note that it is not necessary to specify an l3 value to reflect a length change due to INREC or OUTREC processing; the revised record length is calculated automatically. However, it *is* necessary to specify an l3 value if exit E35 has altered the record length.

The LRECL value specified in the SORTOUT DD statement should either correspond to the l3 value or the LRECL specification should be omitted. In the latter case, SyncSort will automatically calculate the correct LRECL value.

The l3 value is ignored if there is no E35 exit, so it is not possible to use the l3 value to truncate or pad the records.

l4 The minimum length of the variable-length logical records plus the 4-byte Record Descriptor Word. An omitted l4 value defaults to the length from the beginning of the record to the end of the last field referenced by *any* control statement.

- 15** The most frequent record length of the variable-length records. Specify this length value to optimize the size of the segment, i.e., the fixed-length block of main storage, used to contain variable-length records.
- 16** The average work space required by each record, as reported by the HISTOGRM utility program. The 16 value will be ignored for a Tape Sort.
- 17** The segment length recommended by the HISTOGRM utility program. If 17 is omitted, the SIZE parameter on the SORT control statement may be used to determine the impact of segment size on sort performance. Assuming the SIZE parameter reports a SORTIN data set of at least 10,000 records, SyncSort may sample the first 100-200 records to calculate an approximate segment size. An installation may decide to allow record sampling for smaller files. The 17 value will be ignored for a Tape Sort.

Rules for Specifying the Length Parameter

Observe the following rules when specifying length values:

- *All* length values for variable-length records must include 4 bytes for the Record Descriptor Word.
- The 11, 12, and 13 values must represent the *maximum* record lengths and the 14 value must represent the *minimum* record length. If SyncSort encounters a record which exceeds the maximum length or is shorter than the minimum length, the application will either terminate abnormally or produce unpredictable results.

Sample RECORD Control Statement

```
RECORD TYPE=F, LENGTH=(80, , 60)
```

Figure 63. Sample RECORD Control Statement

This sample RECORD control statement defines the record as follows:

- The file contains fixed-length records.
- The input record length (11) is 80 bytes.
- A comma represents the omitted 12 value because an E15 exit does not change the record length.
- The record length after INREC/OUTREC and/or E35 processing is 60 bytes. The SORTOUT LRECL should either be specified as 60 or omitted. If it is omitted, SyncSort will automatically supply the correct value.

RECORD

```
RECORD TYPE=V,LENGTH=(400,300,250,120,200,280,230)
```

Figure 64. Sample RECORD Control Statement

This sample RECORD control statement defines the record as follows:

- The file contains variable-length records. All length values include 4 bytes for the Record Descriptor Word.
- The maximum input record length is 400 bytes.
- The maximum record length after E15 processing is 300 bytes.
- The maximum record length after INREC/OUTREC and/or E35 processing is 250 bytes.
- The minimum record length is 120 bytes.
- The most frequent record length is 200 bytes.
- The average work space required for each record is 280 bytes, as reported by the HISTOGRM utility program.
- The segment length recommended by HISTOGRM is 230 bytes.

In the above example, the 14, 15, 16 and 17 values will be ignored if the application is a merge or copy.

SORT Control Statement

The SORT control statement defines the application as a sort or copy application.

Either a SORT control statement or a MERGE control statement is required for every application.

Cultural Environment Support

Cultural environment support allows you to choose an alternative set of collating rules based on a specified national language. The alternative collating applies to SORT/MERGE and INCLUDE/OMIT processing.

For additional detail, see “LOCALE” on page 5.20.

SORT Control Statement Format

The format of the SORT control statement is illustrated below.

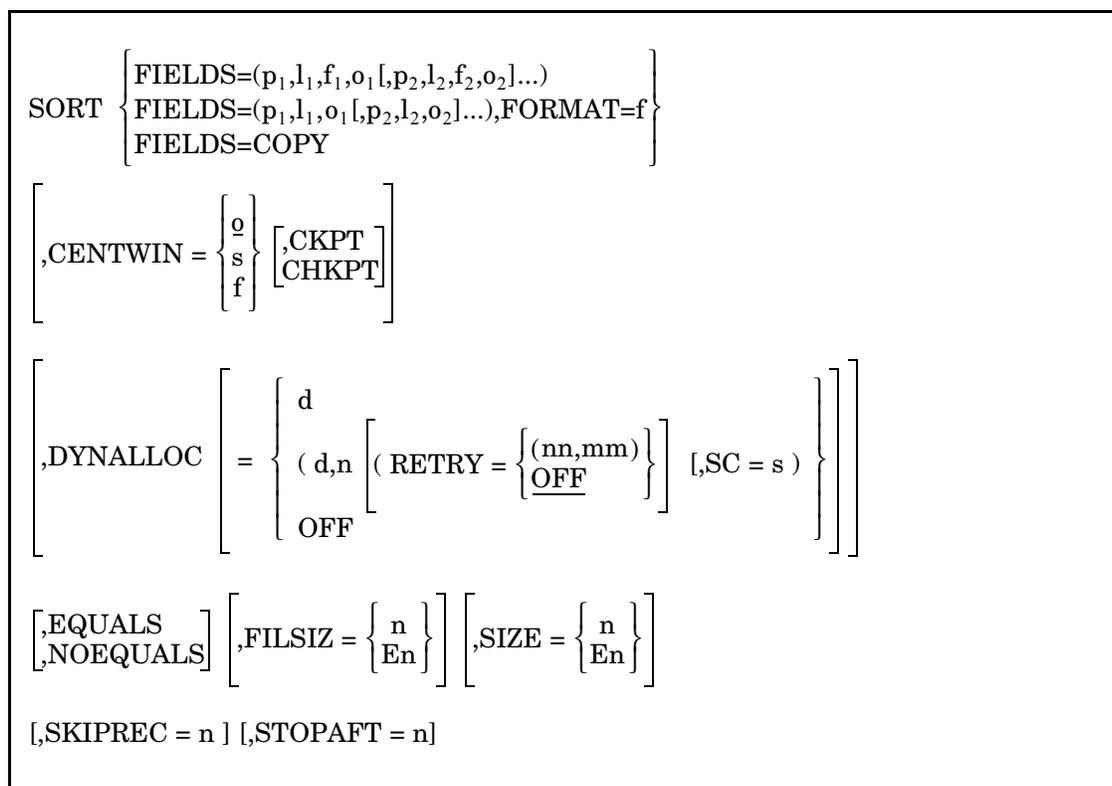


Figure 65. SORT Control Statement Format

FIELDS Parameter (Required)

The FIELDS parameter is required. It describes the control fields.

SORT

List the control fields in order of greatest to least priority, with the primary control field listed first, followed by progressively less significant fields. You can specify up to 128 control fields; however, if fields are complex, the limit for a particular execution may be less than 128.

Each field specified in the FIELDS parameter is identified by its position (p), length (l), format (f) and order (o).

p The position value indicates the first byte of the field relative to the beginning of the input record *after* INREC and/or E15 processing, if specified, have completed.

Binary control fields can begin on any bit of a byte. When a binary field does not begin on a byte boundary, you must specify the bit number (0-7). For example, a position value of 21.3 refers to the 4th bit of the 21st byte of the record.

l The length value indicates the length of the control field. The length value must be an integer number of bytes except for the length of a binary control field which can be specified in bits. For example, a length value of 0.5 refers to a binary control field 5 bits long.

For signed fields, the length value must include the area occupied by the sign.

f The format value indicates the data format. For a list of valid formats, refer to the table in the next section, "Valid Formats for Sort Control Fields." If all the control fields have the same format, you can specify the format value once by using the FORMAT=f subparameter. If you specify both the individual *f* values and the FORMAT subparameter, the individual *f* values will be used. (Note that the *f* values must be specified for each control field).

o The order value indicates how the field is to be collated:

- A=Ascending order
- D=Descending order
- E=As modified by an E61 exit. Ascending order

Valid Formats for Sort Control Fields

The following chart lists the valid formats for sort control fields.

Code	Data Format	Field Length (bytes)
AC*	EBCDIC characters are translated to their ASCII equivalents before sorting.	1 to 4091†
AQ*	Character. Records are sorted according to an alternate sequence specified either in the ALTSEQ control statement or as an installation default.	1 to 4091†
ASL*	Leading separate sign. An ASCII + or - precedes numeric field. One digit per byte.	2 to 256
AST*	Trailing separate sign. An ASCII + or - trails numeric field. One digit per byte.	2 to 256
BI	Binary. Unsigned.	1 bit to 4092**
CH	Character. Unsigned.	1 to 4092**
CLO* OL*	Leading overpunch sign. Hexadecimal F,C,E, or A in the first 4 bits of your field indicates a positive number. Hexadecimal D or B in the first 4 bits indicates a negative number. One digit per byte. CMP=CLC is forced.	1 to 256
CSF FS	Floating sign format. An optional leading sign may be specified immediately to the left of the digits. If the sign is a -, the number is treated as negative. For other characters, the number is treated as positive. Characters to the left of the sign are ignored.	1 to 16
CSL* LS*	Leading separate sign. An EBCDIC + or - precedes numeric field. One digit per byte. CMP=CLC is forced.	2 to 256
CST* TS*	Trailing separate sign. An EBCDIC + or - follows numeric field. One digit per byte. CMP=CLC is forced.	2 to 256
FI	Fixed point. Signed. (Equivalent to Signed Binary.)	1 to 256
FL	Floating point. Normalized. Signed.	2 to 16
PD	Packed decimal. Signed.	1 to 256

Table 23. (Page 1 of 3) Format Code Chart

SORT

Code	Data Format	Field Length (bytes)
PD0*	Packed decimal. 2-8-byte packed decimal data with the first digit and trailing sign ignored. The remaining bytes are treated as packed decimal digits. Typically PD0 is used with century window processing and Y2P format; Y2P processes the year, while PD0 processes month and day.	2-8
Y2B*	Binary. 2-digit, 1-byte binary year data treated as a 4-digit year by CENTWIN (century window) processing.	1
Y2C*	Character. 2-digit character year data treated as a 4-digit year by CENTWIN (century window) processing. Processing is identical to Y2Z fields.	2
Y2D*	Packed decimal. 2-digit, 1-byte packed decimal year data treated as a 4-digit year by CENTWIN (century window) processing.	1
Y2P*	Packed decimal. 2-digit, 2-byte packed decimal year data. Of the four packed digits contained in the 2 bytes, the first digit and trailing sign are ignored; the two inner digits are treated as a 4-digit year by CENTWIN processing.	2
Y2S*	Character or zoned decimal. 2-digit, 2-byte valid numeric data treated as a 4-digit year by CENTWIN (century window) processing, as for Y2C and Y2Z. However, certain data are not treated as year data. Data with binary zeros (X'00') or a blank (X'40') in the first byte will be collated before valid numeric year data for ascending order (after year data for descending order). Data with all binary ones (X'FF') in the first byte will be collated after valid numeric year data for ascending order (before year data for descending order). Zones are ignored, as for Y2C and Y2Z, except for data where the first byte begins with X'00', X'40' or X'FF'.	2
Y2T* Y2U* Y2V* Y2W* Y2X* Y2Y*	Full-date, character, binary, or packed decimal formats. Full-date data formats can be used to sort or merge a variety of date fields. They can process dates ending or starting with year digits (x...xyy or yyx...x). They can also process non-date data commonly used with dates. For details, see page 2.140.	2-6

Table 23. (Page 2 of 3) Format Code Chart

Code	Data Format	Field Length (bytes)
Y2Z*	Zoned decimal. 2-digit, 2-byte zoned decimal year data treated as a 4-digit year by CENTWIN (century window) processing. The zones are ignored. Processing is identical to Y2C fields.	2
ZD CTO* OT*	Zoned decimal. Trailing overpunch in the first 4 bits of the right-most byte gives the sign. Hexadecimal F,C,E, or A indicates a positive number. Hexadecimal D or B indicates a negative number. One digit per byte. CTO forces CMP=CLC.	1 to 256
Notes: * <i>Cannot be used with Tape Sort.</i> ** <i>4084 for variable-length records.</i> † <i>2043 for variable-length records.</i>		

Table 23. (Page 3 of 3) Format Code Chart

For information on the year data formats (Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z) plus the related data format PD0 and the full-date formats, see “CENTWIN Parameter (Optional)” on page 2.134, “Converting Year Data with Century Window Processing on INREC, OUTREC, or OUTFIL OUTREC” on page 2.100, and “Specifying Field-to-Field Standard Comparisons for Year Fields” in the INCLUDE/OMIT Control Statement section of this chapter.

Rules for Specifying Sort Control Fields

- For fixed-length records, all control fields and the sum of their lengths cannot exceed 4092 bytes. When EQUALS is in effect, the number is reduced 4 bytes to 4088 bytes. EXTCOUNT also reduces the number by 4 bytes. Thus, if both EQUALS and EXTCOUNT are in effect, the number is reduced to 4084 bytes.
- For variable-length records, all control fields must be located within the first 4084 bytes, and the sum of their lengths cannot exceed 4084 bytes. When EQUALS is in effect, the number is reduced 4 bytes to 4080 bytes. EXTCOUNT also reduces the number by 4 bytes. Thus, if both EQUALS and EXTCOUNT are in effect, the number is reduced to 4076 bytes.
- Control fields can be in contiguous or non-contiguous locations in the record.
- Remember that for variable-length records, the first 4 bytes are reserved for the Record Descriptor Word, so the first byte of the *data* portion of the record is byte 5.
- If the output file is a key-sequenced VSAM cluster, the VSAM key must be the first control field specified.

SORT

Comparing PD and ZD Control Fields

The CMP PARM determines how PD and ZD control fields will be compared. When CMP=CPD is in effect, the Compare Decimal (CP) instruction is used for the compare. ZD fields are packed and then compared. This method has performance advantages. However, invalid PD data may cause a system 0C7 abend and program termination. Moreover, the integrity of ZD fields is only guaranteed when they contain valid ZD data. The CMP=CPD method cannot be used if any control field exceeds 16 bytes, for variable-length sorts when an even value (0, 2, 4, or 6) is specified for the VLTEST PARM, or for a Tape Sort.

When CMP=CLC is in effect, no data validation is performed and the integrity of the output is maintained, even if the sign for a PD or ZD field is invalid. This method is always used for control fields that exceed 16 bytes, for variable-length sorts when an even value is specified for the VLTEST PARM, and for a Tape Sort.

CENTWIN Parameter (Optional)

The CENTWIN run-time or installation option acts on 2-digit year data. CENTWIN generates a century window (for example, 1950 through 2049) that determines the century to which a 2-digit year belongs. At run-time, CENTWIN can be specified as either a PARM option or a SORT/MERGE control statement parameter. CENTWIN ensures that year data spanning centuries will be sequenced correctly. Without CENTWIN processing, an ascending sort would sequence the year 01 before the year 98. With CENTWIN processing, the 01 field could be recognized as a twenty-first century date (2001) and would thus be sequenced after 98 (1998).

For more information on specifying the CENTWIN option, see “CENTWIN” on page 5.7.

CENTWIN SORT/MERGE processing only applies to data defined as year data formats: Y2B, Y2C, Y2D, Y2P, Y2S, Y2Z, and the full-date formats (Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y). These data formats enable SyncSort to process 2-digit year fields as 4-digit years. A related data format, PD0, can be used to process the month and day portions of packed decimal date fields. To correctly specify date fields for CENTWIN SORT processing, you should be familiar with the CENTWIN-related data formats.

The following describes each of the year data formats and provides SORT control statement examples:

The Y2B Format

This format is used to sequence 2-digit, 1-byte binary year data with CENTWIN processing. The binary values are converted to decimal, and the two low order digits are used as year data. Thus, while binary and decimal values range from 00 to 255, year values range from 00 to 99. The relationship between binary, decimal and year values is shown in the following table:

Binary Value	Decimal Value	Year Value
X'00' to X'63'	00 to 99	00-99
X'64' to X'C7'	100 to 199	00-99
X'C8' to X'FF'	200 to 255	00-55

Table 24. Possible Values Representing Year Data with Y2B

The Y2C and Y2Z Formats

These formats represent 2-digit, 2-byte year data in either character (Y2C) or zoned decimal (Y2Z) format. Either Y2C and Y2Z formats can be used with data of the form

X'xyxy'

where *y* is a hexadecimal year digit 0-9 and *x* is hexadecimal 0 through F. Y2C and Y2Z ignore the *x* digits, leaving *yy*, the 2-digit unsigned year representation.

Suppose you have a character or zoned decimal date field *mmdyy* that begins at byte 20. You can use either Y2C or Y2Z to process the *yy* field. As the following example indicates, you could specify three sort keys to correctly sort this date:

```
SORT FIELDS=(24,2,Y2C,A, * Sorts yy field as 4-digit year
              20,2,CH,A, * Sorts mm field
              22,2,CH,A) * Sorts dd field
```

The *yy* field (24,2) will be processed according to the century window setting. For example, if CENTWIN=1945, the field *yy*=45 will be sequenced as if it were 1945, and *yy*=44 would be sequenced as if it were 2044. Thus, for an ascending sort, 44 would **follow** 45.

The Y2D Format

This format is used to sequence 2-digit, 1-byte packed decimal year data with CENTWIN processing. Use Y2D to extract the year data *yy* from packed decimal date fields. For example, consider a 3-byte packed decimal data field defined as

X'yydds'

This field has the year *yy* in the first byte and the day *ddd* in bytes 2 and 3. The packed decimal sign *s* would be in the last digit (half byte) of the third byte. To sort this date field, which begins at byte 20, with 4-digit year processing, use the following SORT control statement:

SORT

```
SORT FIELDS=(20,1,Y2D,A, * Sorts 2-digit year (yy) as 4-digit year
              21,2,PD,A) * Sorts ddds as 3 digits (ddd)
```

The Y2P Format

This format is used to sequence 2-digit, 2-byte packed decimal year data with CENTWIN processing. Use Y2P to extract the year data *yy* from packed decimal date fields spanning 2 bytes. For example, a packed decimal date of the form *yymmdd* would be stored as 4 bytes:

```
yymmdd = X'0yymmddC'
```

where the trailing C (sometimes F) is a positive sign and the leading 0 pads the field on the left to make an even number of digits.

Notice that the components of the date span bytes:

```
0y ym md dC
```

Y2P handles this condition by ignoring the first and last half bytes of the 2-byte field specification. Thus, Y2P processes *0yym* as *yy*, ignoring the leading digit (0) and the trailing digit *m* that is part of the month.

The following example uses Y2P to sort the year portion of the date field, which begins at byte 20:

```
SORT FIELDS=(20,2,Y2P,A) * Sorts yy field as 4-digit year
```

The field specification 20,2,Y2P treats X'0yym' as X'yy', and CENTWIN processing sorts *yy* as a 4-digit year *yyyy*.

The PD0 format, described below, can assist Y2P by processing month and day data that overlap year data in the original field.

The Y2S Format

This format is used to sequence 2-digit, 2-byte character or zoned decimal data. The Y2S format is identical to Y2C and Y2Z for valid numeric data, but Y2S treats data that begin with X'00', X'40', or X'FF' as non-year data. Thus, the Y2S format can distinguish records that have non-year data in the first byte of the year field, allowing such records to be sorted differently from other records.

Y2S treats non-year data as follows:

SORT

- Data with binary zeros (X'00') or a blank (X'40') in the first byte will not have century window processing applied to it. Instead, such data will be collated in sequence, **before** valid numeric year data for ascending order or **after** the year data for descending order.
- Data with all binary ones (X'FF') in the first byte will also not have century window processing applied to it. Instead, such data will be collated **after** valid year numeric data for ascending order or **before** the year data for descending order.
- Zones are ignored, as for Y2C and Y2Z, except for data where the first byte begins with X'00', X'40', or X'FF'.

As an example, suppose you want to preserve the input order of header and trailer records at the start or end of the file, and your header/trailer records are identified by binary zeros (X'00'), a blank (X'40'), or binary ones (X'FF') in the first byte of the date field.

The Y2S format allows CENTWIN to identify the header/trailer records and treat them differently from other records. Presuming the year data begin in column 20, you would use the following sort key specification:

```
SORT FIELDS=(20,2,Y2S,A) * Sorts yy field as 4-digit year
```

The yy field (20,2) will be processed according to the century window setting. For CENTWIN=1945, data with header and trailer records would be sorted as follows:

SORTIN Input	Record Order after Sorting
X'F9F6'	X'0000'
X'4001'	X'4000'
X'F4F4'	X'4001'
X'4000'	X'F5F1'
X'0000'	X'F9F6'
X'F5F1'	X'F4F4'
X'FF03'	X'FF03'

Note that if the above data were sorted as Y2C or Y2Z format, the output order would be different because the records starting with X'00', X'40', and X'FF' would be interpreted as numeric years. For example, suppose the fields in the above list were defined as Y2Z and sorted with EQUALS:

```
SORT FIELDS=(20,2,Y2Z,A),EQUALS
```

SORT

The data would be processed as follows:

SORTIN Input	Record Order after Sorting
X'F9F6'	X'F5F1'
X'4001'	X'F9F6'
X'F4F4'	X'FF03' (invalid numeric data)
X'4000'	X'4000' (invalid numeric data)
X'0000'	X'0000' (invalid numeric data)
X'F5F1'	X'4001' (invalid numeric data)
X'FF03'	X'F4F4'

The header and trailer records are sequenced as year data according to the CENTWIN setting (CENTWIN=1945), and they lose their position at the start and end of the file.

The PD0 Format

This format is used to sequence 2-8 byte packed decimal data. PD0 ignores the first digit and trailing sign during processing. PD0 is normally used in conjunction with the Y2P data format. The Y2P format is used to process the 2-digit year portion of a packed decimal date field, while the PD0 format is used to process the month and day portion of the field.

Although PD0 is typically used with Y2P, the PD0 format itself is not affected by CENTWIN processing.

Consider the packed decimal date field used in the example above:

```
yymmdd = X'0yymmddC'
```

where the trailing C (sometimes F) is a positive sign and the leading 0 pads the field on the left to make an even number of digits.

Notice that the components of the date span bytes:

```
0y ym md dC
```

The date can be processed as follows:

- Y2P processes the year component X'0yym' as X'yy'.
- PD0 processes the month and day components X'yymmddC' as X'mmdd'.

The following SORT control statement can be used to sort the entire date with CENTWIN processing:

```
SORT FIELDS=(20,2,Y2P,A, * Treats X'0yym' as X'yy'; sorts yy as yyyy  
21,3,PD0,A) * Treats X'yymmddC' as X'mmdd'
```

Full-Date Formats

Full-date formats can be used to sort or merge various date fields, processing dates ending or starting with year digits. They also process non-date data that are used with dates. For a full description of full-date formats, see the following section.

Using Full-Date Formats with CENTWIN

SyncSort's full-date data formats enable you to sort or merge a variety of date fields. The full-date formats are Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y. These date formats can process dates ending or starting with year digits:

- x...xyy (for example: qyy, mmyy, dddy, or mddy)
- yyx...x (for example: yyq, yymm, yyddd, or yymmdd)

The full-date formats also process non-date data commonly used with the dates. SyncSort interprets two-digit years (yy) according to the century window specified by the CENTWIN option. CENTWIN processing does not apply to non-date data.

In most cases, for CH, ZD, and PD date fields the full-date data formats are easier to use than the 2-digit date formats. The 2-digit formats can be more difficult because you must divide the date into its components. This requires care, particularly for PD dates, where date components (q, dd, mm, or yy) may span bytes or occupy only part of a byte. The full-date formats, on the other hand, process such dates automatically.

The table below describes the full-date formats. For date forms not in the table, use the 2-digit year formats or the non-year formats.

Note the following symbols used in the table:

- y** year digit (0-9)
- x** non-year digit (0-9)
- s** sign (hexadecimal A-F)
- 0** unused digit

SORT

Full-Date Format	Data Format	Date Form	Example Date Form	Length (bytes)
Y2T	CH, BI	yyx	yyq	3
		yyxx	yymm	4
		yyxxx	yyddd	5
		yyxxxx	yymmdd	6
Y2U	PD	yyx (X'yyxs')	yyq	2
		yyxxx (X'yyxxxs')	yyddd	3
Y2V	PD	yyxx (X'0yyxxs')	yymm	3
		yyxxxx (X'0yyxxxxs')	yymmdd	4
Y2W	CH, BI	xyy	qyy	3
		xxyy	mmyy	4
		xxxyy	ddydy	5
		xxxxyy	mmddy	6
Y2X	PD	xyy (X'xyys')	qyy	2
		xxxyy (X'xxxys')	ddydy	3
Y2Y	PD	xxyy (X'0xxys')	mmyy	3
		xxxxyy (X'0xxxxys')	mmddy	4

Table 25. Full-Date Formats

The table indicates the full-date formats that can be used with character (CH), binary (BI), or packed decimal (PD) data. Note the recognized non-date values:

Character or binary (Y2T and Y2W full-date formats)

C'0...0' (CH zeros)

C'9...9' (CH nines)

Z'0...0' (ZD zeros)

Z'9...9' (ZD nines)

X'00...00' (BI zeros)

X'40...40' (blanks)

X'FF...FF' (BI ones)

Packed (Y2U, Y2V, Y2X, and Y2Y full-date formats)

P'0...0' (PD zeros)

P'9...9' (PD nines)

The following two examples illustrate how you might use the *Full-Date Formats* table:

- Suppose you have a packed decimal (PD) date field of the form mmyy. To sort this field correctly, you would use the Y2Y 3-byte format from the table. Thus, if the field starts in position 30, you would specify the following SORT control statement to sort in descending order:

```
SORT FIELDS=(30,3,Y2Y,D)
```

Any PD fields of all PD zeros or all PD nines will be processed automatically as non-date data.

- Suppose you have a character (CH) date field of the form yymmdd. To sort this field correctly, you would use the Y2T 6-byte format from the table. Thus, if the field starts in byte 40, you would specify the following SORT control statement to sort in ascending order:

```
SORT FIELDS=(40,6,Y2T,A)
```

Any CH zeros, CH nines, BI zeros, blanks, and BI ones will be processed automatically as non-date data.

Collating Sequence with Full-Date Formats

For full-date formats, the yy component is always sorted first (treated as primary key). This is so even when the yy is physically at the rightmost end of the field, as for Y2W, Y2X, and Y2Y. For example, a 6-byte Y2W field has the form xxxxyy. This is collated with the yy as the primary key and xxxx as the secondary key. Because SyncSort automatically collates the year character first, you don't have to deal with yy manually, for example by using PD0 and Y2D.

SORT

It is important to understand that the xxxx component of a full-date format must be designed to collate as a unit. Suppose you have the 6-byte Y2T field yyxxxx. If you collate this field in ascending order, then yy collates first (the primary key) with xxxx collating second (secondary key). Consider two possibilities:

- If yyxxxx is actually yymmdd, you will be sorting first by year, then month, then day.
- If yyxxxx is actually yyddmm, you will sorting by year, then day, then month. In most cases, sorting in this way would not be what you intended.

To correctly collate a date, the date components must be in an order suitable for collating. For example, mmddyy and yymmdd will collate correctly, but ddmmyy or yyddmm will not. For date forms that will not collate correctly, you must use one of the 2-digit year formats (Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z).

The following table shows the order for ascending collation when using full-date formats with the CENTWIN option:

Full-Date Format	Date Format	Ascending Sort Sequence
Y2T Y2W	CH, BI	BI zeros Blanks CH/ZD zeros Lower century dates (e.g. 1980) Higher century dates (e.g. 2010) CH/ZD nines BI ones
Y2U Y2V Y2X Y2Y	PD	PD zeros Lower century dates (e.g. 1980) Higher century dates (e.g. 2010) PD nines

For a descending sort, the collation order is reversed.

Other date formats (non-full-date), with the exception of Y2S, do not process non-date data; their sort sequence for ascending sorts is simply lower century dates than higher century dates.

Examples Using Full-Date Formats

Example 1 (Y2W)

The following SORT control statement sorts a C'mmddy' date field in ascending order, with the previously set fixed century window 1984-2083:

```
SORT FIELDS=(10,6,Y2W,A)      * Sort C'mmddy' in ascending order
                                * with Y2W
                                * and previously set century window 1984-2083
```

The *Full-Date Formats* table above indicates that the 6-byte Y2W form is appropriate for a CH input field of the form xxxxyy. As shown in the following table, the output will be collated as C'yyyymmdd', with the non-date data (zeros) appearing correctly at the beginning of the sorted output.

SORTIN Input mmddy	Record Order after Sorting mmddy	Actual Date after Sorting yyyy/mm/dd
021783	000000	non-date data
092206	070484	1984/07/04
081395	081395	1995/08/13
110210	092206	2006/09/22
000000	110210	2010/11/02
070484	043060	2060/04/30
043060	021783	2083/02/17

Example 2 (Y2T)

The following SORT control statement sorts a Z'yyddd' date field in descending order, with the previously set fixed century window 1921-2020:

```
SORT FIELDS=(20,5,Y2T,D)      * Sort Z'yyddd' in descending order
                                * with Y2T
                                * and previously set century window 1921-2020
```

The *Full-Date Formats* table above indicates that the 5-byte Y2T form is appropriate for a ZD input field of the form yyddd. As shown in the following table, the output will be collated as Z'yyyddd', with the non-date data (nines and zeros) appearing correctly at the beginning and end of the sorted output.

SORT

SORTIN	Record Order	Actual Date
Input	after Sorting	after Sorting
yyddd	yyddd	yyyy/ddd
00000	99999	non-date data
50237	20153	2020/153
99999	20047	2020/047
20047	01223	2001/223
94001	94001	1994/001
01223	50237	1950/237
20153	21148	1921/148
21148	00000	non-date data

Example 3 (Y2Y)

The following SORT control statement sorts a P'mmddy' (X'0mmddy') date field in ascending order, with the previously set fixed century window 1921-2020:

```
SORT FIELDS=(26,4,Y2Y,A)      * Sort P'mmddy' in ascending order
                                * with Y2Y
                                * and previously set century window 1921-2020
```

The *Full-Date Formats* table above indicates that the 4-byte Y2Y form is appropriate for a PD input field of the form xxxxyy. As shown in the following table, the output will be collated as P'yyyymmdd', with the non-date data (zeros and nines) appearing correctly at the beginning of the sorted output. Note that the first two columns are in hexadecimal.

SORTIN	Record Order	Actual Date
Input	after Sorting	after Sorting
mmddy	mmddy	yyyy/mm/dd
0999999C	0000000C	non-date data
0102250C	0080321C	1921/08/03
0032120C	0102250C	1950/10/22
0010194C	0010194C	1994/01/01
0000000C	0111501C	2001/11/15
0111501C	0032120C	2020/03/21
0080321C	0999999C	non-date data

FIELDS=COPY (Required for a Copy)

Use FIELDS=COPY to copy one or more input files. Multiple files can be copied if they are concatenated to the SORTIN DD statement. Other control statements such as INREC, INCLUDE/OMIT, OUTREC, and OUTFIL may be specified in conjunction with a copy application, allowing you to edit and reformat the file(s) *without* sorting them.

The SUM control statement and an E32 exit cannot be specified with FIELDS=COPY. All Phase 3 exits can be used.

CKPT/CHKPT Parameter (Optional)

The CKPT/CHKPT parameter instructs SyncSort to take a checkpoint at every end-of-volume of a SORTOUT data set when OUTFIL is not used and also at the beginning of Phase 3 before the SORTOUT data set is opened. Either spelling of this parameter is accepted.

This parameter requires a SORTCKPT DD statement. It cannot be specified in conjunction with a user-issued STIMER macro or an incore sort. Checkpoints cannot be taken within a user exit routine.

Refer to “Chapter 13. Performance Considerations” for an explanation of the Checkpoint/Restart feature.

DYNALLOC Parameter (Optional)

The format of the DYNALLOC parameter is illustrated below.

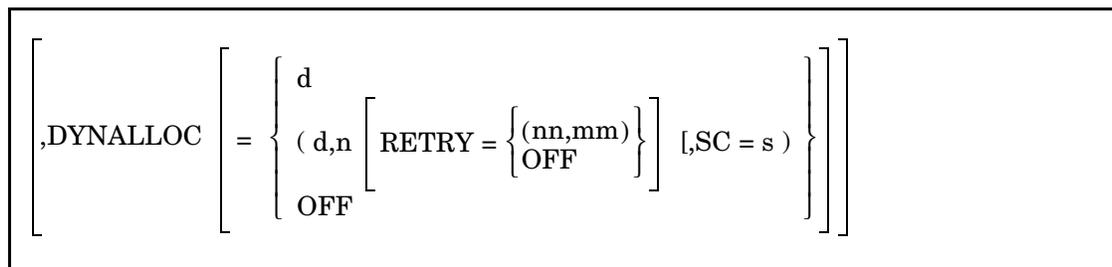


Figure 66. DYNALLOC Parameter Format

DYNALLOC requests the dynamic allocation of SORTWK data sets on device type *d*. Specify the device type either as a decimal number (e.g., 3390) or by the system generic name (e.g., SYSDA). Any disk device accepted for a SORTWK DD statement can be specified. Note that if VIO is specified it will be ignored, and the installation default for the DYNALLOC device type will be used in its place.

Note that the DYNALLOC parameter may be used alone, without any subparameters. In this case, the DYNALLOC installation default settings are used.

For MAXSORT applications, *n* is the number of SORTWK data sets that will be allocated. As many as 32 SORTWK data sets can be specified. The default for *n* is 3.

For non-MAXSORT applications, *n* can be 1 through 255. This value specifies the number of SORTWK data sets that can potentially be allocated. For values of *n* that are 31 or less, SyncSort can automatically raise the number to 32 if the application requires it. When *n* is 33 through 255, this value specifies the maximum number of SORTWK data sets that can be allocated.

DYNALLOC=OFF can be specified to override a DYNALLOC=ON installation default.

SORT

Normally for both MAXSORT and non-MAXSORT applications, any SORTWK data sets provided in the JCL will contribute towards the value of *n*. For instance, if *n* was set to 40 in a non-MAXSORT application and 30 SORTWKs were provided in the JCL, DYNALLOC could obtain 10 additional SORTWKs if needed. Note that there is an installation option to disable DYNALLOC if SORTWKxx DD statements are present.

SyncSort uses the value specified in the **RETRY** parameter to request automatic DYNALLOC retry. This facility attempts to avoid a sortwork capacity exceeded condition when disk space is not immediately available to satisfy a DYNALLOC request. SyncSort will automatically retry a specified number of times and wait a prescribed interval between DYNALLOC requests.

The *nn* in the first position designates the number of times SyncSort will retry a failed DYNALLOC request. The minimum allowed is 0 and the maximum is 16. The *mm* in the second position designates the number of minutes SyncSort waits between each DYNALLOC request. The minimum allowed is 0 and the maximum is 15. A value of 0 can be used to request an immediate retry. **RETRY=OFF** or an *nn* of 0 can be specified to override a **RETRY=ON** installation default.

In an environment where DFSMS manages temporary work data sets, the **SC** subparameter specifies a storage class *s* for SyncSort to use when dynamically allocating SORTWORK data sets. The storage administrator at your installation defines the names of the storage classes you can specify. Note that an installation written automatic class selection (ACS) routine can override the storage class you specify. If SMS is not installed or active to manage temporary work data sets, the *d* device specification will be used in the SORTWORK dynalloc request.

EQUALS/NOEQUALS Parameter (Optional)

The **EQUALS** parameter insures that the original order of equal-keyed records is preserved. These records will be in the same order in the output file as they were in the input file. **NOEQUALS**, the default, specifies that equal-keyed records may not be written in their original input order.

When the **EQUALS** parameter is used with the **SUM** control statement, the first of the equal-keyed records is retained with the sum; all other records are deleted after the specified field(s) have been summed.

EQUALS/NOEQUALS can also be specified as a **PARM** option on the **EXEC** statement. If this option is specified both on the **SORT** control statement and as a **PARM** option, the **SORT** specification takes precedence.

Performance is usually improved when **NOEQUALS** is in effect.

FILSZ Parameter (Optional)

The FILSZ parameter specifies the actual (FILSZ=*n*) or estimated (FILSZ=*En*) decimal number of records to be sorted. This number should reflect any changes produced by INCLUDE/OMIT, E14 and/or E15, SKIPREC and STOPAFT processing.

If FILSZ=*n* is specified, SyncSort will terminate unless exactly *n* records are processed.

If FILSZ is specified for a Tape Sort, use only the *En* specification. This value should indicate the number of records in the input file *without* taking into account records added or deleted by an E14 or E15 exit.

FILSZ can also be specified as a PARM option on the EXEC statement. If this option is specified both on the SORT control statement and as a PARM option, the PARM specification takes precedence.

SIZE Parameter (Optional)

The SIZE parameter specifies the actual (SIZE=*n*) or estimated (SIZE=*En*) decimal number of records read from the input file. Unlike the FILSZ parameter, this number should *not* reflect any changes produced by INCLUDE/OMIT or exit processing, but should reflect SKIPREC and STOPAFT processing.

If the FILSZ parameter is not specified and SIZE=*n* is specified, SyncSort will terminate unless exactly *n* records are processed. If the FILSZ parameter is specified, the SIZE value is considered an estimate whether or not it is preceded by an E.

SKIPREC Parameter (Optional)

The SKIPREC=*n* parameter instructs SyncSort to skip a decimal number of records before the input file is sorted or copied. The *n* records skipped are deleted from the input file before E15 and INCLUDE/OMIT processing, if specified, take place.

If SKIPREC is specified as a PARM option as well as on the SORT control statement, the PARM specification takes precedence.

STOPAFT Parameter (Optional)

The STOPAFT=*n* parameter specifies the number of records to be sorted or copied. These will be the first *n* records after E15, INCLUDE/OMIT and SKIPREC processing, if specified, have completed.

If STOPAFT is specified as a PARM option as well as on the SORT control statement, the PARM specification takes precedence.

STOPAFT cannot be specified for a Tape Sort.

SORT

Sample SORT Control Statements

```
SORT FIELDS=(2,3,2,BI,D,8,2,4,BI,A,25,10,CH,A,15,10,LS,D)
```

Figure 67. Sample SORT Control Statement

This sample SORT control statement indicates four control fields:

- The first, or primary, field begins in bit 4 of byte 2, is 2 bytes long, is in binary format and is to be sorted in descending order.
- The second control field begins in byte 8, is 2 bytes 4 bits long, is a binary format and is to be sorted in ascending order.
- The third control field begins on byte 25, is 10 bytes long, is in character format and is to be sorted in ascending order.
- The fourth control field begins on byte 15, is 10 bytes long, is an EBCDIC numeric field with a leading separate sign and is to be sorted in descending order.

```
SORT FIELDS=(20,5,A,5,10,D,30,5,A),FORMAT=CH,CKPT
```

Figure 68. Sample SORT Control Statement

This sample SORT control statement specifies the following:

- There are three control fields. Because all three fields have the same data format (in this case, character), the FORMAT=CH subparameter is specified so that the CH value does not have to be specified for each of the fields.
- The first control field begins on byte 20, is 5 bytes long and is to be sorted in ascending order.
- The second control field begins on byte 5, is 10 bytes long and is to be sorted in descending order.
- The third control field begins on byte 30, is 5 bytes long and is to be sorted in ascending order.
- SyncSort will take a checkpoint.

SUM Control Statement

The SUM control statement deletes records with equal control fields and optionally summarizes specified numeric fields on those records. Equal keyed records are processed pair by pair. If numeric fields are to be summarized, the data in the summary fields are added, the sum is placed in one of the records, and the other record is deleted. Provided arithmetic overflow does not occur, the SUM control statement produces only *one* record per sort key in the output data set. The records deleted by sum can optionally be written to a separate data set.

The SUM control statement cannot be used when FIELDS=COPY is specified on the SORT or MERGE control statement or for a Tape Sort.

SUM Control Statement Format

The format of the SUM control statement is illustrated below.

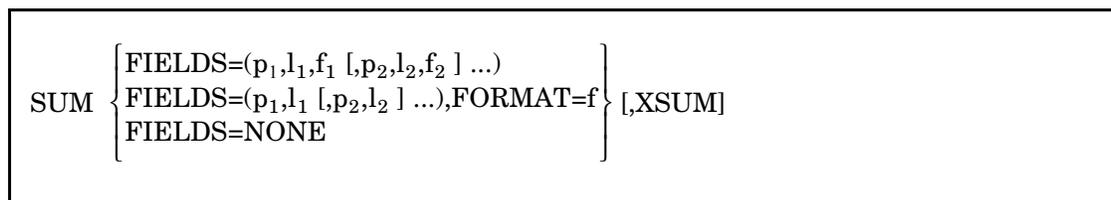


Figure 69. SUM Control Statement Format

FIELDS Parameter (Required)

The FIELDS parameter defines the numeric fields to be summed when the control fields of two or more records are equal. Specify FIELDS=NONE to reduce the sorted data to one record per sort key without summarizing any numeric fields.

Each field specified in the FIELDS parameter is identified by its position p , length l and format f .

- p** The position value indicates the first byte of the field relative to the beginning of the input record *after* INREC and/or E15 processing, if specified, have completed. The field must begin on a byte boundary.
- l** The length value indicates the length of the field. The length must be an integer number of bytes. Refer to the chart below for the permissible lengths.
- f** The format value indicates the data format. Fields with BI, FI, FL, PD and ZD formats can be summarized. If all the summary fields have the same format, you can specify the format value once by using the FORMAT=f subparameter. If both the individual f values and the FORMAT subparameter

SUM

are specified, the individual f values will be used. (Note that f values must be specified for each compare field.)

FORMAT CODE	PERMISSIBLE LENGTH
BI	2, 4, or 8 bytes
FI	2, 4, or 8 bytes
FL	4, 8, or 16 bytes
PD	1 to 16 bytes
ZD	1 to 18 bytes

Table 26. Permissible Lengths for SUM Fields

XSUM Parameter (Optional)

Specify the XSUM parameter if you want records deleted by SUM processing to be written to a data set defined by the SORTXSUM DD statement. These records will be written to SORTXSUM at the time of SUM processing. The records will not undergo OUTREC, E35, and OUTFIL processing because such processing occurs after SUM processing.

The DCB BLKSIZE of the SORTIN data set will not be used to determine the BLKSIZE of the SORTXSUM data set. System determined blocksize will be used when enabled and appropriate. Unblocked output will be generated if system determined blocksize has been disabled and an explicitly specified blocksize has not been provided in the JCL.

The XSUM file will be sequenced in the same order as the SORTOUT file.

Note that XSUM may increase system requirements:

- Adding XSUM to an existing sort application may result in an increase in the amount of SORTWORK space required. This occurs because XSUM delays all summing until Phase 3.
- Adding XSUM to an existing MAXSORT application could cause the generation of additional intermediate output files (SORTOU00 or SORTOU n). This occurs because XSUM delays SUM processing until the final MAXSORT merge pass.
- XSUM may require additional main memory. Specify a region size of 512K or more.

General Considerations for SUM

- If NOEQUALS is in effect, the record which is retained is determined arbitrarily. If EQUALS is in effect, the record which is retained is the first record read. In a SORT application, in a MERGE, the retained record will be from the lowest-numbered input file. The EQUALS parameter can be specified on the SORT or MERGE control statement or as a PARM option.
- A sort or merge control field cannot be summarized. A portion of a control field cannot be included in a sum field.
- Sum fields may not overlap each other.
- Non-sum fields remain unchanged and are retained from the record which contains the sum.
- If arithmetic overflow or underflow occurs during the summing of two records, those records are not summarized and neither record is deleted. Further processing is determined by the option selected at installation through the SUMOVFL parameter or the run time parameter OVFL0. If the RC16 option of this parameter has been selected, processing will terminate with a WER049A critical error. For the RC0 (the delivered default) or the RC4 option, sum processing will continue and a WER049I message will be issued (only for the first occurrence). If a subsequent pair of records with equal control fields can be summarized without causing overflow or underflow, they will be summarized. To avoid arithmetic overflow, use the INREC control statement to insert zeros of the proper format immediately before the sum field. For example, for a PD field, use nZ to insert binary zeros.
- Remember that the first 4 bytes of variable-length records are reserved for the Record Descriptor Word, so the first byte of the *data* portion of the record is byte 5.
- SUM is incompatible with an incore sort. If you specify the SUM control statement, allocate SORTWKxx data sets in the JCL or use the DYNALLOC feature for dynamic SORTWK allocation. If no JCL SORTWKs are provided and DYNALLOC is disabled by default, SUM will cause DYNALLOC to be enabled.
- When FL fields are summarized, user-issued SPIE macros are not permitted and exit routines must not produce exponent overflow or underflow. Because of the numeric rounding performed by the hardware, the exact sum depends on the order in which fields are summed. Thus, the sum may vary slightly for different executions.
- By default, the sign byte of a positive summarized ZD field will be converted to printable format. If you want to disable this action, use the NZDPRINT PARM option. Refer to “ZDPRINT” on page 5.34.

SUM

Sample SUM Control Statements

The following SUM control statement eliminates equal-keyed records without summarizing numeric fields. The XSUM option causes the eliminated records to be written to a data set defined on the SORTXSUM DD statement.

```
SUM FIELDS=NONE, XSUM
```

Figure 70. Sample SUM Control Statement

Records with equal control fields will be eliminated from SORTOUT or SORTOFnn data sets so that only one record is retained.

The following SUM control statement summarizes two numeric fields on records with equal control fields.

```
SUM FIELDS=(20,4,32,4), FORMAT=PD
```

Figure 71. Sample SUM Control Statement

When the control fields are equal, this SUM control statement summarizes the numeric data in the fields beginning in bytes 20 and 32. Because both fields are in packed decimal format, the FORMAT=PD subparameter is used so that the PD value does not have to be specified for each field.

Comprehensive examples illustrating the SUM control statement are provided in “Chapter 3. How to Use SyncSort’s Data Utility Features”.

Chapter 3. How to Use SyncSort's Data Utility Features

Introduction

This chapter assumes that you already know how to sort records and are ready to use SyncSort's Data Utility features for any or all of the following:

- Selecting only those input records and data fields that are needed for an application.
- Eliminating duplicate records.
- Consolidating records into a single record that contains the sum of any numeric data fields.
- Making output data printable and easy to read.
- Writing a multi-sectioned report complete with headers and trailers.
- Generating several output files and reports with a single pass of the sort.

The following examples show how you can accomplish these tasks with SyncSort. Each example is self-contained and provides coding instructions for both the required JCL and the necessary control statements. Use them as starting points for your own applications. For details of control statement syntax see "Chapter 2. SyncSort Control Statements."

Sample Data Utility Applications

The following chart lists applications that demonstrate SyncSort's features.

Feature	Application	Page
Selecting Input Records	Including Relevant Records	3.3
	Omitting Irrelevant Records	3.5
Selecting Relevant Fields from the Input Records	Selecting a Number of Fields from Longer Records	3.7
	Eliminating Irrelevant Data Field(s)	3.8
	Selecting Fields from Variable-Length Records	3.9
Combining Records within a File	Combining Records and Summing Numeric Data Fields	3.11
	Eliminating Duplicate Records	3.12
Making Output Records Printable and Easy to Read	Reordering the Positions of Record Fields	3.14
	Inserting Blanks and Repositioning Record Fields	3.16
	Inserting Binary Zeros	3.18
	Converting Unprintable Data to Readable Form	3.20
	Converting Unprintable Data to Hexadecimal Format	3.22
	Converting and Editing Unprintable Data	3.23
	Putting a Data Field in Standard Format	3.25
	Converting from Variable to Fixed-Length Format	3.27
Printing Input Records on Multiple Output Lines	3.28	
Dividing a Report into Sections	Dividing Output into Sections	3.30
Writing Headers and Trailers for a Report	Writing a Title Page for a Report	3.32
	Writing a Page Header	3.34
	Writing a Section Header	3.35
	Using a Header to Eliminate Duplication Information within a Section	3.37
	Writing a Report Trailer or Summary	3.39
	Writing a Page Trailer	3.40
Totaling and Subtotaling Data	Totaling Data at the End of a Report	3.41
	Subtotaling Data at the End of a Page	3.43
	Totaling Data at the End of a Section	3.44
Obtaining Maximum, Minimum and Average Data	Printing Maximum, Minimum and Average Data in Section Trailers	3.47
Counting Data Records	Obtaining a Count of Data Records	3.49
	Obtaining a Cumulative (Running) Count of Data Records	3.50
Creating Multiple Output Files	Generating Several Output Files with Different Information	3.53
	Writing Identical Output Files to Different Devices	3.55

Selecting Input Records

When only certain records from an input file are needed for an application, SyncSort allows you to set up one or more logical conditions for including only those records. Alternatively, you may specify conditions for omitting records from an application. Each condition is

based on a comparison between two record fields or between a record field and a constant. You may specify the constant as a positive or negative decimal, a hexadecimal or binary constant, or a character literal. Multiple conditions may be specified, provided you connect them with ANDs and ORs.

To specify the conditions for selecting records, use the INCLUDE/OMIT control statement. For complete syntax, and examples of bit level criteria in record selection, see “INCLUDE/OMIT Control Statement” on page 2.16

When processing variable-length records, by default all fields specified must be contained within the record. If an application is expected to reference fields not completely contained within the record, refer to “VLTESTI” on page 5.33. VLTESTI provides for processing of records that do not contain all fields.

Including Relevant Records

Example: A school board requires a list of all students performing below their grade level on standardized exams. (The record layout is given in Figure 72 and a sample record is given in Figure 73.)

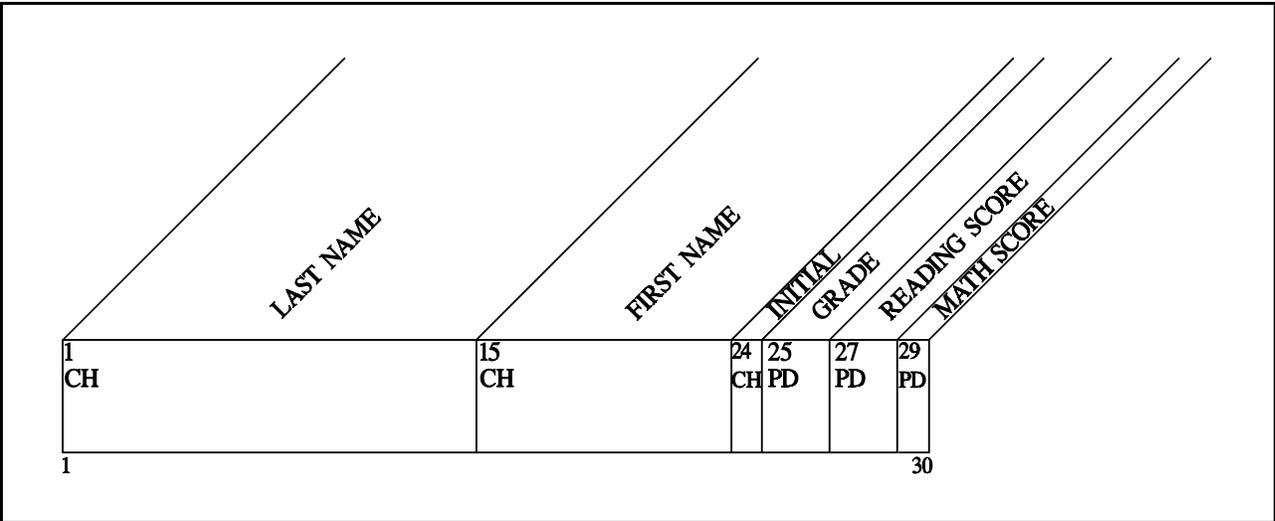


Figure 72. Input Record Layout

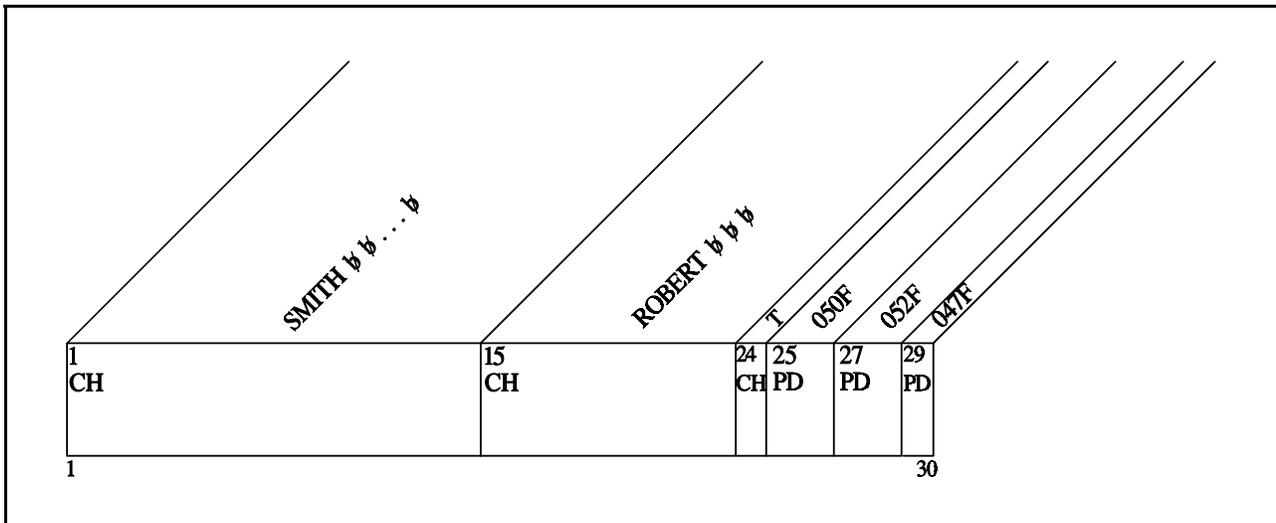


Figure 73. Sample Student Record

To generate the list, the following is coded:

```

//SUBLEV   JOB                               Gives the Jobname
//          EXEC PGM=SYNCSORT                 Identifies the Program
//SYSOUT   DD   SYSOUT=*                       Assigns SyncSort
//*                                               Messages to I/O Device
//SORTIN   DD   DSN=WWBRSM.STUDENTS,DISP=SHR   Defines Input Data Set
//SORTOUT  DD   SYSOUT=*                       Defines Output Data Set
//SORTWK01 DD   UNIT=SYSDA,                   Defines Intermediate
// SPACE=(CYL,15)                               Storage
//SYSIN    DD   *
          INCLUDE COND=(29,2,LT,25,2,OR,27,2,LT,25,2),
          FORMAT=PD                             Selects Records
          SORT   FIELDS=(1,14,CH,A)             Sorts Records

```

Figure 74. JCL and Required Control Statements

Explanation: In this application, two comparisons are necessary to identify the records needed for the list: the Grade field (25,2) has to be compared to the student's Reading Score field (27,2) and to the Mathematics Score field (29,2). All numeric fields on the student records are in packed-decimal (PD) format.

The two-clause INCLUDE statement (see Figure 74) guarantees the selection of the needed records from the file. The first clause (29,2,LT,25,2) guarantees that records with Math Scores less than the Grade field are INCLUDED. The second clause (27,2,LT,25,2) guarantees that records with Reading Scores less than the Grade field are also INCLUDED. The OR connecting the two clauses guarantees that if either or both of the scores are less than the Grade field, the record is selected. Finally, since all the fields are in packed-decimal format (PD), FORMAT=PD is specified.

The sample record shown above will be INCLUDED because the student's Math Score (047F) is lower than the Grade level (050F).

Omitting Irrelevant Records

Example: Records that have an Invoice Status Code of F (fully paid) are to be omitted in preparing a list of only those customers with outstanding payments. (The input record layout is given in Figure 75 and a sample input record is given in Figure 76.)

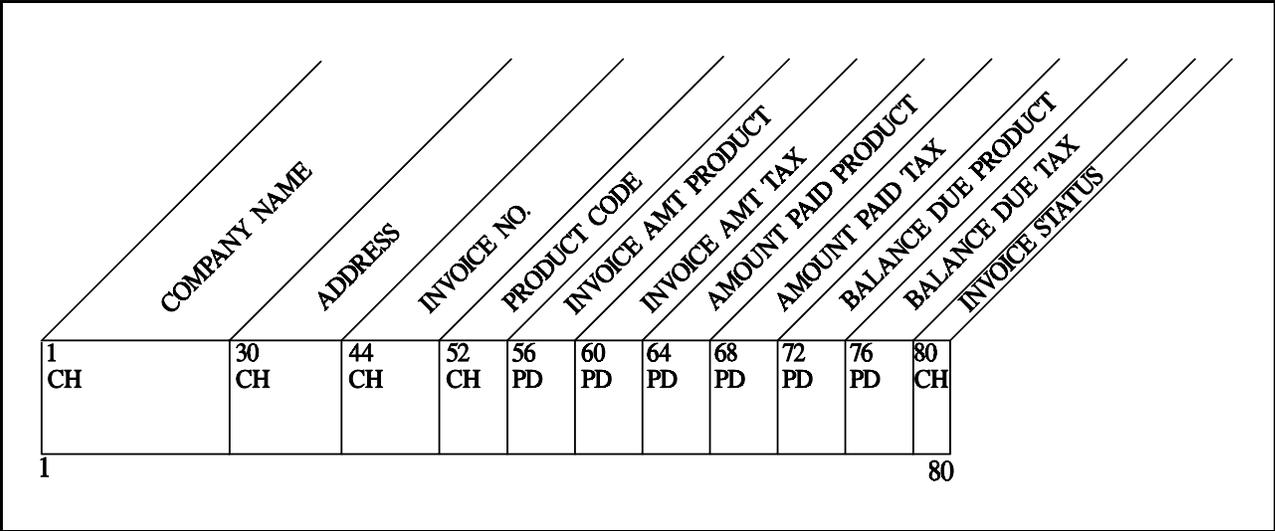


Figure 75. Input Record Layout

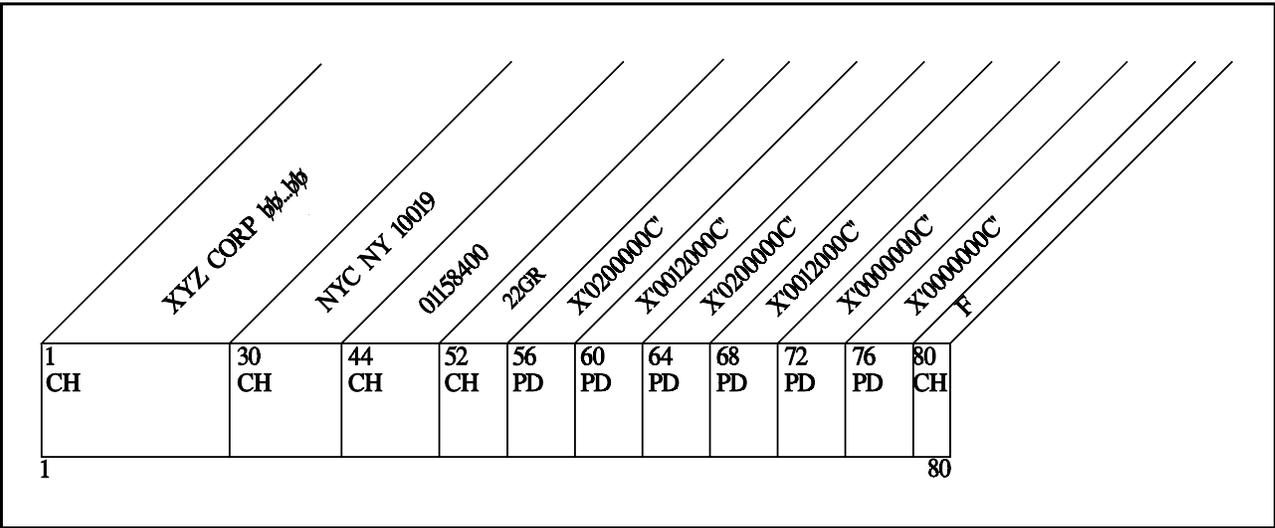


Figure 76. Sample Input Record

To produce this list of customers selected from the masterfile, the following is coded.

//OUTPAY	JOB		Gives the Jobname
//	EXEC	PGM=SYNCSORT	Identifies the Program
//SYSOUT	DD	SYSOUT=*	Assigns SyncSort Messages to
//*			I/O Device
//SORTIN	DD	DSN=NEWINV,DISP=SHR	Defines Input Data Set
//SORTOUT	DD	SYSOUT=*	Defines Output Data Set
//SORTWK01	DD	UNIT=SYSDA,SPACE=(CYL,20)	Defines Intermediate Storage
//SYSIN	DD	*	
	OMIT	COND=(80,1,CH,EQ,C'F')	Omits Records
	SORT	FIELDS=(1,29,CH,A)	Sorts Records

Figure 77. JCL and Required Control Statements

Explanation: In this application, a simple comparison is necessary to identify those master-file records that are not needed: the Invoice Status Code field (80,1,CH) has to be compared to the constant 'F'.

The OMIT statement's condition, 80,1,CH,EQ,C'F', (see Figure 77) guarantees that invoice records, like the sample record shown above, with the Invoice Status Code 'F' are omitted from the sort.

Selecting Relevant Fields from the Input Records

Input records often contain some information that is not relevant to a specific application. For example, records in a personnel masterfile might, in addition to addresses, include salaries and other confidential information that is not required for preparing a mailing list.

SyncSort's Data Utility features allow you to select only those record fields that contain necessary data and to eliminate those that do not. More important, SyncSort enables you to do this editing *before* the records are sorted. As a result, the sort has fewer bytes to handle and processing is more efficient.

For complete syntax of the INREC control statement, see "INREC Control Statement" on page 2.35.

INREC FIELDS=(p ₁ ,l ₁ [,p ₂ ,l ₂ ,...,p _n ,l _n])
--

Figure 78. Basic INREC Statement Format

p,l Specify the beginning position and length in bytes of the input record's relevant fields. When specifying contiguous fields, or fields that directly follow one another, you can simply indicate the starting position of the first field together with the combined length of the fields that are contiguous.

Selecting a Number of Fields from Longer Records

Example: A school wants to rank the entire student body by grade point index. This application simply requires selecting the two relevant fields out of all the fields in the student records and, then, sorting on the Grade Point Index field. (The Input Record layout is given in Figure 79.)

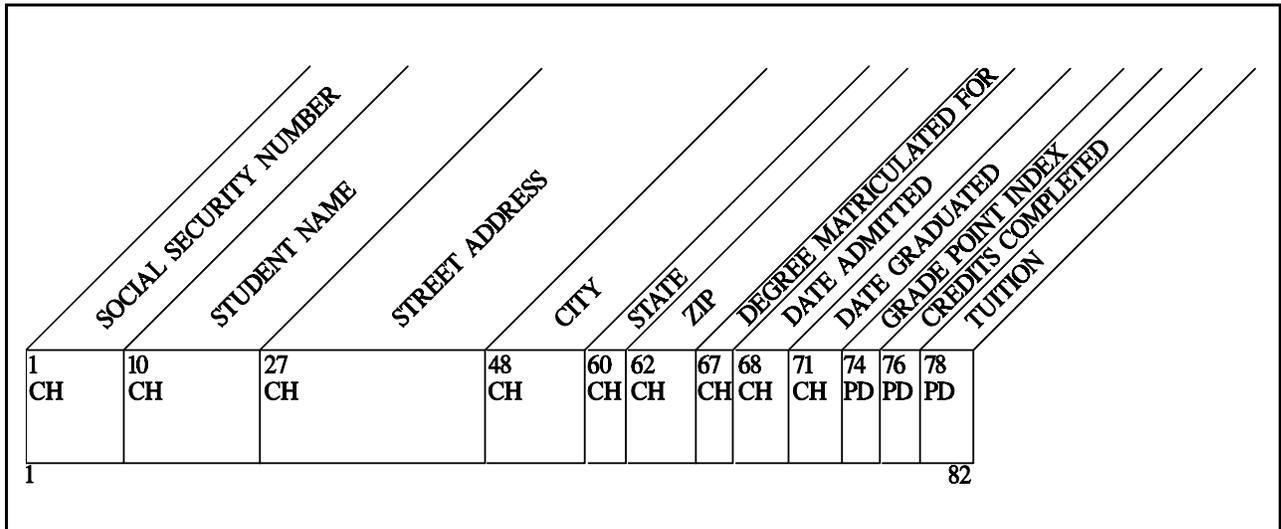


Figure 79. Input Record Layout

To include only the relevant fields and generate the ranked list of students, the following is coded:

```

//RANK      JOB                               Gives the Jobname
//          EXEC  PGM=SYNCSORT                Identifies the Program
//SYSOUT    DD    SYSOUT=*                    Assigns SyncSort Messages
//*                                                to I/O Device
//SORTIN    DD    DSN=TOT.STUDENTS,DISP=SHR   Defines Input Data Set
//SORTOUT   DD    SYSOUT=*                    Defines Output Data Set
//SORTWK01  DD    SPACE=(CYL,10),UNIT=SYSDA   Defines Intermediate Storage
//SYSIN     DD    *
           INREC  FIELDS=(1,9,                Selects Record Fields
                        74,2)
           SORT   FIELDS=(10,2,PD,D)          Sorts Records

```

Figure 80. JCL and Required Control Statements

Figure 81 shows the input record after INREC processing.

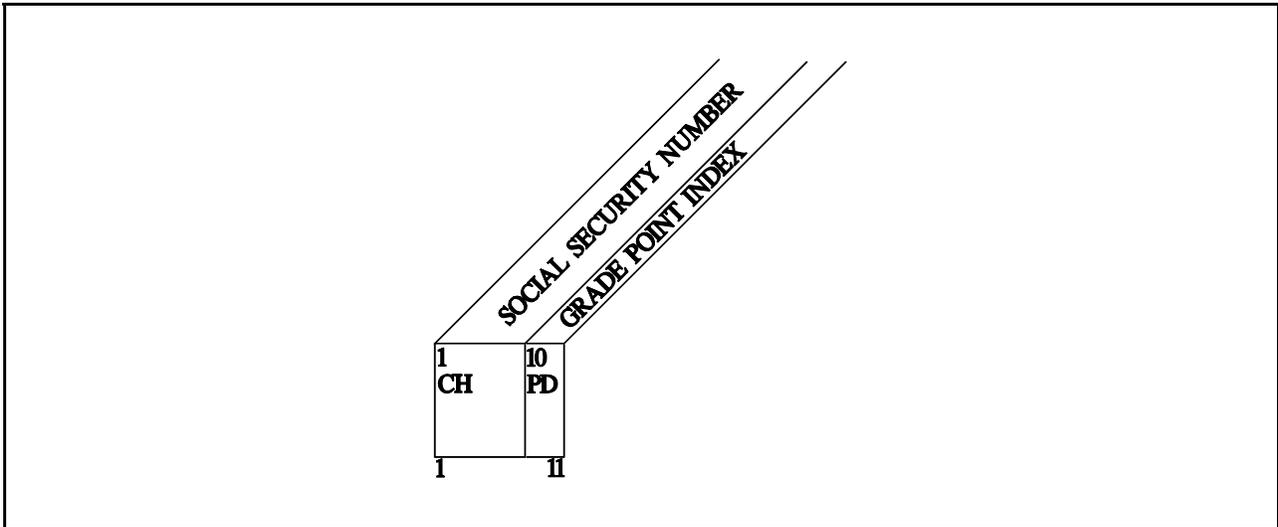


Figure 81. Form of Post-INREC Record

Explanation: Specifying the two relevant data fields--the Social Security Number (1,9) and the Grade Point Index (74,2)--on the INREC statement provides the sort with necessary data for the application and eliminates the fields that are not relevant to the application. INREC processing thus shortens each record to just a little under 14% of its original size.

Eliminating Irrelevant Data Field(s)

Example: For an inventory list, the price code on the masterfile records is not necessary. (The masterfile record layout is given in Figure 82.)

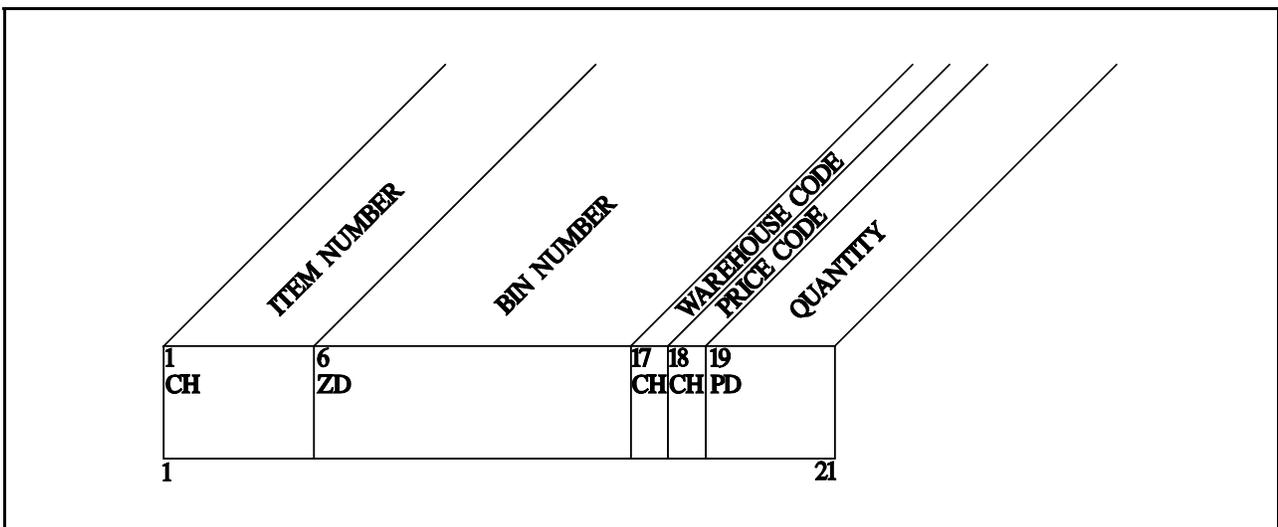


Figure 82. INPUT Record Layout

To eliminate the Price Code field and generate the inventory list, the following is coded.

```

//INVENTR JOB                               Gives the Jobname
//          EXEC PGM=SYNCSORT                Identifies the Program
//SYSOUT DD SYSOUT=*                        Assigns SyncSort Messages
//*                                           to I/O Device
//SORTIN DD DSN=INV.WARHOUS,DISP=SHR        Defines Input Data Set
//SORTOUT DD SYSOUT=*                       Defines Output Data Set
//SORTWK01 DD SPACE=(CYL,15),UNIT=SYSDA    Defines Intermediate Storage
//SYSIN DD *
      INREC FIELDS=(1,17,                    Selects Record Fields
                  19,3)
      SORT FIELDS=(1,5,CH,A)                Sorts Records
      .
      .
      .

```

Figure 83. JCL and Required Control Statements

Figure 84 shows the input record after INREC processing.

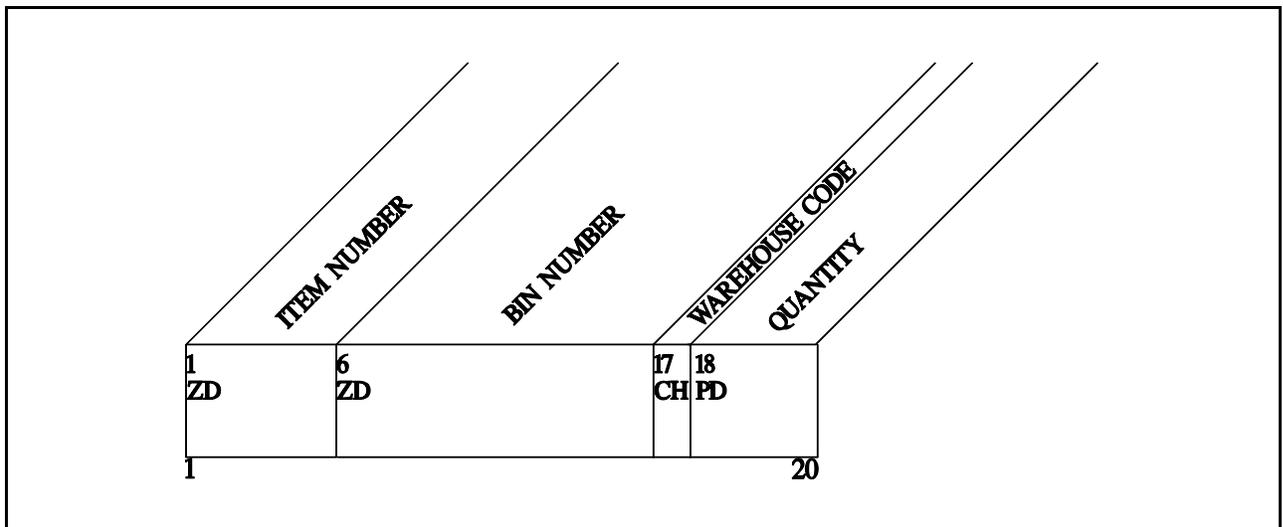


Figure 84. Post-INREC Record Layout

Explanation: Specifying only those fields that are necessary eliminates those that are not necessary for the application. The Price Code field (18,1) has *not* been specified on the INREC statement; it will be deleted from the input records before the records are sorted by item number for the list.

Selecting Fields from Variable-Length Records

Example: For each volume in its collection, a library requires the catalog number and any information concerning translations, other volumes in a series, additional copies on file, and so on. The catalog file consists of variable-length records, and except for the catalog

number, the required information is contained in the variable-length portion of each record. (The record layout is given in Figure 85.)

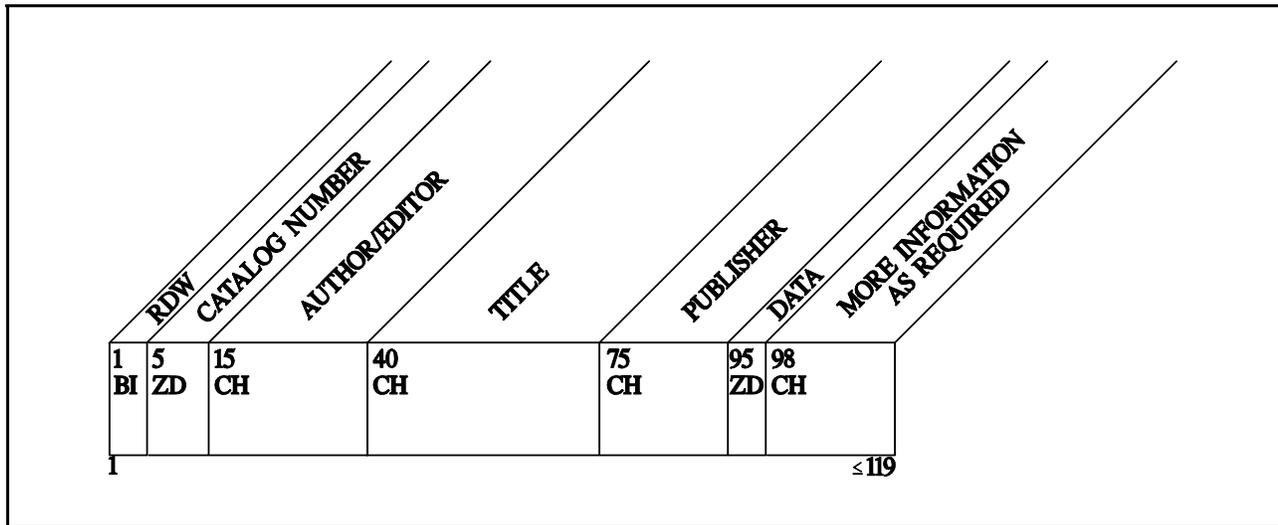


Figure 85. Sample Record Layout

To include only the relevant fields on the input records and to generate this list, the following is coded.

```

//LISTCAT JOB Gives the Jobname
// EXEC PGM=SYNCSORT Identifies the Program
//SYSOUT DD SYSOUT=* Assigns SyncSort Messages
//* to I/O Device
//SORTIN DD DSN=LIB.CATALOG,DISP=SHR Defines Input Data Set
//SORTOUT DD SYSOUT=* Defines Output Data Set
//SORTWK01 DD SPACE=(CYL,10),UNIT=SYSDA Defines Intermediate Storage
//SYSIN DD *
INREC FIELDS=(1,14, Selects Record Fields
              98)
SORT FIELDS=(5,10,ZD,A) Sorts Records
.
.
.

```

Figure 86. JCL and Required Control Statements

Figure 87 shows the input record after INREC processing.

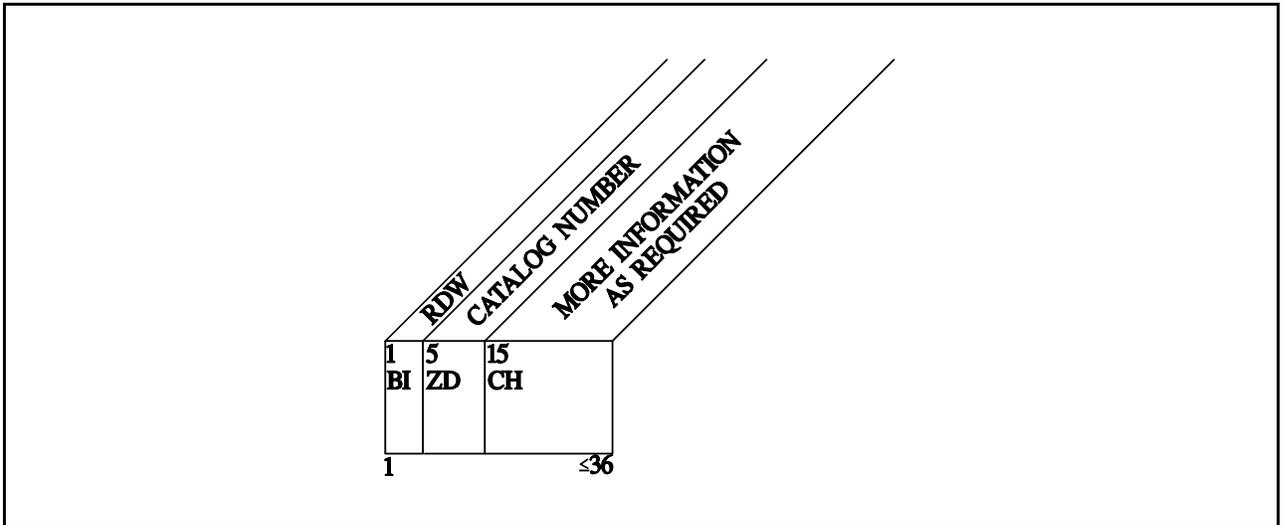


Figure 87. Form of Post-INREC Record

Explanation: When selecting fields on variable-length records, you must observe these two restrictions: (1) The position of the RDW cannot be affected; and (2) at least one byte from the fixed-length portion of the record, in addition to the RDW, must be specified. On the above INREC statement, the first 14 bytes of each record—the 4-byte RDW and the fixed-length Catalog Number field—are retained unchanged. The next field—which contains more information, as required—is indicated only by position (98) since it is of variable-length. This causes the entire variable-length portion of the record (beginning with byte 98) to be included after the initial 14 bytes of the post INREC record. SyncSort automatically adjusts the RDW to reflect the new record length.

Combining Records within a File

Sometimes you may want to shorten a file by consolidating records that have some information in common. For example, a company’s invoice file may contain more than one record for any customer to whom multiple invoices have been issued. In some applications it might then be feasible to consolidate such records—that is, to combine records with identical Customer Name and Address fields into a single record containing the sum of that customer’s charges and payments.

The SUM control statement allows you to combine records in this way. For SUM control statement syntax, see “SUM Control Statement” on page 2.149.

Combining Records and Summing Numeric Data Fields

Example: For an inventory list, a company requires a single record for each product, indicating its item number, warehouse code, and the total quantity in stock. (Figure 88 gives the sample record layout.)

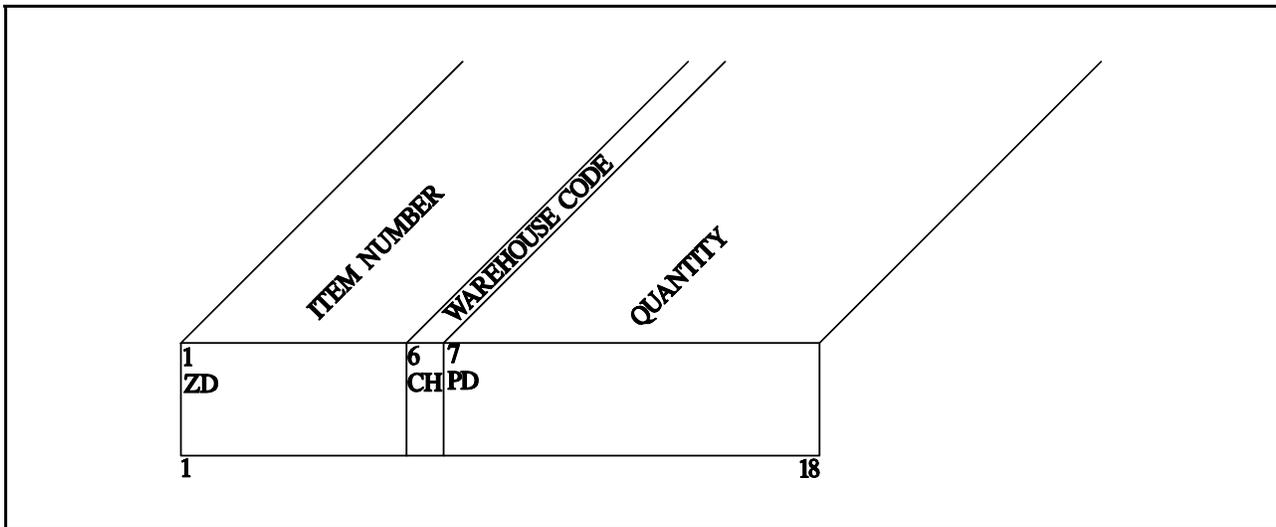


Figure 88. Input Record Layout

To combine those inventory records with identical item numbers and warehouse codes and to produce the required list, the following is coded.

```

//INVENT   JOB                               Gives the Jobname
//          EXEC   PGM=SYNCSORT              Identifies the Program
//SYSOUT   DD     SYSOUT=*                   Assigns SyncSort Messages
//*                                                to I/O Device
//SORTIN   DD     DSN=WRHSE.INVENT,DISP=SHR  Defines Input Data Set
//SORTOUT  DD     SYSOUT=*                   Defines Output Data Set
//SORTWK01 DD     SPACE=(CYL,6),UNIT=SYSDA   Defines Intermediate Storage
//SYSIN    DD     *
          SORT  FIELDS=(6,1,CH,A,1,5,ZD,A)   Sorts Records
          SUM   FIELDS=(7,12,PD)              Combines Records and
                                                Sums Numeric Data

```

Figure 89. JCL and Required Control Statements

Explanation: The list is generated by sorting on the Warehouse Code field (6,1,CH) and the Item Number field (1,5,ZD). Records that have identical information in both these fields are combined into a single record that contains the sum or total of those records' Quantity fields (7,12,PD). That is, the single record will show how many items with the same number are in each warehouse.

Eliminating Duplicate Records

Example: A mailing list is being prepared from an invoice file. To eliminate duplicate entries, any multiple invoice records for the same customer are combined into a single record. (Figure 90 gives the sample record layout.)

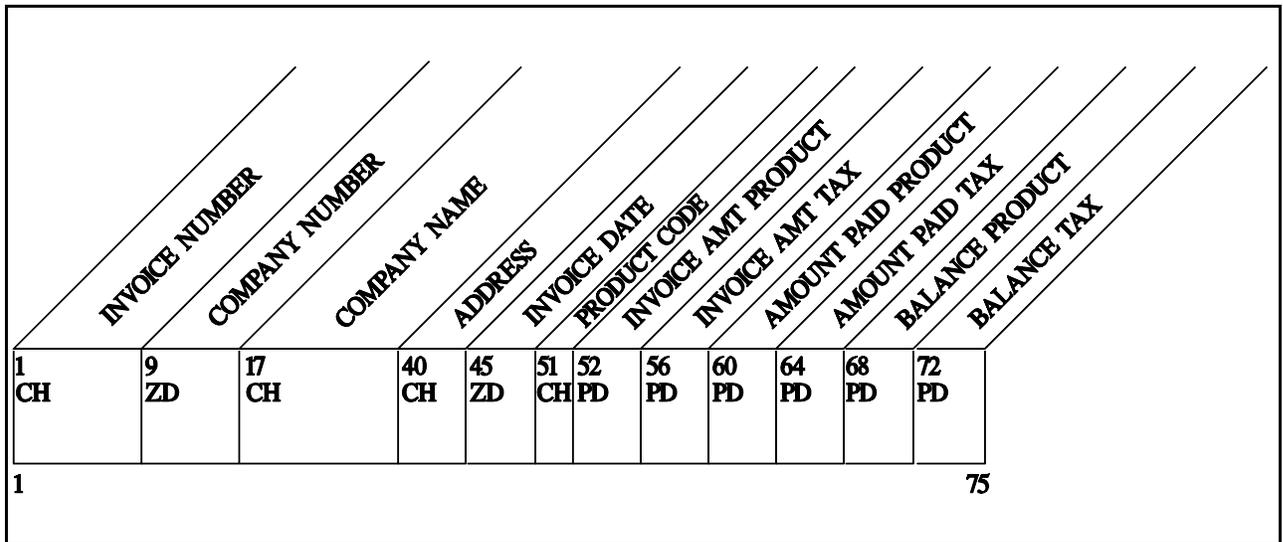


Figure 90. Input Record Layout

To combine multiple invoice records and generate the mailing list, the following is coded.

```

//MAILLIST JOB                               Gives the Jobname
//          EXEC PGM=SYNCSORT                 Identifies the Program
//SYSOUT   DD   SYSOUT=*                     Assigns SyncSort Messages
//*                                               to I/O Device
//SORTIN   DD   DSN=INV.MAST,DISP=SHR        Defines Input Data Set
//SORTOUT  DD   SYSOUT=*                     Defines Output Data Set
//SORTWK01 DD   SPACE=(CYL,5),UNIT=SYSDA    Defines Intermediate Storage
//SYSIN    DD   *
          INREC FIELDS=(17,28)                Selects Relevant Fields
          SORT  FIELDS=(1,23,CH,A)           Sorts Records. Reference is to
                                               Post INREC Record
          SUM   FIELDS=NONE                  Eliminates Duplicate Records

```

Figure 91. JCL and Required Control Statements

Explanation: To prepare the customer mailing list, the only information required from the invoice records is located in the Company Name field (17,23) and the Address field (40,5), which are selected by the INREC statement. Sorting these records in ascending order by company name generates an alphabetical list. Then, because the file contains a record for every transaction, the SUM statement is used to avoid duplicate listings of customers who have had more than one transaction. Note that because none of the fields contains numeric data to be summed, the FIELDS=NONE parameter is used.

Making Output Records Printable and Easy to Read

Because data is usually stored in a compact format, it can be difficult, if not impossible, to read when printed. For example, on a typical input record, there will be no blank space between fields, numeric data will sometimes be lost in leading and trailing zeros, and some data will be in unprintable format.

After processing, you will probably want to edit this data so that it is easy to read. This is bound to entail one or more of the following tasks:

- reordering the position of record fields
- inserting blanks between fields
- inserting binary zeros
- converting numeric data from unprintable to printable format
- converting data to printable hexadecimal format
- using masks or edit patterns to insert dollar signs, decimal points, slashes, and the like.
- formatting the data in a record field on multiple output lines

SyncSort's OUTREC processing, specified either as a control statement or as a parameter on the OUTFIL statement, can perform these and other editing functions. The OUTREC control statement is described below. Any number of the OUTREC statement's subparameters may be specified and must be coded in the order in which the fields will appear in the reformatted record. (Note that when specified as a parameter of OUTFIL, OUTREC is coded identically as for a control statement except that the keyword FIELDS is not used.) See "OUTREC Control Statement Format" on page 2.89 for the complete format of the OUTREC statement.

Reordering the Positions of Record Fields

Example: A data center has decided to reorder the positions of the data fields in masterfile records after sorting them. (Figure 92 gives the layout for the masterfile record.)

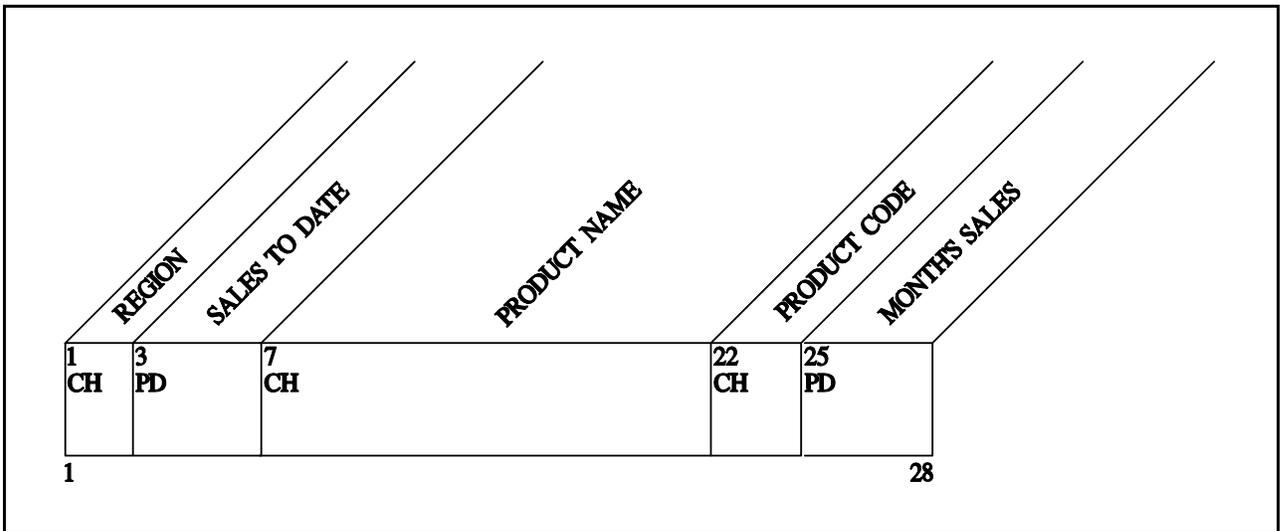


Figure 92. Input Record Layout

To sort the records alphabetically by product name and reposition the data fields, the following is coded:

```

//SORTPROD JOB                               Gives the Jobname
//          EXEC PGM=SYNCSORT                 Identifies the Program
//SYSOUT DD SYSOUT=*                         Assigns SyncSort Messages
//*                                           to I/O Device
//SORTIN DD DSN=PROD.SALES,DISP=SHR          Defines Input Data Set
//SORTOUT DD SYSOUT=*                        Defines Output Data Set
//SORTWK01 DD SPACE=(CYL,10),UNIT=SYSDA     Defines Intermediate Storage
//SYSIN DD *
      SORT  FIELDS=(7,15,CH,A)                Sorts Records
      OUTREC FIELDS=(22,3,                    Repositions Fields on
                    7,15,                     Output Records
                    1,2,
                    25,4,
                    3,4)

```

Figure 93. JCL and Required Control Statements

Figure 94 shows the output record after OUTREC processing.

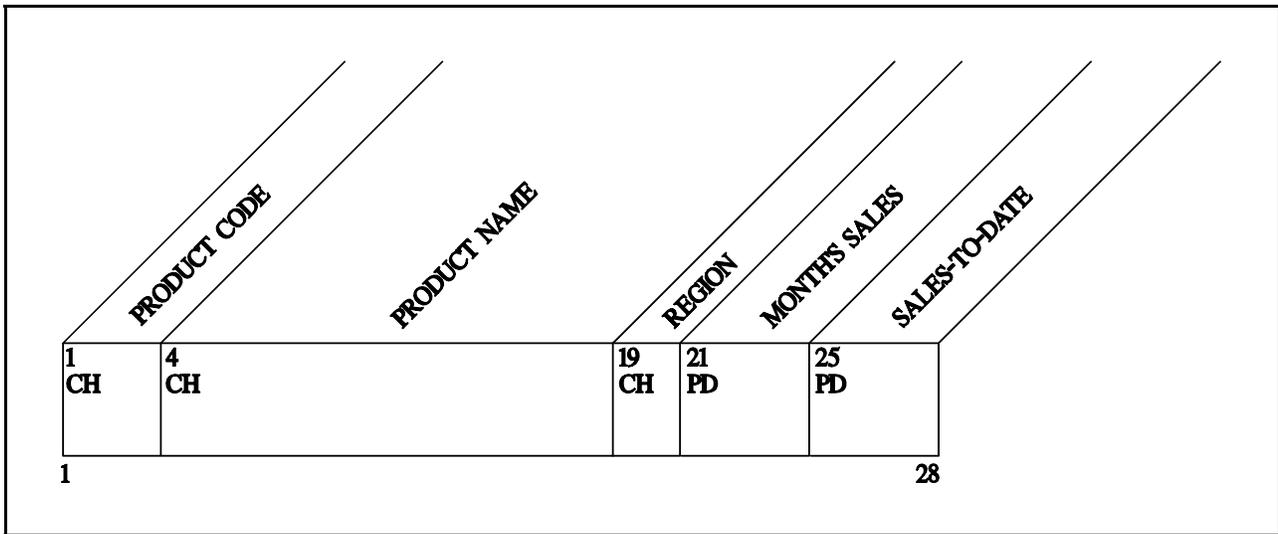


Figure 94. Post-OUTREC Record Layout

Explanation: After the records are sorted alphabetically by company name (7,15,CH), OUTREC processing moves the Product Code field (22,3) to the first byte of the record, the Product Name field (7,15) to the fourth byte, the Region field (1,2) to the nineteenth byte, the Month's Sales field (25,4) to the twenty-first byte, and the Sales to Data field (3,4) to the twenty-fifth byte.

Inserting Blanks and Repositioning Record Fields

Example: The central office of a commercial bank requires that each branch present its masterfile at the end of every month in the format outlined in Figure 95, Branch A, however, has formatted its masterfile records as outlined in Figure 96.

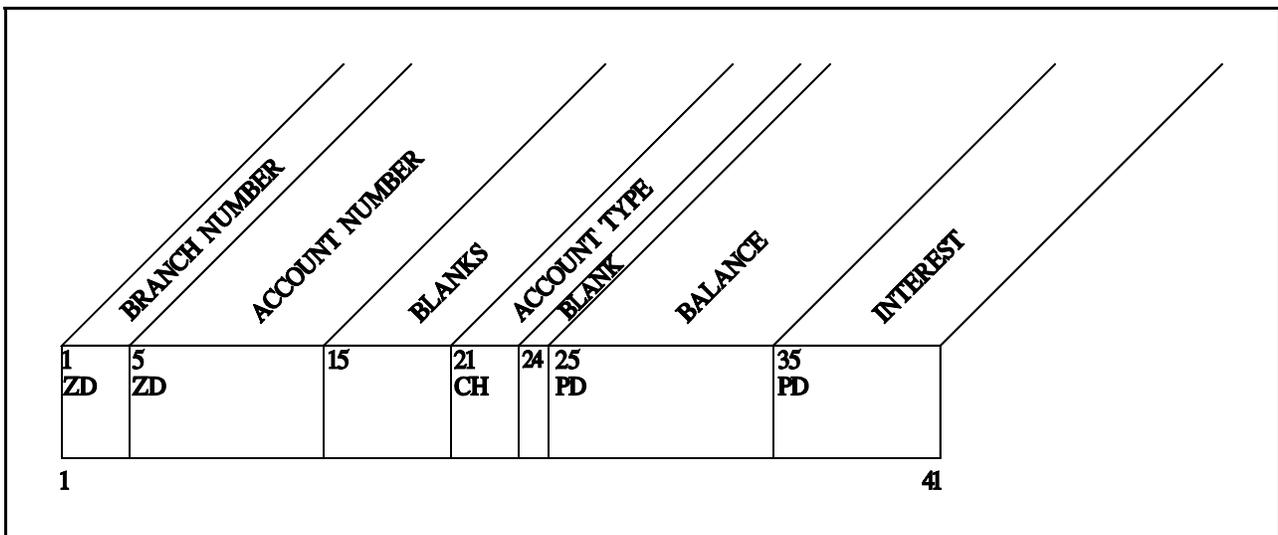


Figure 95. Required Format

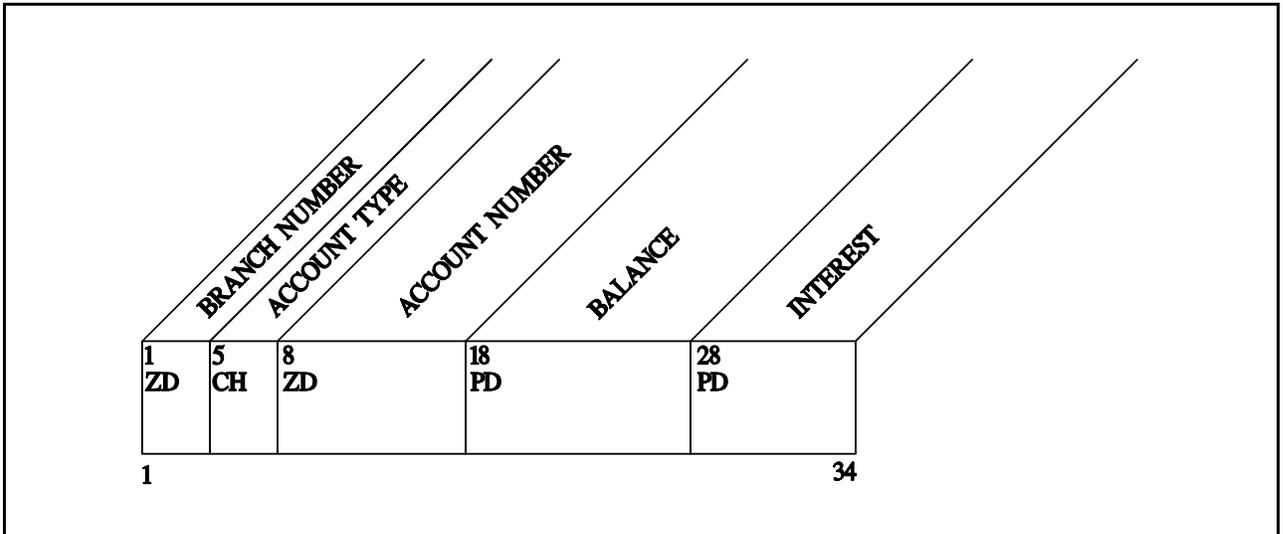


Figure 96. Input Record Layout

To reformat its masterfile records to conform to central-office specifications, the following is coded. Since the records do not require sorting, the SyncSort copy feature is used.

```

//FORMAT   JOB                               Gives the Jobname
//          EXEC   PGM=SYNCSORT              Identifies the Program
//SYSOUT   DD     SYSOUT=*                   Assigns SyncSort Messages
//*                                                to I/O Device
//SORTIN   DD     DSN=ACCT.MAST,DISP=SHR     Defines Input Data Set
//SORTOUT  DD     SYSOUT=*                   Defines Output Data Set
//SYSIN    DD     *
          SORT   FIELDS=COPY                 Copies Records
          OUTREC FIELDS=(1,4,                Repositions Fields on
                        8,10,                Output Records
                        6X,
                        5,3,
                        1X,
                        18,17)

```

Figure 97. JCL and Required Control Statements

Figure 98 shows the effect of OUTREC processing on the output record.

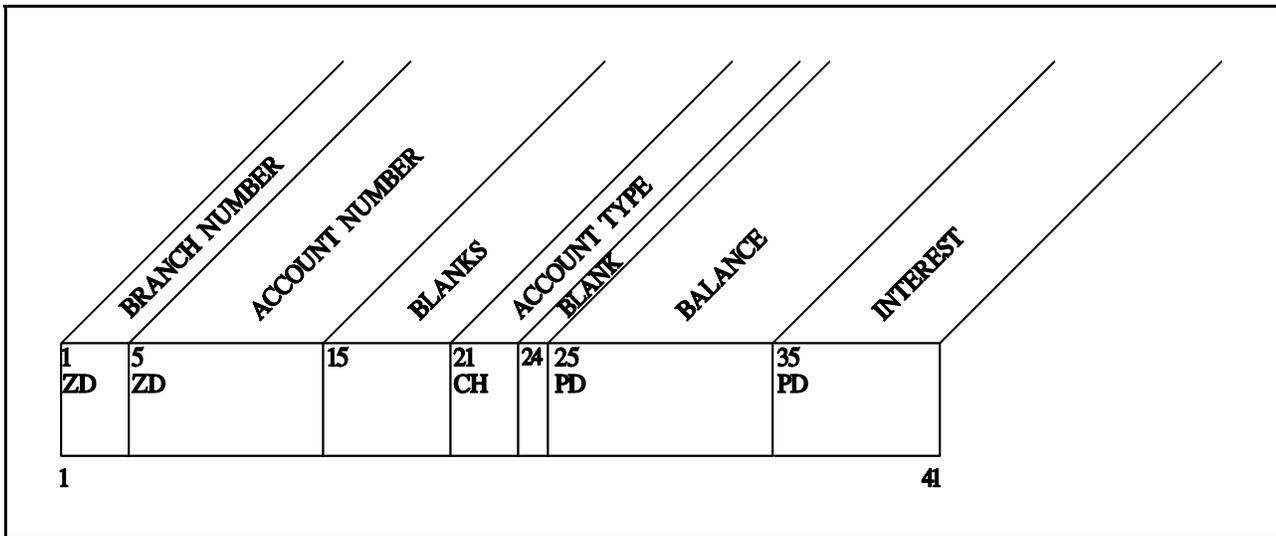


Figure 98. Post-OUTREC Record Layout

Explanation: After the records are copied, OUTREC specifies two types of reformatting: (1) repositioning data fields and (2) inserting blanks between fields. As shown in Figure 98, two fields have been repositioned: the Account Type field now begins on the twenty-first byte as opposed to the fifth byte, and the Account Number field begins on the fifth byte rather than on the eighth. Also, blanks have been inserted using the nX entry to specify the number (n) of blanks. Six blanks have been inserted after the Account Number field and a single blank after the Account Type field. Since the Balance field and Interest field are contiguous, they are treated as a single field in this application.

Inserting Binary Zeros

Example: A manufacturing firm has decided to expand its product line. However, because the Item Number field on its inventory records is too small, the records must be reformatted to allow for more columns for the new products. The Item Number is kept in packed-decimal, PD, format, and the firm wants to add 4 bytes to the current 2 byte field. The new bytes are to precede the current two bytes. Figure 99 gives the input record layout.

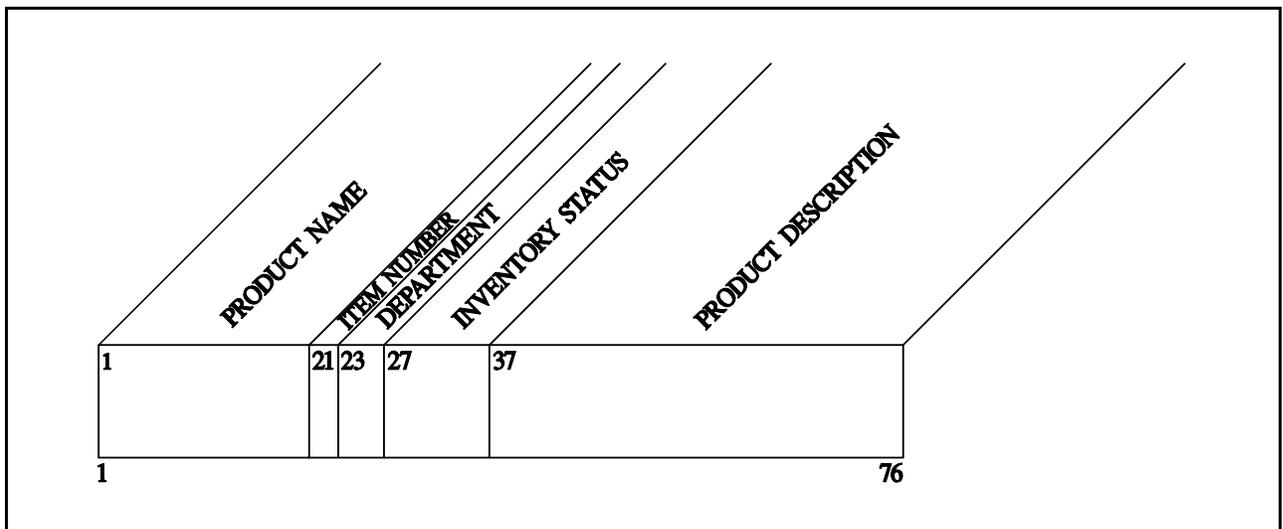


Figure 99. Input Record Layout

To copy the records and insert the 4 bytes of binary zeros, the following is coded.

```

//SORTCP   JOB                               Gives the Jobname
//          EXEC PGM=SYNCSORT                 Identifies the Program
//SYSOUT   DD   SYSOUT=*                     Assigns SyncSort
//*                                               Messages to I/O Device
//SORTIN   DD   DSN=INV.REC,DISP=SHR         Defines Input Data Set
//SORTOUT  DD   DSN=INV.REC.OUT,DISP=(NEW,KEEP), Defines Output Data Set
//          UNIT=SYSDA,SPACE=(TRK,5),
//          VOL=SER=000111
//SYSIN    DD   *
           SORT      FIELDS=COPY             Copies Records
           OUTREC    FIELDS=(1,20,         Inserts Binary Zeros &
                       4Z,                 Reformats Records
                       25:21,56)

```

Figure 100. JCL and Required Control Statements

The effect of OUTREC processing is shown in Figure 101 below.

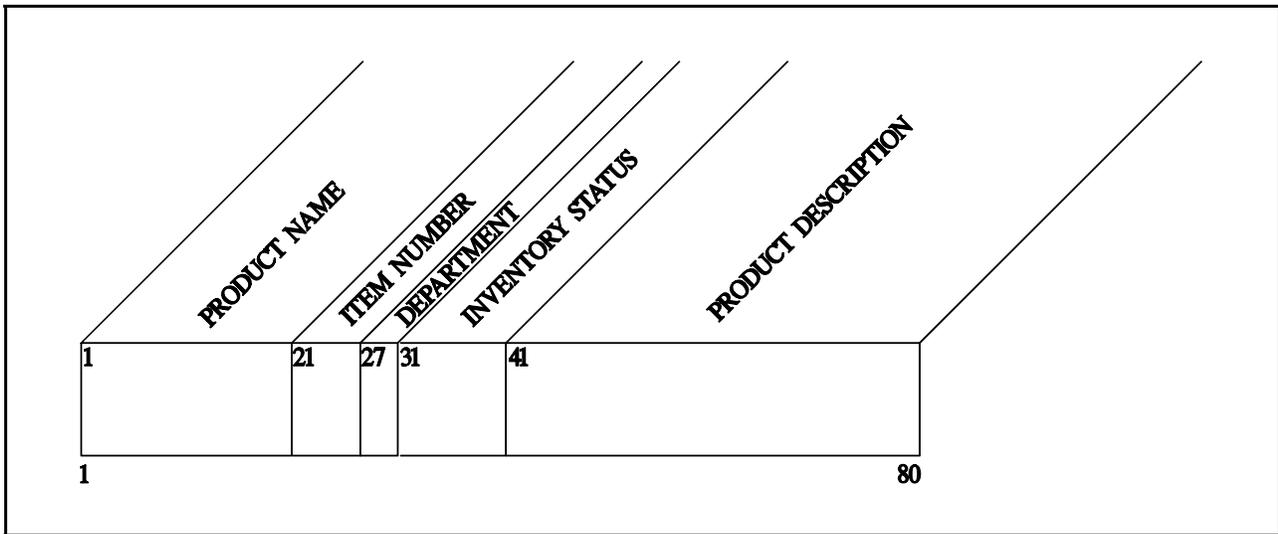


Figure 101. Post-OUTREC Record Layout

Explanation: The records are copied, and OUTREC processing adds 4 bytes of binary zeros (4Z) to the beginning of the Item Number field (21,2). To allow for the 4 additional bytes, the original Item Number field and the fields following it are all copied after the 4 inserted bytes of zeros.

Converting Unprintable Data to Readable Form

Example: For a file of invoice records sorted by company name, the Invoice Amount, Amount Paid, and Balance Due fields are to be converted from packed-decimal to printable format. In addition, any leading zeros will be suppressed and both commas and decimal points will be inserted. (Figure 102 gives the input record layout.)

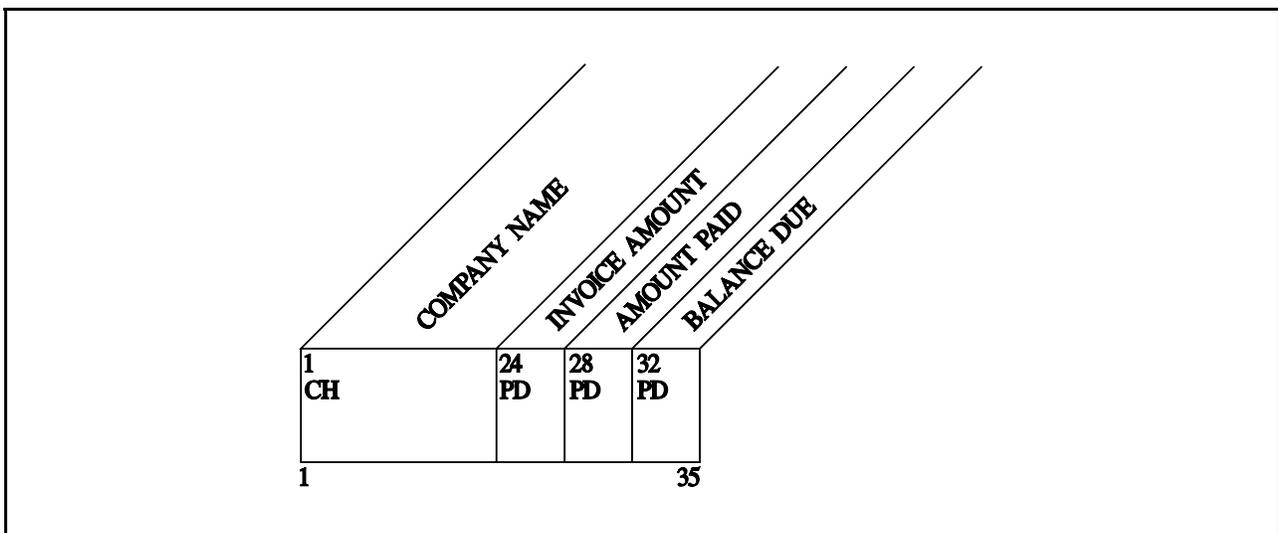


Figure 102. Input Record Layout

To sort the records, convert the three fields of packed-decimal data, and insert the commas and decimal points, the following is coded.

```

//INVOICE  JOB                               Gives the Jobname
//          EXEC  PGM=SYNCSORT                Identifies the Program
//SYSOUT   DD    SYSOUT=*                    Assigns SyncSort
//*                                               Messages to I/O Device
//SORTIN   DD    DSN=NEWINV,DISP=SHR         Defines Input Data Set
//SORTOUT  DD    SYSOUT=*                    Defines Output Data Set
//SORTWK01 DD    SPACE=(CYL,5),UNIT=SYSDA   Defines Intermediate
                                                Storage

//SYSIN    DD    *
          SORT  FIELDS=(1,23,CH,A)           Sorts Records
          OUTREC FIELDS=(17:1,23,
                        52:24,4,PD,M2,      Repositions Record Fields
                        74:28,4,PD,M2,      and Converts Data
                        96:32,4,PD,M2)

```

Figure 103. JCL and Required Control Statements

The effect of OUTREC processing on the input record is shown in Figure 104 below.

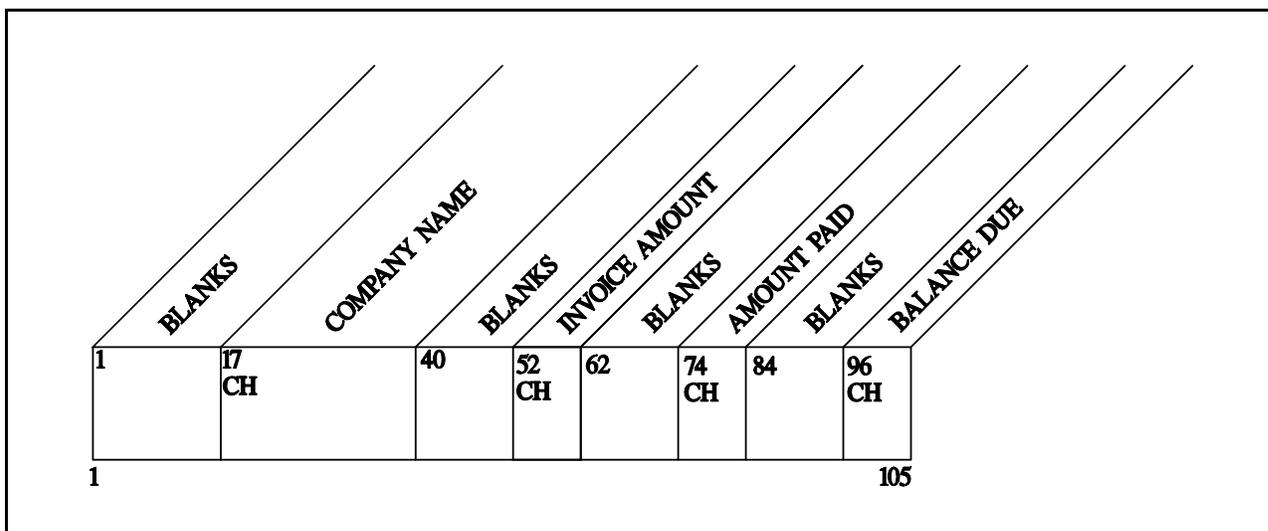


Figure 104. Post-OUTREC Record Layout

Explanation: First the records are sorted alphabetically by company name (1,23,CH). Then, three fields--the Invoice Amount (24,4,PD), the Amount Paid (28,4,PD), and the Balance Due (32,4,PD)--are converted from packed-decimal (PD) into readable format and editing by a SyncSort editing mask (M2) that suppresses the printing of leading zeros and inserts the appropriate commas and decimal points. The number-colon entries (c:) that precede each of the four fields assign a new starting position or, when printing, column for each of the four fields. For example, the Company Name field, which originally began in byte 1 for a length of 23 bytes, now begins in byte 17; the Invoice Amount field, which began in byte

24, begins in byte 52, and so on. Note that after the data is converted and edited, the lengths of the packed-decimal fields increase from four bytes each to ten bytes and that the fields are each separated by twelve blanks.

Converting Unprintable Data to Hexadecimal Format

Example: A bank has discovered that some errors were made in recording the Account Numbers of some of its customers. Specifically, on the transaction records, some Account Number fields, which should contain only packed-decimal, PD, data, appear to contain data that is not valid packed-decimal. Figure 105 shows the input record layout.

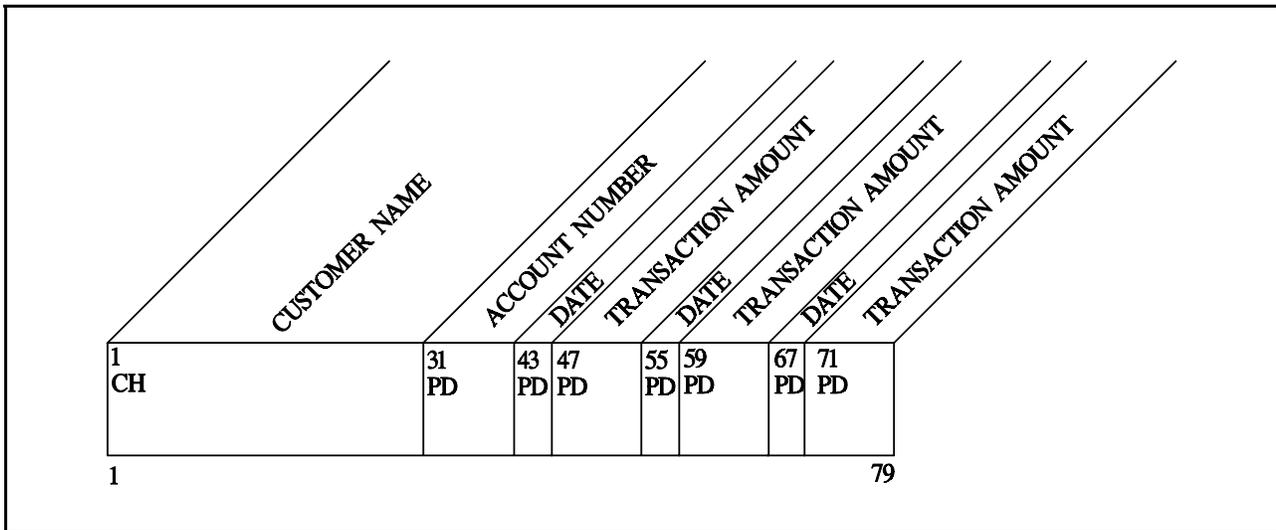


Figure 105. Sample Input Record Layout

In order to find the invalid data, the following is coded.

```

//SORTHEX JOB                               Gives the Jobname
//          EXEC PGM=SYNCSORT                Identifies the Program
//SYSOUT   DD   SYSOUT=*                    Assigns SyncSort
//*                                              Messages to I/O Device
//SORTIN   DD   DSN=TRANS.RECS,DISP=SHR     Defines Input Data Set
//SORTOUT  DD   SYSOUT=*                    Defines Output Data Set
//SYSIN    DD   *
          SORT   FIELDS=COPY                Copies Records
          OUTREC FIELDS=(1,30,              Reformats Output Records
                        36:31,12,HEX)      and Converts Data

```

Figure 106. JCL and Required Control Statements

The effect of OUTREC processing on the input record is shown in Figure 107.

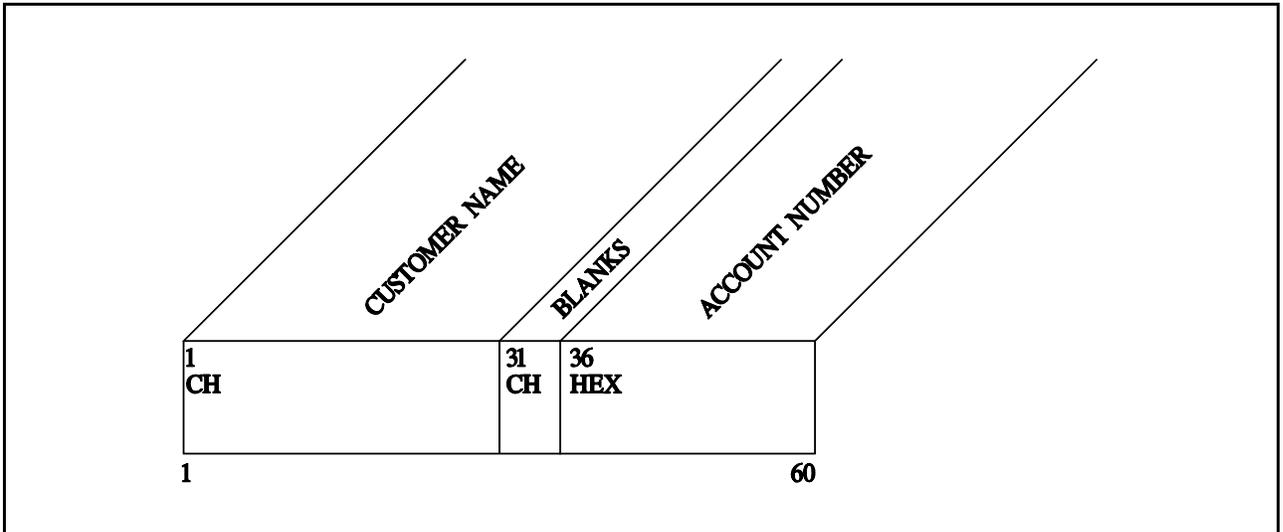


Figure 107. Sample Post-OUTREC Record Layout

Explanation: The records are copied, and OUTREC processing reformats the output record to contain the Customer Name field (1,30) followed in column 36 by the Account Number field converted to hexadecimal format (31,12,HEX). Blanks are automatically inserted in the unspecified columns (31,5). Note that converting the Account Number data to printable hexadecimal expands the original 12-byte field to 24 bytes. The bank can now read the Account Number field in hexadecimal format to determine which records contain invalid data.

Converting and Editing Unprintable Data

Example: For an Outstanding Payments report, the packed-decimal Amount Due field on a company's invoice records is converted to printable format and edited with a floating dollar sign, commas, and a decimal point. In addition, to make the output easy to read, ten blanks are inserted between the Company Name field and the Amount Due field. (Figure 108 gives the input record layout.)

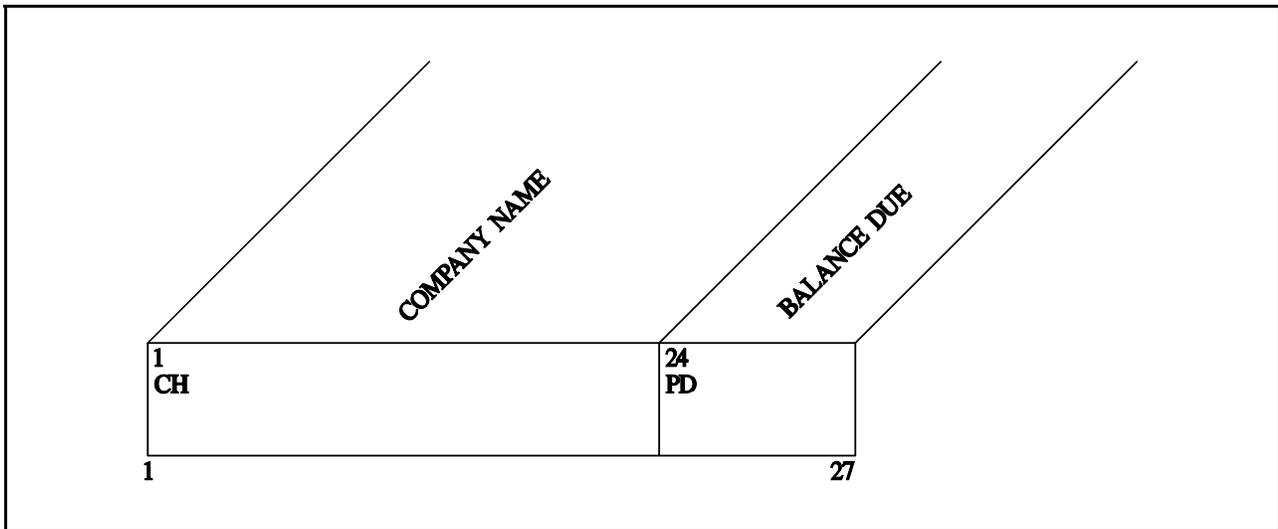


Figure 108. Input Record Layout

To sort the records and accomplish the conversion and editing, the following is coded.

```

//PAYMNT  JOB                               Gives the Jobname
//          EXEC  PGM=SYNCSORT                Identifies the Program
//SYSOUT   DD    SYSOUT=*                     Assigns SyncSort Messages
//*                                               to I/O Device
//SORTIN   DD    DSN=INVOICE,DISP=SHR        Defines Input Data Set
//SORTOUT  DD    SYSOUT=*                     Defines Output Data Set
//SORTWK01 DD    SPACE=(CYL,5),UNIT=SYSDA    Defines Intermediate Storage
//SYSIN    DD    *
          SORT  FIELDS=(1,23,CH,A)           Sorts Records
          OUTREC FIELDS=(1,23,
                        10X,
                        24,4,PD,EDIT=($II,IIT.TT))
Converts and Edits Data
and Inserts Blanks

```

Figure 109. JCL and Required Control Statements

Figure 110 shows the effect of OUTREC processing on the input record.

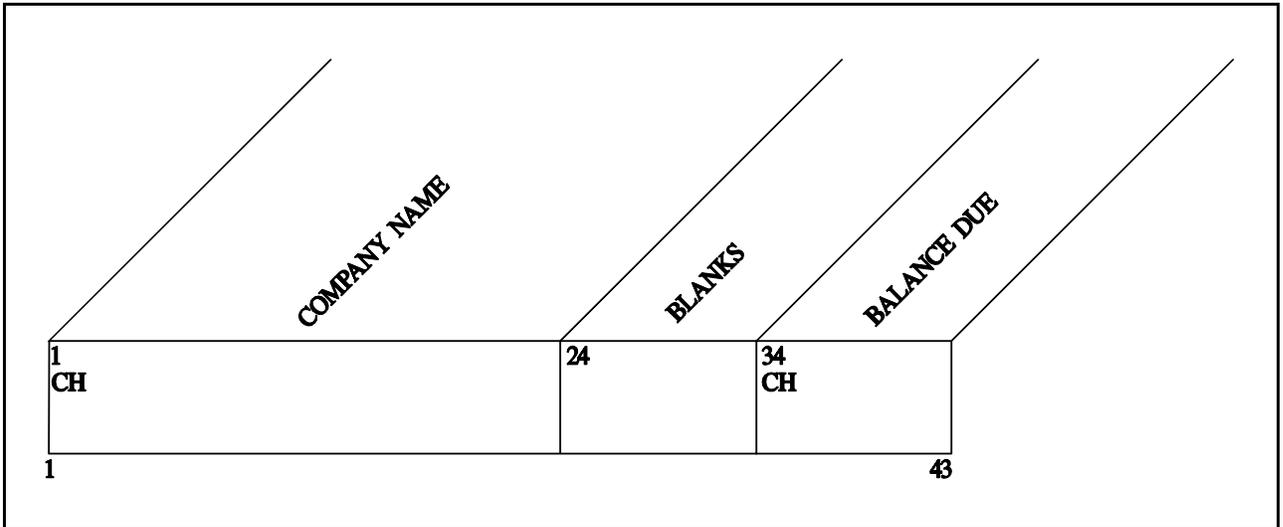


Figure 110. Post-OUTREC Record Layout

Explanation: First the records are sorted alphabetically by Company Name (1,23,CH). Next, OUTREC processing inserts 10 blanks (10X) between the Company Name field (1,23) and the Balance Due field (24,4,PD). OUTREC processing also converts this packed-decimal field to printable format and edits it with the user-provided pattern specified on the EDIT subparameter, EDIT=(\$II,IIT.TT). This pattern provides for a floating dollar sign as well as the appropriate comma and decimal point. The Is indicate that leading zeros should *not* be printed and the Ts indicate that zeros in those positions should be printed. Note that this conversion and editing of the data cause the length of the Balance Due field to increase from its original length of four bytes to ten bytes.

Putting a Data Field in Standard Format

Example: The date field on insurance-policy records is stored in zoned-decimal format but without slashes separating the month, day, and year. After the records are sorted, these slashes will be inserted and the date will appear in the standard mm/dd/yy format. (Figure 111 gives the input record layout.)

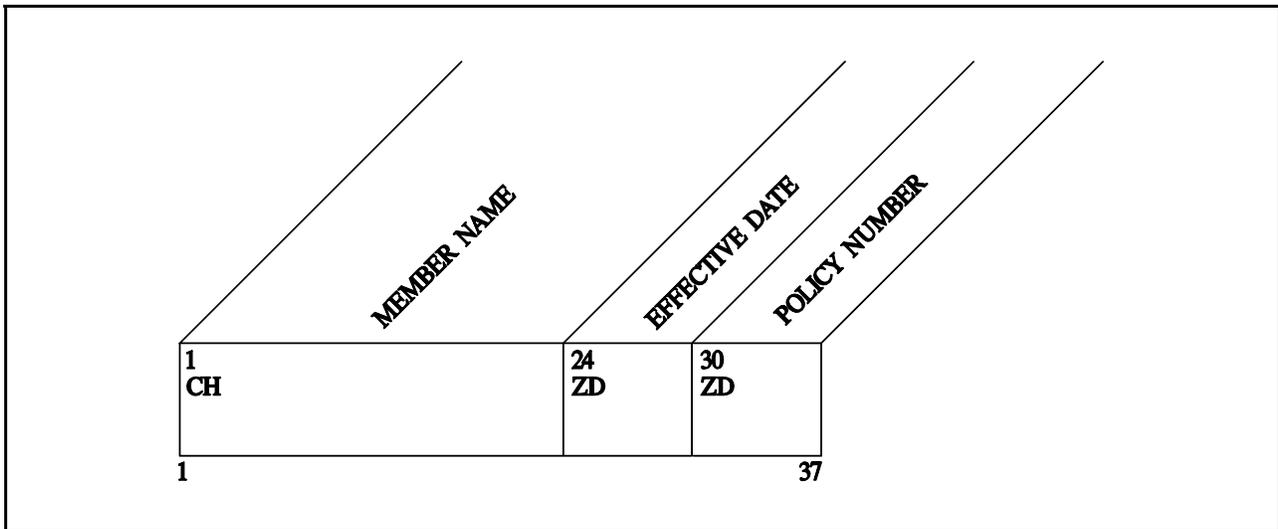


Figure 111. Input Record Layout

To sort the records and format the date field with the required slashes, the following is coded.

```

//SORTDT  JOB                               Gives the Jobname
//          EXEC  PGM=SYNCSORT              Identifies the Program
//SYSOUT  DD    SYSOUT=*                    Assigns SyncSort Messages
//*                                              to I/O Device
//SORTIN  DD    DSN=NEW.POLCY,DISP=SHR      Defines Input Data Set
//SORTOUT DD    SYSOUT=*                    Defines Output Data Set
//SORTWK01 DD   SPACE=(CYL,5),UNIT=SYSDA   Defines Intermediate Storage
//SYSIN   DD    *
          SORT FIELDS=(1,23,CH,A)           Sorts Records
          OUTREC FIELDS=(1:1,23,           Edits Data and Repositions
                        30:24,6,ZD,M9,     Record Fields
                        45:30,8)

```

Figure 112. JCL and Required Control Statements

The effect of OUTREC processing is shown in Figure 113.

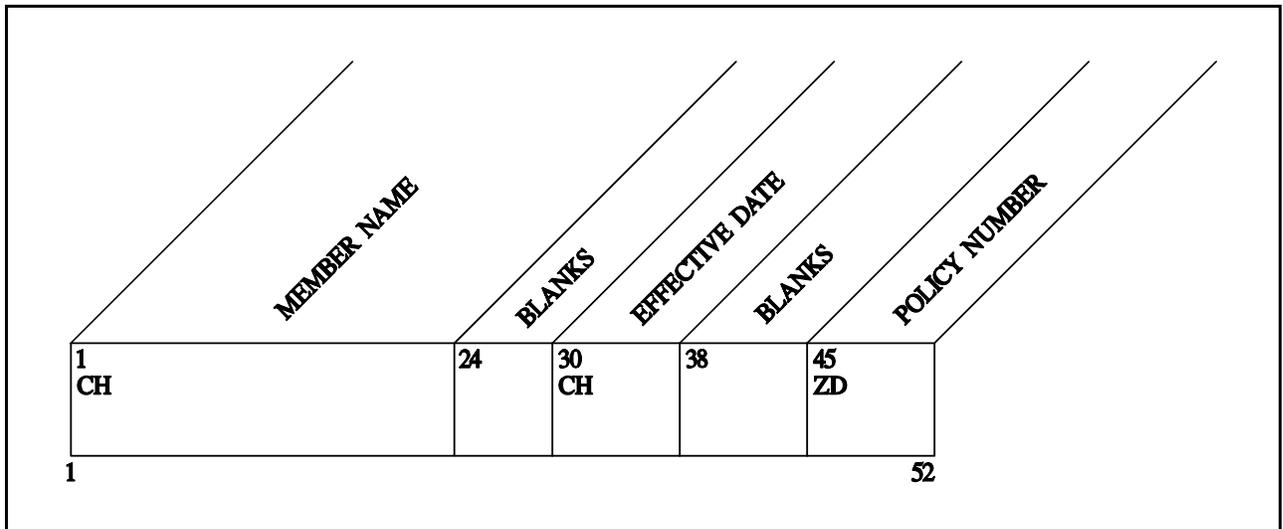


Figure 113. Post-OUTREC Record Layout

Explanation: The records are sorted alphabetically by Member Name (1,23,CH). The OUTREC statement repositions the Effective Date field (24,6,ZD) and the Policy Number field (30,8,ZD) in columns 30 and 45 respectively, leaving blanks between each of the three fields. In addition, the OUTREC statement edits the Effective Date field with an M9 editing mask that places slashes between the month, date, and year. Note that editing the Date field increases its size from six to eight bytes.

Converting from Variable to Fixed-Length Format

Example: In this example, there are three output files. The first is variable and the remaining two are fixed-length format. The variable output file is the standard output file from the sort. In order to convert the output from variable to fixed-length format, you should specify CONVERT on the OUTREC parameters of each of your OUTFIL control statements. The following are the JCL and control statements to effect this result.

```

//          JOB
//          EXEC      PGM=SYNCSORT
//SYSOUT    DD      SYSOUT=*
//SORTWK01  DD      SPACE=(CYL,5),UNIT=SYSDA
//SORTIN    DD      DSN=VARIN,DISP=SHR
//SORTOUT   DD      UNIT=SYSDA,SPACE=(CYL,(1,1)),
//              DISP=(,PASS),DSN=&&VAROUT
//SORTOF1   DD      UNIT=SYSDA,SPACE=(CYL,(1,1)),
//              DISP=(,PASS),DSN=&&FIX1OUT
//SORTOF2   DD      UNIT=SYSDA,SPACE=(CYL,(1,1)),
//              DISP=(,PASS),DSN=&&FIX2OUT
//SYSIN     DD      *
              SORT FIELDS=(1,23,CH,A,28,2,CH,A)
              OUTFIL FILES=1,
                  INCLUDE=(28,2,CH,EQ,C'92'),
                  OUTREC=(1,23),CONVERT
              OUTFIL FILES=2,
                  INCLUDE=(28,2,CH,EQ,C'93'),
                  OUTREC=(1,23),CONVERT

```

Figure 114. Using the CONVERT Parameter

Printing Input Records on Multiple Output Lines

Example: In this example, five input record fields, shown in Figure 115, are copied to an output file with each field printed as a separate output line.

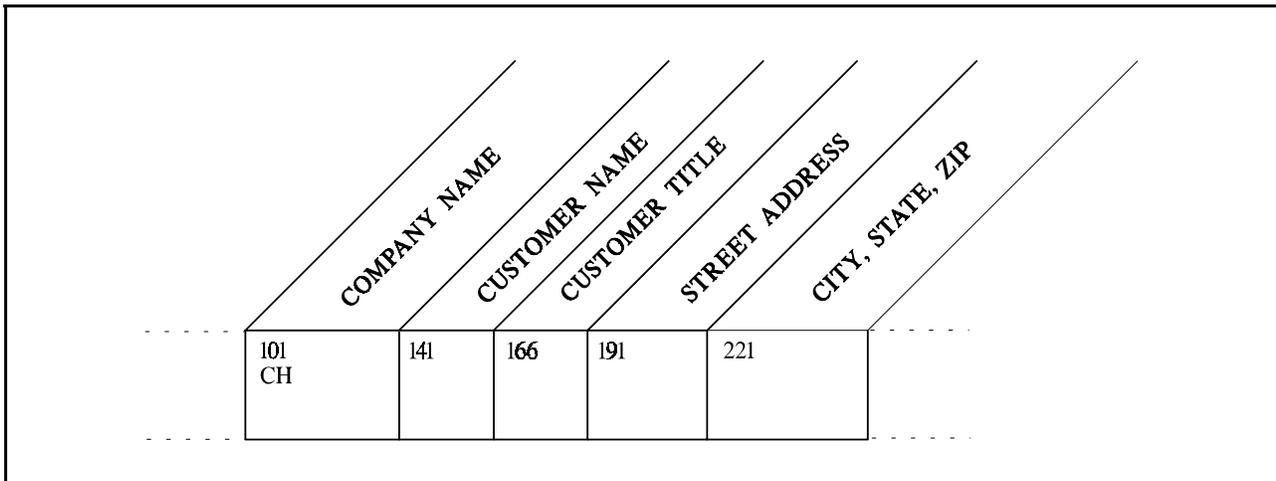


Figure 115. Input Record Layout

Multiple output lines are created by specifying a newline character, i.e. / (slash), in the OUTREC parameter of an OUTFIL control statement. As shown in Figure 116, the newline character follows the specification of each input field's starting position and length.

//MULTILIN JOB			<i>Gives the Jobname</i>
//	EXEC	PGM=SYNCSORT	<i>Identifies the Program</i>
//SYSOUT	DD	SYSOUT=*	<i>Assigns SyncSort Messages to I/O Device</i>
//*			
//SORTIN	DD	DSN=&&DATA,DISP=SHR	<i>Defines Input Data Set</i>
//*			
//SORTWK01	DD	UNIT=SYSDA,SPACE=(CYL,(3,3))	<i>Defines Intermediate Storage</i>
//*			
//SORTWK02	DD	UNIT=SYSDA,SPACE=(CYL,(3,3))	<i>Defines Intermediate Storage</i>
//*			
//SORTOUT	DD	SYSOUT=*	<i>Defines Output Data Set</i>
//*			
//SYSIN	DD	*	
		SORT FIELDS=(101,40,CH,A)	<i>Sorts Records</i>
		OUTFIL CONVERT,	<i>Converts Data</i>
		HEADER2=('CUSTOMER ADDRESS LIST',3/),	<i>Prints a Page Header</i>
*			
		OUTREC=(101,40,/,	<i>Prints the Data in the Field and Starts a New Output Line</i>
*			
*			
		141,25,/,	<i>As Above</i>
		166,25,/,	<i>As Above</i>
		191,30,/,	<i>As Above</i>
		266,35,2/)	<i>As Above but Starts 2 New Output Lines</i>
*			

Figure 116. JCL and Control Statements for Multiline Output

Once SyncSort has printed the data in the COMPANY NAME field, it starts a new output line, prints on it the data in the next field, CUSTOMER NAME, starts a new line, and so forth. After printing the contents of the last field (CITY, STATE AND ZIP), SyncSort creates two new lines (2/).

Figure 117 provides an excerpt from the output file where the input record is formatted on multiple lines. A blank line appears in the second and third set of multi-line output because the corresponding input record fields (i.e. CUSTOMER TITLE and CUSTOMER NAME) were blank.

```

CUSTOMER ADDRESS LIST

AARON'S ROD INC.                               First Set of Multiline Output
DAVID LAURENCE
SYS PROG
6936 YOUNGMAN BLVD.
GREAT NECK             CT.    06854

BLAKE'S VISION TECHNOLOGY                       Second Set of Multiline Output
MR. N. FRYE

261 ALBION PLACE
SEA BRIGHT             NJ.    08572

COLTRANE & COMPANY                               Third Set of Multiline Output

DATA CENTER MANAGER
300 DORIAN AVENUE
NEW YORK               NY.    11220

```

Figure 117. Sample Multiline Output

Dividing a Report into Sections

When printing sorted output, you may want to divide it into sections. For example, after sorting a personnel file alphabetically by company name and department, you might want to print each department's records as a separate section and leave some blank lines between each section. You might even want to print each section as a separate page of the report. SyncSort allows you to print groups of records that have identical information in one or more sort fields as sections and to separate each section by a specified number of lines or a page break.

To divide output into sections, use the SECTIONS parameter on the OUTFIL control statement. For complete syntax of the SECTIONS parameter, see "SECTIONS Parameter (Optional)" on page 2.80.

Dividing Output into Sections

Example: A personnel roster is to be divided into sections by Department. (Figure 118 presents the layout for the input record.)

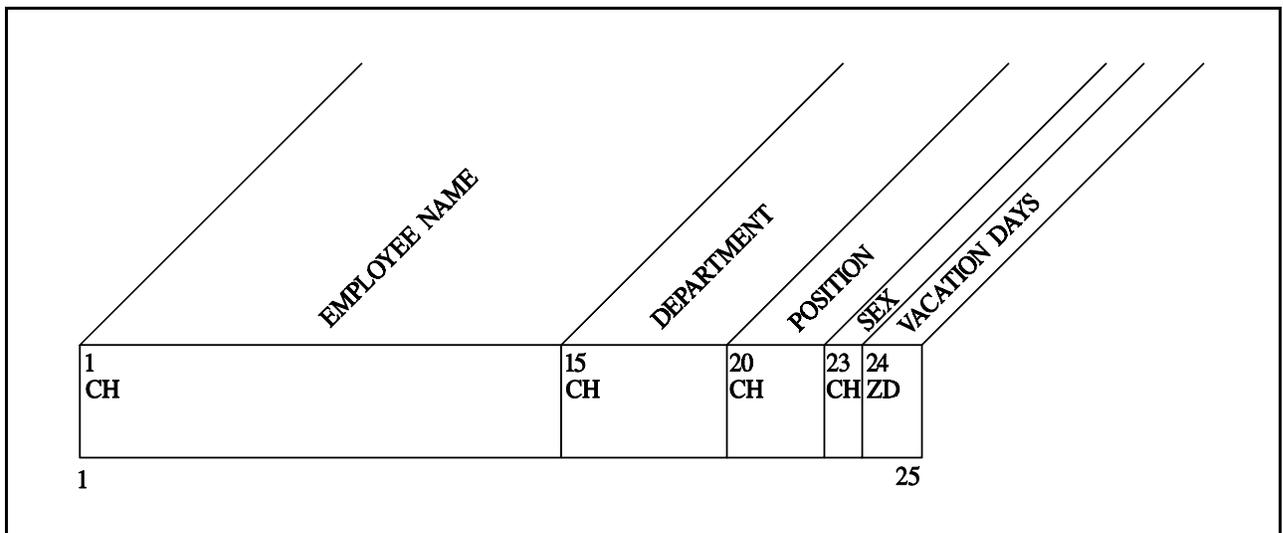


Figure 118. Input Record Layout

To sort the records and generate a list that is divided by Department, the following is coded.

```

//ROSTER   JOB                               Gives the Jobname
//         EXEC   PGM=SYNCSORT                Identifies the Program
//SYSOUT   DD     SYSOUT=*                    Assigns SyncSort Messages
//*                                               to I/O Device
//SORTIN   DD     DSN=PRSNL,DISP=SHR          Defines Input Data Set
//SORTOUT  DD     SYSOUT=*                    Defines Output Data Set
//SORTWK01 DD     SPACE=(CYL,2),UNIT=SYSDA   Defines Intermediate Storage
//SYSIN    DD     *
          SORT   FIELDS=(15,5,A,1,14,A),FORMAT=CH Sorts Records
          OUTFIL OUTREC=(6:15,5,              Repositions Record Fields
                        14:1,14,
                        33:20,3,
                        44:23,1,
                        54:24,2),
          SECTIONS=(15,5,SKIP=5L)            Sections Records

```

Figure 119. JCL and Required Control Statements

A sample of the listing generated is shown in Figure 120.

ACCTG	BELL	PAT	SUP	F	03
ACCTG	EMERY	PAUL	CLK	M	04
ACCTG	JONES	MARK	CLK	M	01
ACCTG	NORTH	NANCY	MGR	F	02
ACCTG	OWEN	JERRY	CLK	M	03
ACCTG	TWAIN	JOAN	SEC	F	05
ACCTG	WEST	DONNA	CLK	F	03
PRSNL	SMITHE	JON	CLK	M	00
PRSNL	TOWERS	LINDA	CLK	F	02
PRSNL	VREES	GEORGE	CLK	M	02
PRSNL	WU	JANE	SUP	F	05
PRSNL	YOUNG	RUSS	MGR	M	03

Figure 120. Sample Output

Explanation: After the records are sorted alphabetically by Department (15,5) and Employee Name (1,14), they are divided into sections by department. That is, every time there is a change in the Department field (15,5 in the input record) the printer skips 5 lines (5L) before printing the next record. (Note, in the Sample Output above, the five-line break that occurs between ACCTG and PRSNL.) The OUTREC parameter is used to reposition the record fields and to leave blanks between them.

Writing Headers and Trailers for a Report

Headers are used to provide report, page, and section headings such as titles, page numbers, the current date, labels for each column of data, and the like. Similarly, trailers are used for report, page, and section summaries. You can use them, for example, to provide totals for columns of numeric data (see "Totaling and Subtotaling Data") or to indicate the end of a section with, say, a string of asterisks or to provide a list of abbreviations used in the report.

To generate Headers and/or Trailers, use the HEADER and TRAILER parameters of the OUTFIL control statement. For complete syntax, see "HEADER1/HEADER2 Parameters (Optional)" on page 2.68 and "TRAILER Parameters (Optional)" on page 2.73

Writing a Title Page for a Report

Example: Marketing wants a title page for its monthly departmental sales report. The three-line title will begin on line 16 and three blank lines will separate each line of the title. The three lines will start printing in columns 49, 59, and 63, respectively.

To print this title page, the following is coded:

Writing a Page Header

Example: Marketing wants the first line of every page of its departmental sales report to contain the program number, report title, page number, and date. They want the third line of every page to contain an identifying label for each column of data. Each of these lines will begin printing in column one.

To print the page header, the following is coded.

```
//DSRPT      JOB                               Gives the Jobname
//          EXEC  PGM=SYNCSORT                 Identifies the Program
//SYSOUT    DD   SYSOUT=*                     Assigns SyncSort Messages
//*                                                to I/O Device
//SORTIN    DD   DSN=MRKTNG.SALES,DISP=SHR    Defines Input Data Set
//SORTOUT   DD   SYSOUT=*                     Defines Output Data Set
//SORTWK01  DD   SPACE=(CYL,5),UNIT=SYSDA    Defines Intermediate Storage
//SYSIN     DD   *
          SORT  FIELDS=(1,15,CH,A)           Sorts Records
          .
          .
          .
          OUTFIL .
          .
          HEADER2=(1:'PGM NUMBER 5',
                  46:'DEPARTMENT SALES REPORT FOR FEBRUARY 1992',
                  101:'DATE:',
                  107:&DATE,                    Generates Page Heading
                  121:'PAGE:',
                  127:&PAGE,/,
                  1:'DEPARTMENT',
                  40:'SALES MANAGER',
                  61:'SALES REP',
                  78:'SALES THIS PERIOD',
                  103:'SALES YEAR TO DATE',/),
          .
          .
```

Figure 123. JCL and Required Control Statements

Figure 124 shows a representation of the header that is generated by the above HEADER2 parameter.

```
PGM NUMBER 5      DEPARTMENT SALES REPORT FOR FEBRUARY 1992   DATE: 02/01/
92  PAGE: 1

DEPARTMENT  SALES MANAGER  SALES REP  SALES THIS PERIOD  SALES YEAR TO DA
TE
```

Figure 124. Sample HEADER2

Explanation: The HEADER2 parameter produces the page header shown above. Because no forward spacing is specified, the page header begins on the first line of every page. Each of the HEADER2's number-colon entries (c:), for example, 1:, indicates the column in which the entry following the colon begins to print. Thus, the literal 'PGM NUMBER 5' is printed beginning in column 1, and so on. The &DATE and the &PAGE entries generate a current date and a consecutive page number, respectively. The date and the page number appear after the labels DATE: and PAGE:, which are specified like the other literals.

The double slashes (//) following the &PAGE entry direct the printer to forward space two lines, that is, to leave one blank line, before printing the next group of literals that constitute the labels for the columns of data.

Writing a Section Header

Example: Marketing wants each section of its departmental sales report to have its own heading. The heading will consist of one line containing an identifying label for each column of data. The heading will begin printing in column one.

To print the section header, the following is coded.

```

//DSRPT    JOB                               Gives the Jobname
//          EXEC  PGM=SYNCSORT              Identifies the Program
//SYSOUT   DD    SYSOUT=*                   Assigns SyncSort Messages
//*                                               to I/O Device
//SORTIN   DD    DSN=MRKTNG.SALES,DISP=SHR  Defines Input Data Set
//SORTOUT  DD    SYSOUT=*                   Defines Output Data Set
//SORTWK01 DD    SPACE=(CYL,5),UNIT=SYSDA   Defines Intermediate Storage
//SYSIN    DD    *
          SORT    FIELDS=(1,15,CH,A)        Sorts Records
          OUTFIL  OUTREC=(1:1,15,          Repositions Fields on Output
                        23:23,7,          Records and Edits Data
                        51:48,3,
                        72:60,4,PD,EDIT=( $II,IIT.TT ),
                        101:64,4,PD,EDIT=( $II,IIT.TT ),
                        114:C' '),
          SECTIONS=(1,15,SKIP=5L,          Generates Section Breaks
          HEADER3=(1:'DEPARTMENT',        Generates Section Headings
                  23:'SALES MGR',
                  48:'SALES REP',
                  68:'SALES THIS PERIOD',
                  97:'SALES YEAR TO DATE',//))

```

Figure 125. JCL and Required Control Statements

Figure 126 shows the header that is generated by the above HEADER3 subparameter.

DEPARTMENT	SALES MGR	SALES REP	SALES THIS PERIOD	SALES YEAR TO DATE
OVER COUNTER	CASEY	075	\$14,000.00	\$27,000.00
OVER COUNTER	CASEY	093	13,550.00	32,000.00
OVER COUNTER	CASEY	084	11,755.00	24,850.00
OVER COUNTER	CASEY	090	12,250.00	25,000.00
OVER COUNTER	CASEY	095	13,075.00	26,180.00
DEPARTMENT	SALES MGR.	SALES REP	SALES THIS PERIOD	SALES YEAR TO DATE
SURGICAL	KILDARE	003	\$11,750.00	\$25,320.00
SURGICAL	KILDARE	007	\$14,300.00	24,900.00
SURGICAL	KILDARE	009	11,110.00	30,850.00
SURGICAL	KILDARE	004	13,375.00	27,505.00
.
.
.

Figure 126. Sample Sections with HEADER3

Explanation: The HEADER3 subparameter on the SECTIONS parameter generates a header that prints at the beginning of each section. Its primary purpose here is to provide labels for the columns of data that appear in each section. Each of the number-colon entries (c:) specifies the column in which the entry following it should begin to print. Thus, the literal string 'DEPARTMENT' begins to print in column 1, the literal string 'SALES MGR' begins to print in column 23, and so on. Blanks are automatically inserted in the space between the columns that are specified. On the OUTREC parameter a blank has been inserted in column 114 (114:C' ') so that the output record length will equal that of the header. Note that if the HEADER3 in this example were used in conjunction with the preceding HEADER2 example, there would be no need to specify the labels for the columns of data in the HEADER2.

Using a Header to Eliminate Duplicate Information within a Section

Example: Rather than repeat the department name and sales manager, which are identical for every record included in a section of the departmental sales report, marketing wants this information to appear only once-within the section headers of the report. Therefore, the section headers' first two entries (Department and Sales Manager) will be drawn directly from the first data record in each section.

To print the section header with the input data fields, the following is coded.

```
//DSRPT      JOB                               Gives the Jobname
//          EXEC  PGM=SYNCSORT                 Identifies the Program
//SYSOUT    DD   SYSOUT=*                     Assigns SyncSort Messages
//*                                                to I/O Device
//SORTIN    DD   DSN=MRKTNG.SALES,DISP=SHR     Defines Input Data Set
//SORTOUT   DD   SYSOUT=*                     Defines Output Data Set
//SORTWK01  DD   SPACE=(CYL,5),UNIT=SYSDA     Defines Intermediate Storage
//SYSIN     DD   *
           SORT  FIELDS=(1,15,CH,A)           Sorts Records
           .
           .
           .
           OUTFIL...,                          Repositions Fields on Output
           OUTREC=(25:48,3,                    Records and Edits Data
                   37:60,4,PD,EDIT=($II,IIT.TT),
                   56:64,4,PD,EDIT=($II,IIT.TT),
                   71:C' '),
           SECTIONS=(1,15,SKIP=2L,             Generates Section Breaks
           HEADER3=(1:1,15,                   Generates Section Headings
                   16:23,7,
                   23:'SALES REP',
                   34:'SALES THIS PERIOD',
                   54:'SALES YEAR TO DATE'))
```

Figure 127. JCL and Required Control Statements

Figure 128 shows the header that is generated by the above HEADER3 subparameter.

OVER COUNTER	CASEY	SALES REP	SALES THIS PERIOD	SALES YEAR TO DATE
		075	\$14,000.00	\$27,000.00
		093	\$13,550.00	\$32,000.00
		084	\$11,755.00	\$24,850.00
		090	\$12,250.00	\$25,000.00
		095	\$13,075.00	\$26,180.00
SURGICAL	KILDARE	SALES REP	SALES THIS PERIOD	SALES YEAR TO DATE
		003	\$11,750.00	\$25,320.00
		007	\$14,300.00	\$24,900.00
		009	\$11,110.00	\$30,850.00
		004	\$13,375.00	\$27,505.00
		.	.	.
		.	.	.
		.	.	.

Figure 128. Sample Sections with HEADER3 Including Data from Input Record

Explanation: The HEADER3 subparameter on the SECTIONS parameter generates a header that prints at the beginning of each section. Its primary purpose here is to provide

individualized section headings that contain the Department Name and the Sales Manager from the records in that section as well as labels for the columns of data. The first two entries in this header, 1:1.15 and 16:23,7 (the Department Name and Sales Manager, respectively), are drawn directly from the input record to eliminate the repetition of these fields in the detail lines of each section. Note that specifying these fields in the HEADER3 eliminates the need to include them in OUTREC processing as was necessary in the preceding example. Each of the number-colon entries (c:) specifies the column in which the entry following it should begin to print. Thus, the Department field, (1,15) begins to print in column 1; the Sales Manager field, in column 16; the literal string "SALES REP", in column 48, and so on. Blanks are automatically inserted in the space between the columns that are specified. It should be pointed out that on the OUTREC parameter a blank has been inserted in column 71 (71:C' ') so that the output record length will equal that of the header.

Writing a Report Trailer or Summary

Example: The final page of marketing's departmental sales report will contain a note saying that February sales figures include residual 1992 sales not previously recorded. This note will begin on the 21st line of the page and start printing in the 33rd column of the page.

To print the report trailer, the following is coded.

```

//DSRPT      JOB                               Gives the Jobname
//          EXEC  PGM=SYNCSORT                 Identifies the Program
//SYSOUT     DD   SYSOUT=*                     Assigns SyncSort Messages
//*          to I/O Device
//SORTIN     DD   DSN=MRKTNG.SALES,DISP=SHR    Defines Input Data Set
//SORTOUT    DD   SYSOUT=*                     Defines Output Data Set
//SORTWK01   DD   SPACE=(CYL,5),UNIT=SYSDA    Defines Intermediate Storage
//SYSIN      DD   *
           SORT FIELDS=(1,15,CH,A)           Sorts Records
           .
           .
           .
OUTFIL      .
           .
           TRAILER1=(20/,                       Generates Report Trailer
                   33:'FEBRUARY SALES FIGURES INCLUDE RESIDUAL 1992',
                   'SALES NOT PREVIOUSLY RECORDED')

```

Figure 129. JCL and Required Control Statements

Figure 123 shows the trailer that is generated by the above TRAILER1 parameter.

```
FEBRUARY SALES FIGURES INCLUDE RESIDUAL 1992 SALES NOT PREVIOUSLY RECORDED
```

Figure 130. Sample TRAILER1

Explanation: The TRAILER1 parameter produces a report trailer or summary that constitutes the final page of a report. Unless otherwise specified, it begins on the first line of the page. The TRAILER1's initial number-slash (n/) entry, 20/, directs the printer to forward space 20 blank lines before printing on the 21st line. The next entry, a number-colon (c:) entry, is used to center the literal string that follows it by having the string of characters begin printing in the appropriate column. It specifies column 33 as the beginning position for printing the literal string, 'FEBRUARY SALES FIGURES INCLUDE RESIDUAL 1992 SALES NOT PREVIOUSLY RECORDED'.

Writing a Page Trailer

Example: Marketing wants the last line on every page of its departmental-sales report to contain a note identifying the information as confidential. This line will begin printing in column one.

To print the page trailer, the following is coded.

```
//DSRPT    JOB                               Gives the Jobname
//          EXEC  PGM=SYNCSORT                Identifies the Program
//SYSOUT   DD    SYSOUT=*                    Assigns SyncSort Messages
//*                                               to I/O Device
//SORTIN   DD    DSN=MRKTNG.SALES,DISP=SHR    Defines Input Data Set
//SORTOUT  DD    SYSOUT=*                    Defines Output Data Set
//SORTWK01 DD    SPACE=(CYL,5),UNIT=SYSDA    Defines Intermediate Storage
//SYSIN    DD    *
          SORT FIELDS=(1,15,CH,A)           Sorts Records
          .
          .
          .
          OUTFIL .
          .
          .
          TRAILER2=(5*'','CONFIDENTIAL INFORMATION',
          5*'','CONFIDENTIAL INFORMATION',5*'')
          .
          .
          .
          Generates Page Trailer
```

Figure 131. JCL and Required Control Statements

Figure 132 shows the trailer that is generated by the above TRAILER2 parameter.

```
*****CONFIDENTIAL INFORMATION*****CONFIDENTIAL INFORMATION*****
```

Figure 132. Sample TRAILER3

Explanation: The TRAILER2 coded above provides a trailer that appears at the bottom of every logical page. The first entry, 5*' , a literal enclosed in single quotes (in this case an asterisk) and a repetition factor (5), specifies that 5 asterisks should be printed. Because no column was specified, the trailer begins in column one. The next entry, 'C O N F I D E N T I A L I N F O R M A T I O N ' , specifies that the literal string enclosed in the single quotes should directly follow the asterisks. Note that the literal string is printed exactly as it is coded within the quotation marks. That is, there is a blank between every letter and two blanks between each word. The trailer's other entries specify the printing of another five asterisks followed by the literal string 'C O N F I D E N T I A L I N F O R M A T I O N ' and finally another five asterisks.

Totaling and Subtotaling Data

Writing a summary or trailer for a report will sometimes involve providing totals for columns of figures. For example, you would probably want a trailer for an inventory report to contain the total number of items on hand. The OUTFIL statement allows you to write trailers that contain both totals and subtotals. Moreover, you can total data at the end of a report, at the end of a page, and also at the end of a section.

To generate total and subtotals, use the TOTAL and SUBTOTAL entries of OUTFIL's TRAILER parameters and subparameter. For details of syntax, see "TRAILER Parameters (Optional)" on page 2.73

Totaling Data at the End of a Report

Example: The departmental sales report's final page will be a summary containing both the total for the sales this period and the total for the sales to date. The trailer will begin on the 21st line of the page and each total will have an identifying label.

To print the report trailer, the following is coded.

```

//DSRPT   JOB                               Gives the Jobname
//        EXEC   PGM=SYNCSORT               Identifies the Program
//SYSOUT  DD    SYSOUT=*                     Assigns SyncSort Messages
//*                               to I/O Device
//SORTIN  DD    DSN=MRKTNG.SALES,           Defines Input Data Set
                               DISP=SHR
//SORTOUT DD    SYSOUT=*                     Defines Output Data Set
//SORTWK01 DD   SPACE=(CYL,5),              Defines Intermediate Storage
                               UNIT=SYSDA
//SYSIN   DD    *
SORT  FIELDS=(1,15,CH,A)                    Sorts Records
.
.
.
OUTFIL.
.
.
TRAILER1=(20/,                               Generates Report Trailer with Totals
          40:'SALES THIS PERIOD:',
          59:TOT=(24,4,PD,EDIT=($II,IIT.TT)),
          73:'SALES TO DATE:',
          88:TOT=(28,4,PD,EDIT=($II,IIT.TT)))

```

Figure 133. JCL and Required Control Statements

Figure 134 shows the trailer that is generated by the above TRAILER1 parameter.

```

SALES THIS PERIOD: $35,807.85   SALES TO DATE: $62,305.25

```

Figure 134. Sample TRAILER1

Explanation: The TRAILER1 parameter produces a report trailer or summary that constitutes the final page of a report. Unless otherwise specified, it begins on the first line of the page. This TRAILER1's initial number-slash(n/) entry, 20/, directs the printer to forward space 20 blank lines before printing. The next entry, a number-colon (c:) entry, is used to center the literal string that follows it by having the string of characters begin printing in the appropriate column. It specifies column 40 as the beginning position for the literal string 'SALES THIS PERIOD:' that labels the numeric data following it. This TRAILER's other number-colon plus literal-string entry functions the same way.

The two TOT entries, TOT=(...), generate the trailer's totals. These entries specify the numeric data used and its format. Thus the four bytes of packed-decimal data that begin in byte 24 (24,4,PD) and the four bytes that begin in byte 28 (28,4,PD) of the input record are converted to printable format. This data is then edited by the EDIT pattern (\$II,IIT.TT), which suppresses the printing of leading zeros and inserts a floating dollar sign as well as a necessary comma and decimal point. The pattern uses an I to indicate those zeros in the total that should not be printed and a T to indicate those that should.

Note: Be sure to code all the necessary parentheses when using the TOTAL and EDIT entries.

Subtotaling Data at the End of a Page

Example: The page trailer for a report listing invoices is to contain the totals for the Amount Paid and the Balance Due fields of the invoice records printed up to and including that page. These totals will appear directly below the columns of figures and be separated from them by strings of hyphens. An identifying label, TOTALS:, will appear on the same line as the totals and will begin in column 40.

To generate the trailer, the following is coded.

```
//INVLST  JOB                               Gives the Jobname
//        EXEC  PGM=SYNCSORT                 Identifies the Program
//SYSOUT  DD    SYSOUT=*                     Assigns SyncSort Messages
//*                                             to I/O Device
//SORTIN  DD    DSN=INVOICE,DISP=SHR        Defines Input Data Set
//SORTOUT DD    SYSOUT=*                     Defines Output Data Set
//SORTWK01 DD   SPACE=(CYL,5),UNIT=SYSDA   Defines Intermediate Storage
//SYSIN   DD    *
          SORT  FIELDS=(9,23,A,36,2,A,32,4,A), Sorts Records
          FORMAT=CH
          .
          .
          .
          OUTFIL.
          .
          .
          TRAILER2=(65:10'- ', 86:10'- ',/, Generates Page Trailer
                  40:'TOTALS:',           with Running Totals
                  65:SUB=(46,4,PD,EDIT=($II,IIT.TT)),
                  86:SUB=(54,4,PD,EDIT=($II,IIT.TT)))
```

Figure 135. JCL and Required Control Statements

Figure 136 shows the trailer that is produced.

.
.
.
MERLINS TRUST CO	82124054	12/15/92	0.00	1,500.00
MEWER COLLEGE	83013324	1/17/92	0.00	1,500.00
NORTHEAST INDUST	83013303	1/17/92	200.00	200.00
PARK PLACE CORP	83022211	2/15/92	0.00	650.00
PATIO PRODUCTS	83022203	2/15/92	0.00	850.00
PINES ASSOCIATES	83022587	2/15/92	0.00	750.00
POLL DATA CORP	82124019	12/15/92	0.00	600.00
PRIESTLEY METALS	83022201	2/15/92	0.00	1,600.00
REGENCY TRUST CO	82124011	12/15/92	0.00	1,500.00
REPUBLIC DATA	83013306	1/17/92	0.00	1,100.00
RIBBIT TECHNOLOGIES	82124020	12/15/92	0.00	360.00
RICE FEATURES	82124015	12/15/92	750.00	750.00
RICE FEATURES	83013298	1/17/92	0.00	1,500.00
RICE FEATURES	83022198	2/15/92	0.00	1,500.00
ROBINS NEST CORP	83013353	1/17/92	0.00	900.00
SIDNEY COLLEGE	82124016	12/15/92	0.00	5,000.00
SIDNEY COLLEGE	83013297	1/17/92	0.00	2,500.00
			-----	-----
		TOTALS:	\$6,150.00	\$66,475.00

Figure 136. TRAILER2 with SUBTOTAL

Explanation: The above TRAILER2 provides for totaling the figures in the Amount Paid field (46,4,PD) and the Amount Due field (54,4,PD) on the invoice records. Because the SUB (SUBTOTAL) entry is specified, the totals that appear at the bottom of each page represent running totals, that is, the totals for all the records that have been printed up to and including that page. The TRAILER2 also generates the identifying label TOTALS: (40:'TOTALS:') and strings of hyphens at the bottoms of the columns to be totaled (65:10'-, 86:10'-).

The totaled data for each field is converted to printable format and, after being edited, begins printing in the columns specified with the two number colon entries (c:), 65: and 86:. The data is edited by the EDIT pattern, (\$II,IIT.TT), which suppresses the printing of leading zeros and inserts a floating dollar sign as well as the necessary comma and decimal point. The pattern uses an I to indicate the zeros in the total that should not be printed and a T to indicate those that should.

Totaling Data at the End of a Section

Example: The section trailer for an accounts receivable report sectioned by month is to contain the totals for the Amount Paid and the Balance Due columns of each section. These totals will appear directly below the columns of figures and be separated from them by strings of hyphens. An identifying label, TOTALS:, will appear on the same line as the totals and will begin in column 40.

To generate the trailer, the following is coded.

```

//ACTREC   JOB                               Gives the Jobname
//          EXEC   PGM=SYNCSORT              Identifies the Program
//SYSOUT   DD     SYSOUT=*                   Assigns SyncSort Messages
//*                                               to I/O Device
//SORTIN   DD     DSN=NEW.INV,DISP=SHR       Defines Input Data Set
//SORTOUT  DD     SYSOUT=*                   Defines Output Data Set
//SORTWK01 DD     SPACE=(CYL,5),UNIT=SYSDA  Defines Intermediate Storage
//SYSIN    DD     *
          SORT FIELDS=(9,23,A,36,2,A,32,4,A), Sorts Records
          FORMAT=CH
          .
          .
          .
          OUTFIL.
          .
          .
          SECTIONS=(32,4,SKIP=3L,           Generates Section Breaks
          TRAILER3=(65:10'-' ,86:10'-' ,/,   Generates Section Trailer
          40:'TOTALS:',                       with Totals
          65:TOT=(46,4,PD,EDIT=($II,IIT.TT)),
          86:TOT=(54,4,PD,EDIT=($II,IIT.TT)))

```

Figure 137. JCL and Required Control Statements

Figure 138 shows the section trailer, with totals, that is produced.

.
.
.
WINIFRED INDUST	82124013	12/15/91	300.00	350.00
			-----	-----
	TOTALS:		\$2,600.00	\$19,770.00
ARLINE FRAGRANCES	83013304	1/17/92	0.00	7,500.00
CHARACTER DATA	83013343	1/17/92	0.00	1,100.00
COUNTRY INDUSTRIAL	83013557	1/17/92	0.00	950.00
DUNHAM INDUST INC	83013302	1/17/92	0.00	850.00
ECHO LABS INC	83013300	1/17/92	0.00	550.00
ESS SECURITIES	83013311	1/17/92	0.00	550.00
EVERMORE INDUST	83013556	1/17/92	2,000.00	3,000.00
GOODEY FOODS	83013356	1/17/92	0.00	600.00
GROSS BOOKS CO	83013264	1/17/92	0.00	2,500.00
HARVEY MOTORS CO	83013301	1/17/92	2,000.00	3,000.00
KALABRA CORPORATION	83013555	1/17/92	0.00	1,500.00
MEWER COLLEGE	83013324	1/17/92	0.00	1,500.00
NORTHEAST INDUST	83013303	1/17/92	200.00	200.00
REPUBLIC DATA	83013306	1/17/92	0.00	1,100.00
RICE FEATURES	83013298	1/17/92	0.00	1,500.00
ROBINS NEST CORP	83013353	1/17/92	0.00	900.00
SIDNEY COLLEGE	83013297	1/17/92	0.00	2,500.00
SOUTHWEST INDUST	83013503	1/17/92	200.00	200.00
SPENSERS INDUST	83013989	1/17/92	0.00	650.00
UNITED INTERESTS INC	83013309	1/17/92	0.00	1,500.00
WINIFRED INDUST	83013299	1/17/92	0.00	650.00
			-----	-----
	TOTALS:		\$4,400.00	\$32,800.00

Figure 138. TRAILER3 with TOTAL

Explanation: In addition to generating strings of hyphens at the bottom of the columns to be totaled (65:10'-',86:10'-') and the identifying label TOTALS: on the line below (40:"TOTALS:'), the TRAILER3 provides for totaling the figures in the Amount Paid field (46,4,PD) and the Amount Due field (54,4,PD) on the invoice records. Note that because the TOT (TOTAL) entry is specified, the totals that appear at the end of each section represent that totals *only* for the records that are included in that section.

The totaled data for each field is converted to printable format and, after being edited, begins printing in the columns specified with the two number colon entries (c:), 65: and 86:. The data is edited by the EDIT pattern, (\$II,IIT.TT), which suppresses the printing of leading zeros and inserts a floating dollar sign as well as the necessary comma and decimal point. The pattern uses an I to indicate the zeros in the total that should not be printed and a T to indicate those that should.

Obtaining Maximum, Minimum and Average Data

A report may need to include maximum, minimum, and average data. The parameters provided for this type of reporting are MIN, SUBMIN, MAX, SUBMAX, AVG and SUBAVG. The syntax is the same as for TOTAL and SUBTOTAL. See “Totaling and Subtotaling Data” on page 3.41 and “TRAILER Parameters (Optional)” on page 2.73.

Printing Maximum, Minimum and Average Data in Section Trailers

Example: The section trailers for an accounts receivable report sectioned by data group (AAA, BBB, etc.) are to contain six edited numeric values for a 6-byte field that begins at byte 8 (8,6). The values to be printed are the following:

- The minimum data value up to that point in the report (SUBMIN)
- The minimum data value in the section (MIN)
- The maximum data value up to that point in the report (SUBMAX)
- The maximum data value in the section (MAX)
- The average data value up to that point in the report (SUBAVG)
- The average data value in the section (AVG)

Each value will be preceded, on the same line, by appropriate identifying text. Two columns of data will be printed.

To print the report, the following is coded:

```
SORT FIELDS=(1,3,CH,A,5,2,CH,A)   SORT DATA BY GROUP AND SECTION
OUTFIL FILES=(OUT),
SECTIONS=(1,3,SKIP=3L,
HEADER3=(3:'GROUP',2X,1,3,/,16:'SECTION',6X,'VALUE',/),
TRAILER3=(//,4:'MINIMUM VALUE TO THIS POINT= ',
          35:SUBMIN=(8,6,ZD,M2),/,
          4:'MINIMUM VALUE FOR THIS GROUP= ',
          35:MIN=(8,6,ZD,M2),//,
          4:'MAXIMUM VALUE TO THIS POINT= ',
          35:SUBMAX=(8,6,ZD,M2),/,
          4:'MAXIMUM VALUE FOR THIS GROUP= ',
          35:MAX=(8,6,ZD,M2),//,
          4:'AVERAGE VALUE TO THIS POINT= ',
          35:SUBAVG=(8,6,ZD,M2)/,
          4:'AVERAGE VALUE FOR THIS GROUP= ',
          35:AVG=(8,6,ZD,M2)),
OUTREC=(18:5,2,26:8,6,ZD,M2,80:1X)
```

The following shows two sections from the report, with the resulting values for subminimums, minimums, submaximums, maximums, subaverages and averages:

GROUP AAA

SECTION	VALUE
01	38.42
01	923.12
01	8,756.33
02	9,723.63
02	67.43
02	175.66
03	645.83
03	673.41
03	23.71

MINIMUM VALUE TO THIS POINT= 23.71
MINIMUM VALUE FOR THIS GROUP= 23.71

MAXIMUM VALUE TO THIS POINT= 9,723.63
MAXIMUM VALUE FOR THIS GROUP= 9,723.63

AVERAGE VALUE TO THIS POINT= 2,336.39
AVERAGE VALUE FOR THIS GROUP= 2,336.39

GROUP BBB

SECTION	VALUE
01	0.01
01	456.11
01	874.01
02	4,354.00
02	2,583.54
02	3.57
03	809.01
03	934.53
03	853.21

MINIMUM VALUE TO THIS POINT= 0.01
MINIMUM VALUE FOR THIS GROUP= 0.01

MAXIMUM VALUE TO THIS POINT= 9,723.63
MAXIMUM VALUE FOR THIS GROUP= 4,354.00

AVERAGE VALUE TO THIS POINT= 1,771.97
AVERAGE VALUE FOR THIS GROUP= 1,207.55

Explanation: The SECTION parameter generates a section break on field 1,3, which identifies data groups (AAA, BBB, etc.). The HEADER3 parameter defines section headers that print the label "GROUP" followed by the data group identifier. HEADER3 also defines two column headings: "SECTION," which identifies the column containing section numbers, and "VALUE," which identifies the columns containing the numeric data.

The TRAILER3 subparameters are SUBMIN, MIN, SUBMAX, MAX, SUBAVG and AVG. They specify the six values to appear in the section trailer. The values are all derived from the same field (8,6) and are suitably edited with mask M2 (8,6,ZD,M2).

The OUTREC parameter places the two data fields (5,2 and 8,6) in the report and edits the 8,6 field in the same way as for the six values in the section trailer (8,6,ZD,M2). The blank space placed at position 80 (80:1X) ensures that the output record is long enough to contain the header records.

Counting Data Records

Trailers in a report will sometimes require you to obtain a record count or a count for a particular type of item in a specific part of a report. The OUTFIL statement allows you to write trailers that contain such a count as well as cumulative, or running, counts of records. Moreover, you can obtain these counts at the end of a report, at the end of a page, and at the end of a section.

To generate these counts, use the COUNT and SUBCOUNT subparameters (or COUNT15 and SUBCOUNT15). These subparameters can be used in conjunction with all other TRAILER entries. For syntax of COUNT and SUBCOUNT (as well as COUNT15 and SUBCOUNT15), see “TRAILER Parameters (Optional)” on page 2.73.

Obtaining a Count of Data Records

Example: Marketing wants a count of the total number of customers with outstanding payments included in the summary of its outstanding invoices report.

To get this record count and print it as part of the report summary, the following is coded.

```

//INVLST  JOB                               Gives the Jobname
//          EXEC  PGM=SYNCSORT              Identifies the Program
//SYSOUT  DD    SYSOUT=*                    Assigns SyncSort Messages
//*                                              to I/O Device
//SORTIN  DD    DSN=INVOICE,DISP=SHR       Defines Input Data Set
//SORTOUT DD    SYSOUT=*                    Defines Output Data Set
//SORTWK01 DD   SPACE=(CYL,5),UNIT=SYSDA   Defines Intermediate Storage
//SYSIN   DD    *
          SORT  FIELDS=(1,23,CH,A)         Sorts Records
          .
          .
          .
          OUTFIL.
          .
          .
          TRAILER1=(20/,                    Generates Report Summary
                   40:'NUMBER OF CUSTOMERS WITH OUTSTANDING PAYMENTS:',
                   COUNT)

```

Figure 139. JCL and Required Control Statements

Figure 140 shows the trailer containing the record count.

```

NUMBER OF CUSTOMERS WITH OUTSTANDING PAYMENTS:    52

```

Figure 140. Report Trailer Containing Record Count

Explanation: Since each record in the report represents an individual customer, coding the COUNT entry in the TRAILER1 will provide the total number of customers with outstanding payments. This TRAILER1 produces a report trailer, or summary, that constitutes the final page of a report. It will print on the 21st line of the page (20/) and begin printing the literal string 'NUMBER OF CUSTOMERS WITH OUTSTANDING PAYMENTS: ' in column 40.

Obtaining a Cumulative (Running) Count of Data Records

Example: For an outstanding invoices report sectioned by month, marketing wants a cumulative, or running, count of invoices to date at the end of each section as well as a total count of each month's invoices included as section trailers.

To generate these record counts, the following is coded.

```

//INVLST   JOB                               Gives the Jobname
//          EXEC   PGM=SYNCSORT              Identifies the Program
//SYSOUT   DD     SYSOUT=*                   Assigns SyncSort Messages
//*                                               to I/O Device
//SORTIN   DD     DSN=INVOICE,DISP=SHR       Defines Input Data Set
//SORTOUT  DD     SYSOUT=*                   Defines Output Data Set
//SORTWK01 DD     SPACE=(CYL,5),UNIT=SYSDA   Defines Intermediate Storage
//SYSIN    *
          SORT FIELDS=(28,2,ZD,A,           Sorts Records
                        24,2,ZD,A,
                        1,23,ZD,A)
.
.
.
          OUTFIL.
.
.
          SECTIONS=(24,6,SKIP=1L,           Generates Sections with Record
          TRAILER3=(/
          Count & Cumulative Record Subcount
          95:'MONTH'S NUMBER OF INVOICES: ',COUNT,/,
          95:'NUMBER OF INVOICES TO DATE: ',SUBCOUNT))
,

```

Figure 141. JCL and Required Control Statements

Figure 142 shows the trailers containing the counts of records.

.	.	.	.
.	.	.	.
.	.	.	.
RIBBIT TECHNOLOGIES	2/15/91	360.00	21.60
RICE FEATURES	12/15/91	750.00	75.00
SIDNEY COLLEGE	12/15/91	5,000.00	300.00
SNAP FEATURES	12/15/91	750.00	75.00
WEBB BROS CORP	12/15/91	600.00	36.00
WELLINGTON IMPORTS	12/15/91	750.00	45.00
WINIFRED INDUST	12/15/91	350.00	26.00
			MONTH'S NUMBER OF INVOICES: 17
			NUMBER OF INVOICES TO DATE: 17
ARLINE FRAGRANCES	1/17/92	7,500.00	618.75
CHARACTER DATA	1/17/92	1,100.00	50.75
COUNTRY INDUSTRIAL	1/17/92	850.00	0.00
DUNHAM INDUST CO	1/17/92	850.00	0.00
ECHO LABS INC	1/17/92	550.00	22.00
ESS SECURITIES	1/17/92	550.00	22.00
EVERMORE INDUST	1/17/92	3,000.00	225.00
GOODEY FOODS	1/17/92	600.00	30.00
GROSS BOOKS CO	1/17/92	2,500.00	150.00
HARVEY MOTORS CO	1/17/92	3,000.00	225.00
KALABRA CORP	1/17/92	1,500.00	90.00
MEWER COLLEGE	1/17/92	1,500.00	75.00
NORTHEAST INDUST	1/17/92	200.00	20.00
REPUBLIC DATA	1/17/92	1,100.00	90.75
RICE FEATURES	1/17/92	1,500.00	75.00
ROBINS NEST CORP	1/17/92	900.00	54.00
SIDNEY COLLEGE	1/17/92	2,500.00	150.00
SOUTHWEST INDUST	1/17/92	200.00	20.00
SPENSERS INDUST	1/17/92	650.00	26.00
UNITED INTERESTS	1/17/92	1,500.00	90.00
WINIFRED INDUST	1/17/92	650.00	26.00
			MONTH'S NUMBER OF INVOICES: 21
			NUMBER OF INVOICES TO DATE: 38
BALTIC AVENUE CORP	2/15/92	650.00	29.25
BATHO PRODUCTS	2/15/92	850.00	51.00
CARRINGTON OIL	2/15/92	1,600.00	64.00
CDR TRUST INC	2/15/92	1,500.00	75.00
ECHO LABS INC	2/15/92	550.00	22.00
ESS SECURITIES	2/15/92	550.00	22.00
FASTEROOT EQUIP	2/15/92	1,700.00	76.50
FEDERAL FABRICS	2/15/92	1,750.00	70.00
.	.	.	.
.	.	.	.
.	.	.	.

Figure 142. TRAILER3 Containing Record Counts and Cumulative Record Counts

Explanation: The trailer's first / entry causes the printer to leave one blank line after the data records and before printing the trailer. The second / entry indicates the end of the trailer's first line. The identical number-colon entries (95:) set the starting positions of the literal strings that follow them: 'MONTH' 'S NUMBER OF INVOICES: ' and 'NUMBER OF INVOICES TO DATE: '.(Note that the apostrophe in MONTH'S is doubled because a single apostrophe would signal the end of a literal string.) Finally, because each data record in this report represents an invoice, the TRAILER3's COUNT entry generates a count of each month's invoices and the SUBCOUNT entry generates a cumulative, or running, count of the invoices. The leading zeros in these 8-byte fields are suppressed.

Creating Multiple Output Files

Data centers often use the same masterfile for different purposes. Assume, for example, that you wanted to produce two reports using a masterfile of cash-receipt records. One report was to present the total cash receipts for the current month; the second, for the year to date. This would typically entail running a separate sort for each report. SortWriter's multiple-output feature, however, enables you to produce both reports with a single pass of the sort. In addition, you can specify the same or different devices to receive the separate output files.

Note: All the output files will be sequenced in the same way, as specified on the SORT or MERGE statement. If you need to sort the output files differently, you should use PipeSort, a Syncsort product that works with SyncSort for z/OS to reduce total elapsed time by generating multiple, differently sequenced output files from a single read of the input data.

To generate multiple output files, code the OUTFIL statement. For syntax of the OUTFIL control statement, see "OUTFIL Control Statement" on page 2.59.

Generating Several Output Files with Different Information

Example: Marketing wants three output files of customer records. The first will contain a list of U.S. and European customers. The second will contain a list of U.S. customers only, and the third will contain a list of European customers only.

To generate the three separate files, the following is coded.

```

//CUSTRCD JOB                               Gives the Jobname
//      EXEC PGM=SYNCSORT                    Identifies the Program
//SYSOUT DD SYSOUT=A                        Assigns SyncSort Messages
//*                                          to I/O Device
//SORTIN DD DSN=SALES.RECORDS,              Defines Input Data Set
//      VOL=SER=DISK1,
//      DISP=SHR
//SORTOF1 DD DSN=SORTED.CUSTM.RECORDS,     Defines First Output Data
//      UNIT=TAPE,VOL=SER=112231,         Set Containing All
//*                                          Customer Records
//      DISP=(NEW,KEEP)
//SORTOF2 DD DSN=SORTED.DCUSTM.RECORDS,    Defines Second Output Data
//      UNIT=TAPE,VOL=SER=112232,         Set Containing Domestic
//      DISP=(NEW,KEEP)                   Customer Records Only
//SORTOF3 DD DSN=SORTED.ECUSTM.RECORDS,    Defines Third Output Data
//*                                          Set Containing European
//      UNIT=TAPE,VOL=SER=112233,         Customers Only
//      DISP=(NEW,KEEP)
//SORTWK01 DD SPACE=(CYL,20),UNIT=SYSDA    Defines Intermediate Storage
//SORTWK02 DD SPACE=(CYL,20),UNIT=SYSDA    Defines Intermediate Storage
//SORTWK03 DD SPACE=(CYL,20),UNIT=SYSDA    Defines Intermediate Storage
//SYSIN DD *
      SORT FIELDS=(10,15,CH,A)             Sorts Records
      OUTFIL FILES=1,                      OUTFIL Statement for SORTOF1
          INCLUDE=ALL                       Including All Records
      OUTFIL FILES=2,                      OUTFIL Statement for SORTOF2
          INCLUDE=(67,3,CH,EQ,C'USA')       Including USA Records
      OUTFIL FILES=3,                      OUTFIL Statement for SORTOF3
          INCLUDE=(67,3,CH,EQ,C'EUR')       Including Eur. Records

```

Figure 143. JCL and Required Control Statements

Explanation: Creating the three requested output files requires coding three SORTOFxDD statements in the JCL: SORTOF1, SORTOF2, and SORTOF3 as well as three OUTFIL statements. Each of the OUTFIL statements is connected by a FILES parameter to one of the output files defined in the JCL. Specifying 1 on the FILES parameter connects its OUTFIL statement with the output file defined by the SORTOF1 DD statement in the JCL. Likewise, specifying 2 connects its OUTFIL statement with the output file defined by SORTOF2, and so on. The first output file will contain all the records from the input file (INCLUDE=ALL). The second output file will include only those records that contain the character string 'USA' beginning in byte 67, (INCLUDE=(67,3,CH,EQ,C'USA')), which indicates that these records are for USA customers. And similarly, the third output file will include only those records that contain the character string 'EUR' beginning in byte 67, which indicates that these records are for European customers.

Writing Identical Output Files to Different Devices

Example: Personnel wants a printed copy of its updated masterfile as well as copies on disk and on tape.

To generate these three copies of the same file on different devices, the following is coded.

```
//MULTOUT JOB                               Gives the Jobname
//          EXEC PGM=SYNCSORT                Identifies the Program
//SYSOUT   DD   SYSOUT=*                     Assigns SyncSort Messages
//*                                               to I/O Device
//SORTIN   DD   DSN=PERSNL.RECORDS,         Defines Input Data Set
//          VOL=SER=DISK1,
//          DISP=SHR
//SORTOFPR DD   SYSOUT=*                     Defines Printed Output
//*                                               Data Set
//SORTOFTP DD   DSN=PERSNL.RECORDS.TAPE,    Defines Tape Output Data Set
//          UNIT=TAPE,VOL=SER=112233,
//          DISP=(NEW,KEEP)
//SORTOFDS DD   DSN=PERSNL.RECORDS.DISK,    Defines Disk Output Data Set
//          UNIT=DISK1,DISP=(NEW,KEEP),
//          SPACE=(CYL,60)
//SORTWK01 DD   SPACE=(CYL,20),UNIT=SYSDA   Defines Intermediate Storage
//SORTWK02 DD   SPACE=(CYL,20),UNIT=SYSDA   Defines Intermediate Storage
//SORTWK03 DD   SPACE=(CYL,20),UNIT=SYSDA   Defines Intermediate Storage
//SYSIN    DD   *
           SORT  FIELDS=(1,40,CH,A)         Sorts Records
           OUTFIL FILES=(PR,TP,DS)         Creates Multiple Output
```

Figure 144. JCL and Required Control Statements

Explanation: Creating the three copies of the updated masterfile requires coding only one OUTFIL statement with a FILES parameter. The FILES parameter instructs SyncSort to look for multiple output files defined in the JCL and to send its output to the devices specified in the SORTOFxx statements. Thus, the output that has been sorted as specified on the SORT statement (1,40,CH,A) will be sent to the printer specified in the SORTOFPR statement, to the tape volume specified in the SORTOFTP statement, and to the disk data set specified in the SORTOFDS statement.

Chapter 4. JCL and Sample JCL/Control Statement Streams

SyncSort's job control statements follow the standard operating system conventions described in the z/OS job control language manuals. Each program application therefore requires a JOB statement, an EXEC statement, and a DD (data definition) statement for every data set used. (The single exception to this is the dynamic allocation of work files via DYNALLOC or DYNATAPE.) The inclusion and coding requirements of particular job control statements depend on such factors as whether SyncSort is program-invoked or initiated directly, whether any exits are coded, and, of course, whether the sorting technique requested is Disk Sort, MAXSORT, PARASORT or Tape Sort.

All aspects of program initiation which are specific to the sort/merge (such as the dedicated DD names SORTIN and SORTOUT) are documented in this chapter. For complete coding instructions, refer to a z/OS MVS JCL reference manual.

The following table summarizes Disk Sort's DD statement requirements.

Disk Sort DD Statements	
//STEPLIB DD //JOLIB DD	Instructs operating system to look for the sort program in a specified data set.
//SYSOUT DD	Message data set. Required unless all messages are routed to console.
//SORTIN DD	SORT input data set. Required unless there is an E15. Ignored if the invoking program supplies an inline E15 exit routine; optional if the MODS statement activates an E15 exit routine.
//SORTIN _{nn} DD //SORTIN _n DD	MERGE input data set. Required unless there is an E32.
//SORTOUT DD	Output data set. Required unless there is an E35. Ignored if the invoking program supplies an inline E35 exit routine; optional if the MODS statement activates an E35 exit routine.
//SORTOF _{xx} DD //SORTOF _x DD //fname	OUTFILE output data sets. One required for each FILES or FNAMES specification.
//SORTXSUM DD	Output data set of records eliminated by the SUM control statement. Required when the XSUM parameter is specified.
//SORTWK _{xxx} DD //SORTWK _n DD	Disk work area definition. Required unless incore sort, DYNALLOC, MERGE, COPY or restarting at a MAXSORT merge breakpoint.
//SYSIN DD	Control statement data set. Required unless the invoking program supplies the address of a 24-bit or a 31-bit extended parameter list.
//\$ORTPARM DD	Used to override PARM or control statement information.
//SORTCKPT DD	Checkpoint data set. Required for Checkpoint-Restart.
//SORTMODS DD	Required if user exits are in SYSIN.
//SYSLIN DD //SYSLMOD DD //SYSPRINT DD //SYSUT1 DD	Required if user exits are to be linkage-edited at execution time.
//ddname DD	Required for exits unless the exit is inline in LINKLIB/ JOBLIB/STEPLIB or in SYSIN.

EXEC Statement

The EXEC statement is required in order to indicate to the operating system that the job is a sort/merge application. For a Disk Sort, the format of the EXEC statement is as follows.

To use a sort cataloged procedure, omit PGM= and specify the appropriate procedure name.

<code>//stepname EXEC</code>	$\left\{ \begin{array}{l} \text{PGM=SYNCSORT} \\ \text{PGM=SORT} \\ \text{PGM=IERRCO00} \\ \text{PGM=IGHRCO00} \\ \text{PGM=ICEMAN} \end{array} \right\}$	<code>[,PARM='...']</code>
------------------------------	---	----------------------------

Figure 145. Disk Sort EXEC Statement Format

The PARM parameter may be used to pass the sort/merge program a variety of keyword parameters, modifying it to meet the needs of the individual application.

For MAXSORT, PARASORT, DB2 Query Support, and Tape Sort

The format of the EXEC statement varies with the sorting technique chosen. The MAXSORT and PARASORT PARM options are used to request the MAXSORT or PARASORT sorting technique. The DB2 PARM option is used to request the DB2 Query function. These PARMs are compatible with any of the PGM names for Disk Sort. A Tape Sort application, on the other hand, requires a PGM name of SORT, IERRCO00, IGHRCO00, or ICEMAN. When PGM=SYNCSORT is used, SORTWK must be assigned to disk. The set of available PARM options is also dependent on sorting technique. Refer to “Chapter 5. PARM Options” for a description of the available options.

Coding Conventions for DD Statements

The following table summarizes the standard coding conventions for DD statements as they relate to the sort/merge program. For more detailed information, refer to an z/OS job control language manual.

Parameter	Subparameter	Required?
DSNAME/DSN		To access a labeled data set (e.g., SORTIN, STEPLIB) or to keep or catalog the data set being created (e.g., SORTOUT, SORTOU00).
DCB	RECFM, LRECL, and BLKSIZE OPTCD and BUFOFF	DCB not required for disk or standard labeled tape input. To override the values in the data set label of an old data set; to override the values in the first SORTIN or SORTINnn file for a new data set. To indicate ASCII input and output.
UNIT		For an input file that is not cataloged or passed; for a new data set
SPACE		For a new DASD data set.
VOLUME/VOL		For an input file that is not cataloged or passed; for a DASD output data set to be cataloged or passed.
LABEL		To override (1,SL).
DISP		To override (NEW,DELETE).

STEPLIB/JOBLIB DD Statement

If SyncSort has been installed in a private user library or in a test library, a STEPLIB or JOBLIB DD statement is required. The sample DD statement below instructs the operating system to look for the sort in a partitioned data set named SYNCTEST.

```
//STEPLIB DD DSN=SYNCTEST,DISP=SHR
```

Figure 146. Sample STEPLIB DD Statement

SYSOUT DD Statement

This defines the data set for SyncSort messages.

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

Figure 147. Sample SYSOUT DD Statement

If the SYSOUT DD statement is omitted, any message routed to it will be diverted to the console. Omitting the SYSOUT DD statement and setting the MSG=SC PARM (critical messages to the console, all messages to the printer), for example, will result in all messages being sent to the console.

SORTIN DD Statement

The SORTIN DD statement defines the data set to be sorted or copied. (The input file for a merge application is defined by the SORTINnn DD statement.) It is required for all sorts except those where an E15 exit (COBOL Input Procedure) provides all the input records.

The SORTIN file must have physical sequential or extended sequential organization or be a member of a partitioned data set or PDSE. It may reside on any device supported by BSAM or VSAM and if it is a VSAM data set, may be key-sequenced, entry-sequenced or relative record. SORTIN data sets may also be BatchPipes/z/OS pipes or HFS data sets. DCB information need not be supplied for a disk or standard labeled tape file. Any of the information accessed from a standard label can be overridden by coding the appropriate DCB parameter in the JCL.

The maximum record lengths supported are 32,760 bytes for fixed-length records and 32,767 bytes for variable-length records.

By default SyncSort does not accept an uninitialized SORTIN data set and will terminate processing with a WER400A message. An uninitialized data set is one that has been newly created but never successfully closed. The UNINTDS PARM or installation option can be used to change SyncSort's default mode of processing to accept an uninitialized input data set and process it as an empty file. See "UNINTDS" on page 5.31.

In this example, the data set to be sorted/copied is named SALESIN. It resides on one reel

```
//SORTIN DD DSN=SALESIN,DISP=(OLD,KEEP),  
//          UNIT=TAPE,VOL=SER=123456
```

Figure 148. Sample SORTIN DD Statement

of tape whose volume serial number is 123456. SALESIN is the first data set on that tape and has a standard label.

To access a SORTIN data set that resides in hiperbatch use the HBSI PARM. For more information about HBSI see "Chapter 5. PARM Options".

Concatenating Input Data Sets

The SORTIN file may consist of concatenated data sets, up to the limit supported by the operating system.

SyncSort must determine one set of DCB characteristics to use for reading all data sets in the concatenation. The following rules apply to the DCB characteristics:

- When the first data set is fixed-length (RECFM=F, FB, FBS), all subsequent data sets must be fixed-length and have the same LRECL.
- When the first data set is variable-length (RECFM=V, VB, VS, VBS), all subsequent data sets must be variable-length.
- For variable-length data sets, the LRECL of the first data set is used except for the following situations:
 - The LRECL of a subsequent data set is used if that LRECL is the largest found and is available at sort initialization. An LRECL is available at initialization if it is specified on a SORTIN DD statement or exists in the label of a SORTIN disk data set.
 - A record length specified via the L1 value on the RECORD control statement is used if it is the largest record length found.
- For both fixed and variable-length data sets, the BLKSIZE of the first data set is used unless the BLKSIZE of a subsequent data set is the largest found and is available at sort initialization. A BLKSIZE is available at initialization if it is specified on a SORTIN DD statement or exists in the label of a SORTIN disk data set.

The following shows sample JCL for concatenating input data sets:

```
//SORTIN DD      DSN=AUGUST.SALES,DISP=(OLD,KEEP),
//              UNIT=3390,VOL=SER=DISK1,
//              DCB=(LRECL=200,RECFM=VB,BLKSIZE=7404)
//              DD      DSN=JUNE.SALES,DISP=(OLD,KEEP),
//              UNIT=TAPE,VOL=SER=123456,LABEL=(2,SL),
//              DCB=(LRECL=200,RECFM=V,BLKSIZE=8004)
//              DD      DSN=JULY.SALES,DISP=(OLD,KEEP),
//              UNIT=TAPE,VOL=SER=654321,LABEL=(1,SL),
//              DCB=(LRECL=100,RECFM=VB,BLKSIZE=8004)
```

Figure 149. Sample Disk and Tape Data Set Concatenation to SORTIN

In the preceding example, one disk and two tape data sets have been concatenated. Any one of these data sets could be presented first. Position is not dependent upon BLKSIZE or LRECL. If the LRECL or BLKSIZE cannot be determined at SORT initialization, the first data set must carry the largest LRECL or BLKSIZE of the concatenation. Typically the LRECL or BLKSIZE cannot be determined when the input consists of concatenated tape data sets and the JCL lacks a DCB specification.

Sorting Large Input Data Sets

The MAXSORT technique is recommended for sorting very large amounts of data when disk work space is limited. With this technique, SORTWK requirements are independent of SORTIN size; thus, regardless of the size of the file, it can be sorted by one sort program using disk work files. MAXSORT's breakpoint/restart capability breaks the overlarge sorting application into smaller individual sorts; high priority jobs can execute between these smaller sorts without forcing any data to be resorted. See "Chapter 9. MAXSORT".

Reducing Elapsed Time for SORTS with Multi-volume or Concatenated Tape SORTIN

The PARASORT technique can be used to improve elapsed time performance of sorts that use multi-volume or concatenated tape SORTIN data sets. See "Chapter 10. PARASORT" on page 10.1.

SORTINnn or SORTINn DD Statement

SORTINnn and SORTINn DD statements are used to define the input to a merge application. (Use the SORTIN DD statement to define the data set to be sorted or copied.) SORTINnn or SORTINn DD statements are required for all merge applications unless an E32 exit supplies the input data. SORTINnn and SORTINn data sets may be BatchPipes/z/ OS pipes or HFS data sets.

It is possible to merge up to 100 data sets. Each input data set is specified on a SORTINnn or SORTINn DD statement. The valid range for n is 0 through 9; for nn, 00 through 99. If both SORTINx and a SORTIN0x are specified, they are treated as duplicates and only the first definition is processed. Each file must receive a different number. Numbers may be skipped or used out of order. There are no restrictions as to which input files are to receive which numbers.

Each input data set must have the same RECFM, and the records in each file must be ready to be sorted in the desired sequence.

By default, SyncSort does not accept an uninitialized SORTINnn or SORTINn data set and will terminate processing with a WER400A message. An uninitialized data set is one that has been newly created, but never successfully closed. The UNINTDS PARM or installation option can be used to change SyncSort's default mode of processing to accept an uninitialized input data set and process it as an empty file. See "UNINTDS" on page 5.31.

```

//SORTIN17    DD    DSN=BRANCHA.FICA,VOL=SER=131313,
//
//            DISP=OLD,UNIT=3480
//SORTIN01    DD    DSN=BRANCHC.FICA,VOL=SER=242424,
//            DISP=OLD,UNIT=3390
//SORTIN24    DD    DSN=BRANCHB.FICA,VOL=SER=121212,
//            DISP=OLD,UNIT=3400-3,LABEL=(,NL),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)

```

Figure 150. Sample SORTINnn DD Statements (Merge)

In this example, the DCB information for the first two of the three files to be merged is supplied by the file labels. In order for the merge to execute, these files must have a RECFM of F or FB, as indicated by the third file's RECFM value.

SORTOUT, SORTOFxx, SORTOFx and SORTXSUM DD Statements

The SORTOUT, SORTOFxx, SORTOFx and SORTXSUM DD statements are used to define one or more output files. The FNAMES parameter of the OUTFIL control statement may also specify DD names of output files. All output is directed to SORTOUT unless an inline E35 exit (COBOL output procedure) assumes the full responsibility for output processing. Records eliminated by SUM processing will be written to the SORTXSUM DD statement if the XSUM option was selected on the SUM control statement. These output data sets may be directed to a BSAM or VSAM supported device, to BatchPipes/z/OS pipes or to HFS data sets.

```

//SORTOUT     DD    DSN=MASTER.OUT,UNIT=SYSDA,
//
//            DISP=(NEW,KEEP),SPACE=(TRK,10),
//            VOL=SER=DSK002
//SORTOF01     DD    DSN=REPORT.OUT,UNIT=SYSDA,
//            DISP=(NEW,KEEP),SPACE=(TRK,10),
//            VOL=SER=DSK002

```

Figure 151. Sample SORTOUT/SORTOFxx DD Statements

In the preceding example, the missing DCB parameters except BLKSIZE will default to those assigned to SORTIN or (for a merge application) to those assigned to the last SORTINnn in the JCL stream. The DCB BLKSIZE, if missing, will be determined via system-determined blocksize when it is active or from SORTIN if SORTOUT and SORTIN LRECLs are the same, otherwise SyncSort will select an appropriate BLKSIZE.

If a sort or a merge has an LRECL specified in the output DD JCL that is found to be smaller than the internally processed record length (determined from SORTIN, the LENGTH values of a RECORD statement, or an INREC statement), SyncSort processing

will be controlled by the SOTRN installation option or its run time override parameter TRUNC. (SYNCGENR applications are controlled by the SOTRNGN installation option.) If the parameter setting allows truncation, SyncSort will write the records to the output data set by truncating the records to the LRECL of that data set. The delivered default allows truncation. SyncSort will not truncate records after OUTREC processing. If the option disallows truncation, a WER462A error message will be issued.

If an application that is processing fixed-length data has an LRECL specified in the SORTOUT or SORTXSUM JCL that is found to be longer than the internally processed record length, SyncSort will normally pad the output records with binary zeros. See the discussion of the PAD parameter in chapter 5 for additional controls that can be applied to applications with both a SORTIN and a SORTOUT where the SORTOUT LRECL is longer than the SORTIN LRECL. This padding will be done for SORTXSUM and for SORTOUT when OUTFIL is not in use. It will not be done for any OUTFIL files. If the option disallows padding, a WER462A error message will be issued. The delivered default allows padding.

If RECFM is specified and the report writing features of the OUTFIL control statement are being used, the RECFM of the output file must include the 'A' subparameter, except when the REMOVECC parameter is in use.

For a COPY or MERGE, the output file must not be the same as any of the input files.

Secondary Allocation

If the automatic secondary allocation option was enabled at installation time, requesting secondary allocation on the output DD statements is not required. This feature automatically provides output space for each of the output files.

To place a SORTOUT data set into hiperbatch so that subsequent job steps can access it, use HBSO. For more information about HBSO see the *PARM Option* chapter in this manual.

SORTWKxx or SORTWKx DD Statement

For non-MAXSORT applications, up to 255 data sets may be specified for intermediate storage when sorting. (MAXSORT, which is recommended for large sorting applications, is limited to 32 SORTWK data sets.) Each work file carries a SORTWKxx or SORTWKx name. x can be any alphanumeric or national (\$, #, @) character. Each SORTWKxx or SORTWKx *must* be allocated on a single unit and a single volume.

Disk Sorts may feature any of the following devices: 3350, 3375, 3380, 3390, and 9345. When device types are mixed, each device is used to full capacity. Note that although SORTWK space can be allocated in blocks, tracks, or cylinders, allocating in cylinders will yield optimal performance. The CONTIG option of the SPACE parameter should be avoided since it may delay allocation and offers no performance advantage.

The SORTWKxx DD statement in the following example establishes a primary allocation of 20 cylinders of work space.

```
//SORTWK02 DD UNIT=3390,SPACE=(CYL,20)
```

Figure 152. Sample SORTWKxx DD Statement for Disk Sorts

Secondary Allocation

There is no need to specify RLSE and a secondary allocation value on the SORTWKxx DD statement at installations that have set these defaults at SyncSort installation time.

Are SORTWKxx DD Statements Necessary?

SORTWKxx DD statements are not used for merge or copy applications. They are not required for sorts executed using the DYNALLOC option. Provided neither DYNALLOC nor FIELDS=COPY is in effect, it will be necessary to include SORTWK data sets whenever any of these conditions holds:

- INCORE is set to OFF.
- An E14 or E16 is included.
- Checkpoint-Restart is specified.
- The criteria for an incore sort are not met. (See the discussion of incore sorts in “Chapter 13. Performance Considerations”.)
- SUM, OUTREC or OUTFIL is used.
- SORTOUT is a VSAM data set.

Note: Sort applications that use SUM, OUTREC, OUTFIL or VSAM SORTOUT and do not provide JCL SORTWORKs may have DYNALLOC automatically enabled. This will allow the completion of a sort that would have terminated for lack of required SORTWORK space.

Initiating Tape Sort

Tape Sorts use the following devices for intermediate storage: 2400 and 3400 series tape units with densities of 800, 1600 and 6250 BPI. Each reel of tape must be full-size (2400 feet long). Tape cartridges devices (3480, 3490, 3490E and 3590) may also be used.

When intermediate storage is on tape, from 3 to 32 data sets may be specified. Tape SORTWKxx files must begin with SORTWK01 and be numbered consecutively. When different device types and tape densities are mixed, the lowest density is used to calculate the

capacity of each SORTWK volume. The MAXSORT technique (supporting only disk SORTWK files) is strongly recommended for large sorts.

The xxxx in the UNIT parameter of the following example represents the installation-specific name used to define a tape device.

```
//SORTWK01 DD UNIT= { 2400 }  
                   { 3400 }  
                   { xxxx }
```

Figure 153. SORTWKxx DD Statement Format for Tape Sorts

For more information, see “Chapter 12. Tape Sort”.

SYSIN DD Statement

The data set defined by the SYSIN DD statement contains SyncSort control statements. The SYSIN DD statement is required in order to initiate the sort/merge through job control language.

```
//SYSIN DD *  
      SORT      FIELDS=( 5 , 3 , CH , A )  
      OMIT      COND=( 12 , 6 , PD , EQ , 0 )  
      END  
/*
```

Figure 154. Sample SYSIN DD Statement

\$ORTPARM DD Statement

The data set defined by the \$ORTPARM DD statement may contain PARM parameters and any of the sort control statements.

Parameters and control statements passed via the \$ORTPARM DD statement generally override all others passed, whether the sort/merge is called from a program or initiated through job control language.

The \$ORTPARM DD record format must be F or FB, and the record length must be 80 bytes. Labels are not allowed on \$ORTPARM card images. Leading blanks are not required on a PARM card image, but at least one leading blank must precede a sort control statement keyword.

The \$ORTPARM data sets must be formatted in accordance with the following rules:

- PARM specifications included in the \$ORTPARM data sets must be specified before any sort control statement specifications.
- PARMs must be specified without the keyword PARM= and without quotation marks.
- A comma in columns 2-70 of a PARM card image followed by a blank, or a comma alone in column 71, may be used to indicate that the next record is part of the current statement. However, if the PARM specification is present through column 71, a continuation character must be specified in column 72 to indicate continuation.
- Comments may be included on \$ORTPARM card images provided there is a blank between the last PARM specification and the comment. You may continue a comment by placing a continuation character in column 72 if there are no additional PARMs. In this case, the entire next card image will be considered a comment. If additional PARMs will follow the comment, you may continue that comment by coding an asterisk (*) in column 1 of the next card image.

Note: Refer to “Chapter 2. SyncSort Control Statements” for additional formatting requirements.

The following example of a \$ORTPARM data set illustrates the conventions for defining the \$ORTPARM data set.

```
//$ORTPARM DD *
  BMSG,STOPAFT=500,
  EQUALS
  SORT FIELDS=(1,8,PD,A)
```

Figure 155. Sample \$ORTPARM DD Statement

The \$ORTPARM data set in the previous example overrides the options set in the associated invoking program (or job control stream) to sort 500 records from the input file. These will be the first 500 records that meet whatever criteria have been set by the original application (which might include, for example, the INCLUDE/OMIT control statement). BMSG turns on the WERnnnB message set, so that the processing accorded these 500 records is fully documented. EQUALS preserves the order of equal-keyed records from input to output.

```

//$ORTPARM DD *
  BMSG,STOPAFT=500,
  EQUALS
  SUM  FIELDS=(12,4,30,8,38,8),
        FORMAT=PD
  SORT FIELDS=(1,8,PD,A)

```

Figure 156. Sample \$ORTPARM DD Statement

The preceding example illustrates how to include control statements more than 80 bytes long; continuation card images are indicated by a blank field following an operand-comma combination.

```

//SYSIN DD *
  OUTFIL FILES=(1,2,3),
  .
  .
  .
//$ORTPARM DD *
  OUTFIL FILES=(3,4,5),
  .
  .
  .

```

Figure 157. Sample \$ORTPARM DD Statement

In this example, the OUTFIL control statement in \$ORTPARM overrides the OUTFIL control statement in SYSIN for file 3, and adds OUTFIL specifications for files 4 and 5.

\$ORTPARM Processing for Century Window COBOL Applications

The \$ORTPARM DD facility is particularly useful for COBOL sorts requiring century window processing of year data with SyncSort's year data formats. The year data formats are not supported by COBOL. Therefore, when a data format specification needs to be changed for century window processing, it is necessary to override SORT control statements generated by COBOL. The override can be accomplished with a \$ORTPARM DD statement. The following example shows a \$ORTPARM DD used for this purpose.

```

//$ORTPARM DD *
  SORT FIELDS=(10,2,Y2Z,A),CENTWIN=1980

```

Figure 158. Sample \$ORTPARM DD Statement for Century Window Processing

In this example, the 2-digit year field (10,2) will have century window processing applied to it via the Y2Z year data format and the CENTWIN option.

As described in the previous section, multiple sort invocations by the same COBOL program would require multiple \$ORTPARM DD statements, each with the FREE=CLOSE parameter.

\$ORTPARM DD Processing for Multiple Sort Invocations

When SyncSort is to be invoked more than once in the same job step, you may need different \$ORTPARM DD control data sets for each invocation. For multiple control data sets, define each one in the JCL stream, in the desired order, as a disk data set (or partitioned data set member) with the FREE=CLOSE parameter added. FREE=CLOSE will cause the first sort \$ORTPARM data set to be dynamically deallocated by the first sort execution, and so forth for each sort execution. The following example shows sample JCL with two \$ORTPARM DD statements:

```
//$ORTPARM DD DSN=SORT.OPTIONS(SORT1),DISP=SHR,FREE=CLOSE WILL
//
//           BE USED BY FIRST SORT EXECUTION
//$ORTPARM DD DSN=SORT.OPTIONS(SORT2),DISP=SHR,FREE=CLOSE WILL
//
//           BE USED BY SECOND SORT EXECUTION
.
.
//$ORTPARM DD DSN=SORT.OPTIONS(SORTn),DISP=SHR,FREE=CLOSE WILL
//
//           BE USED BY THE nTH SORT EXECUTION
```

Figure 159. Sample Multiple \$ORTPARM DD Statements

Processing will proceed from top to bottom of this \$ORTPARM data set list. This sequence must be maintained in the JCL so that the multiple sorts can read the \$ORTPARM data sets in the correct order.

Multiple \$ORTPARM datasets are available only in a JES2 environment. JES3 does not support the specification of multiple DD statements for the same DDNAME.

The \$ORTPARM DD statement for Tape Sort may include only *one* 80-byte record, which in turn may only feature PARMs. \$ORTPARM cannot be used to override Tape Sort control statements.

SORTCKPT DD Statement

This DD statement is only used when the CKPT/CHKPT option is set on the SORT/MERGE control statement, requesting the Checkpoint-Restart feature. Refer to “Chapter 13. Performance Considerations” for an explanation of this feature.

For Exit Routines that Require Link-editing at Execution Time

The following DD statements are required whenever an exit routine is to be link-edited at execution time.

SORTMODS DD Statement

The partitioned data set defined must be large enough to contain all the exit routines entered in SYSIN. For exits not entered in SYSIN, it is necessary to supply DD statements defining the libraries in which the routines reside.

```
//SORTMODS DD    SPACE=(CYL,(2,,4)),UNIT=SYSDA
```

Figure 160. Sample SORTMODS DD Statement

SYSLIN DD Statement

The SYSLIN DD statement defines the temporary data set that will contain the linkage editor control statements created by SyncSort for the exit routine(s).

```
//SYSLIN DD    DSN=&&TEMP,UNIT=SYSSQ,SPACE=(TRK,1)
```

Figure 161. Sample SYSLIN DD Statement

SYSLMOD DD Statement

The SYSLMOD DD statement defines the temporary data set that will contain the link-edited exit module(s).

```
//SYSLMOD DD    DSN=&&TEMP2,UNIT=SYSDA,  
//              SPACE=(TRK,(10,5,2))
```

Figure 162. Sample SYSLMOD DD Statement

SYSPRINT DD Statement

The SYSPRINT DD statement defines the message data set for the link-editing of sort exits.

```
//SYSPRINT DD SYSOUT=A
```

Figure 163. Sample SYSPRINT DD Statement

SYSUT1 DD Statement

The SYSUT1 DD statement is used to define the temporary data set used as a work area when SyncSort link-edits an exit routine.

```
//SYSUT1 DD DSN=&&TEMP3,UNIT=SYSDA,  
// SPACE=(CYL,(5,5))
```

Figure 164. Sample SYSUT1 DD Statement

DD Statements for MAXSORT, PARASORT, DB2 Query Support, and Tape Sort

The MAXSORT technique is initiated by means of the MAXSORT PARM, and utilizes additional MAXSORT DD statements (SORTBKPT, SORTOU00, SORTOU_{nn}) and PARMs. With MAXSORT, SORTWK files must be allocated to disk devices. This technique is strongly recommended for very large sorting applications in a limited disk work space environment.

The PARASORT technique is initiated by means of the PARASORT PARM and utilizes additional PARASORT DD statements (SORTPAR1, SORTPAR2, SORTPAR3, SORTPAR4). PARASORT requires disk SORTWK devices. This technique can improve the elapsed time of sorting applications that have multi-volume tape SORTIN data sets.

The DB2 Query Support technique is initiated by means of the DB2 Query Support PARM and utilizes the DB2 Query Support DD statement SORTDBIN. This technique allows DB2 data to be passed directly into a SORT or COPY operation, without the use of setup steps or the need for user-written E15 exits.

Tape Sort is initiated by assigning tape work devices. The use of Tape Sort constrains the set of PARMs available to the sort, requires a SORTLIB DD statement, and restricts the coding of the \$ORTPARM and SORTWK_{xx} statements.

For detailed descriptions of these techniques refer to “Chapter 9. MAXSORT”, “Chapter 10. PARASORT”, and “Chapter 12. Tape Sort”.

Sample JCL/Control Statement Streams

The sample JCL/control statement streams in this section illustrate how to specify sort, merge and copy applications with and without exit routines. An example illustrating multiple output is also included. Refer to “Chapter 3. How to Use SyncSort’s Data Utility Features” for comprehensive examples illustrating the data utility and report writing features. Examples of how to invoke SyncSort from a program, COBOL exit routines, MAXSORTs, PARASORTs and Tape Sorts are provided in the appropriate chapters.

Sorts without Exit Routines

Example 1

//SORTOMIT	JOB		1
//SORT1	EXEC	PGM=SYNCSORT, PARM='STOPAFT=1000'	2
//STEPLIB	DD	DSN=SORT.RESI.DENCE, DISP=SHR	3
//SYSOUT	DD	SYSOUT=A	4
//SORTIN	DD	DSN=INPUT, UNIT=3490,	5
//		VOL=SER=012345, DISP=(OLD, KEEP),	
//		DCB=(LRECL=100, RECFM=FB,	
//		BLKSIZE=32700), LABEL=(1, SL)	
//SORTOUT	DD	DSN=OUTPUT, VOL=SER=543210,	6
//		UNIT=3490, DISP=(NEW, KEEP),	
//		DCB=(LRECL=100, RECFM=FB,	
//		BLKSIZE=0), LABEL=(1, SL)	
//SORTWK01	DD	SPACE=(CYL,(20)), UNIT=SYSDA	7
//SORTWK02	DD	SPACE=(CYL,(20)), UNIT=SYSDA	
//SORTWK03	DD	SPACE=(CYL,(20)), UNIT=SYSDA	
//SORTWK04	DD	SPACE=(CYL,(20)), UNIT=SYSDA	
//SORTWK05	DD	SPACE=(CYL,(20)), UNIT=SYSDA	
//SYSIN	DD	*	8
	SORT	FIELDS=(1,8,CH,A)	9
	OMIT	COND=(1,8,CH,EQ,C'JOHN DOE')	10
	END		11
/*			12

Figure 165. Sample JCL/Control Stream (1)

1. The JOB statement gives SORTOMIT as the jobname.
2. The EXEC statement identifies SYNCSORT as the program to be executed. The STOPAFT PARM instructs SyncSort to terminate after sorting 1,000 records.
3. The STEPLIB DD statement instructs the system to look for SyncSort in the library named SORT.RESI.DENCE. The DISP indicates that this library may be shared.

4. The SYSOUT DD statement assigns the SyncSort messages to the output device associated with SYSOUT class A.
5. The SORTIN DD statement gives INPUT as the input data set name, specifies a 3490 tape unit with the volume serial number 012345. The data set is already in existence.

The DCB parameter shows an LRECL of 100 bytes, a fixed blocked RECFM, and a 32700-byte BLKSIZE. The LABEL parameter shows that INPUT is the first data set on the tape, and that it has a standard label.

6. The SORTOUT DD statement gives OUTPUT as the output data set name, and specifies a 3490 tape unit with the volume serial number 543210. The data set is not in existence yet.

The DCB parameter for SORTOUT specifies the same LRECL and RECFM as SORTIN. The BLKSIZE will be selected by System Determined BLKSIZE (SDB) if active or by SyncSort if SDB is not active.

7. The five SORTWKxx DD statements reserve space on direct access devices for intermediate storage. Twenty cylinders are allocated for each of the five SORTWKxx data sets.
8. The SYSIN DD * statement marks the beginning of the system input stream that includes the sort control statements.
9. The SORT control statement specifies that one control field will be sorted on. It begins on byte 1 of the record, is 8 bytes long, contains character data, and is to be sorted in ascending order.
10. The OMIT control statement eliminates any record with *JOHN DOE* in its first eight bytes (i.e., in the sort control key). *JOHN DOE* records are not sorted and are not included in the STOPAFT figure. The EXEC statement's STOPAFT PARM terminates the sort after 1,000 (non-*JOHN DOE*) records have been put into the proper sequence.
11. The END control statement marks the end of the control statements.
12. The delimiter statement marks the end of the SYSIN input stream.

Example 2

```
//SUMSORT      JOB                                1
//             EXEC  PGM=SYNCSORT , PARM= ' EQUALS ' 2
//STEPLIB      DD   DSN=SORT.RESI.DENCE , DISP=SHR   3
//SYSOUT       DD   SYSOUT=A                          4
//SORTIN       DD   DSN=FEB92 , EMPLOYEE.MASTER ,    5
//             UNIT=3490 , VOL=SER=135790 ,
//             DISP= (OLD , KEEP)
//             DD   DSN=FEB92.EMPLOYEE.UPDATE ,
//             UNIT=3490 , VOL=SER=999999 ,
//             DISP= (OLD , KEEP)
//SORTOUT      DD   DSN=MAR92.EMPLOYEE.MASTER ,      6
//             UNIT=3490 , VOL=SER=246809 ,
//             DISP= (NEW , KEEP)
//SORTWK01     DD   UNIT=SYSDA , SPACE= (CYL , 20)   7
//SYSIN        DD   *                                8
//             SORT  FIELDS= (1 , 9 , ZD , A , 10 , 2 , BI , A) 9
//             SUM   FIELDS= (12 , 4 , PD)              10
/*                                                    11
```

Figure 166. Sample JCL/Control Stream (2)

1. The JOB statement gives SUMSORT as the jobname.
2. The EXEC statement identifies SYNCSORT as the program to be executed. The EQUALS PARM interacts with the SUM control statement to preserve the first of a series of equal-keyed records.
3. The STEPLIB DD statement instructs the system to look for SyncSort in the library named SORT.RESI.DENCE. The DISP shows the library may be shared.
4. The SYSOUT DD statement assigns the SyncSort messages to the output device associated with class A.
5. The SORTIN DD statements define two concatenated data sets: FEB92.EMPLOYEE.MASTER and FEB92.EMPLOYEE.UPDATE. They are found on standard labeled 3490 tape units (volume serial numbers 135790 and 999999, respectively). These data sets are already in existence.
6. The SORTOUT DD statement gives MAR92.EMPLOYEE.MASTER as the output data set name and specifies a 3490 tape unit with the volume serial number 246809. The data set is not in existence yet.

The DCB RECFM and LRECL parameters for SORTOUT default to that of the first SORTIN file. The BLKSIZE will be selected by System Determined BLKSIZE (SDB) if active or by SyncSort if SDB is not active.

7. The SORTWK01 DD statement reserves space on a direct access device for intermediate storage. Twenty cylinders are allocated. Intermediate storage must be provided whenever the SUM control statement is used with a sort.
8. The SYSIN DD statement marks the beginning of the system input stream that includes the sort control statements.
9. The SORT control statement specifies that two control fields will be sorted on. The major control field begins on byte 1 of the record, is 9 bytes long, contains zoned decimal data, and is to be sorted in ascending numerical order. The second, less significant, control field is found in the next two bytes of the record (bytes 10 and 11), is in (unsigned) binary format, and is to be sorted in ascending order.
10. Whenever two records have equal control fields, the sort will attempt to summarize them. If the result of summing the packed decimal data found in the 4-byte field beginning at byte 12 can be contained in four bytes, one of the two records will be retained, the sum stored in bytes 12-15, and the other record will be deleted. The EQUALS PARM guarantees that the first of the two records will be preserved; thus, if a record from the FEB92.EMPLOYEE.MASTER file has the same key as one from the FEB92.EMPLOYEE.UPDATE file, it is the master record which is retained in the output file, containing their sum.
11. The delimiter statement marks the end of the SYSIN input stream.

Example 3

//SORTSKIP	JOB		1
//	EXEC	PGM=SYNCSORT	2
//\$ORTPARM	DD	*	3
		STOPAFT=100	
//STEPLIB	DD	DSN=SORT.RESI.DENCE,DISP=SHR	4
//SYSOUT	DD	SYSOUT=A	5
//SORTIN	DD	DSN=EXPORT.SHIPPING.VOL6,	6
//		UNIT=TAPE,VOL=SER=112233,	
//		DISP=(OLD,KEEP)	
//SORTOUT	DD	DSN=RECENT.MAJOR.EXPORTS,	7
//		UNIT=TAPE,VOL=SER=332211,	
//		DISP=(NEW,KEEP)	
//SORTWK01	DD	SPACE=(CYL,20),UNIT=SYSDA	8
//SORTWK02	DD	SPACE=(CYL,20),UNIT=SYSDA	
//SORTWK03	DD	SPACE=(CYL,20),UNIT=SYSDA	
//SYSIN	DD	*	9
		SORT FIELDS=(19,5,CH,A),	10
		EQUALS,SKIPREC=1000	
		INCLUDE COND=(37,4,BI,GE,X'50')	11
/*			12

Figure 167. Sample JCL/Control Stream (3)

1. The JOB statement gives SORTSKIP as the jobname.
2. The EXEC statement identifies SYNCSORT as the program to be executed.
3. The \$ORTPARM DD statement is used here to initiate a test run of the SORTSKIP job by supplying the STOPAFT PARM to SyncSort. It instructs SyncSort to terminate after sorting the first 100 of the records INCLUDE selects from the SKIPREC-edited input file.
4. The STEPLIB DD statement instructs the system to look for SyncSort in the library named SORT.RESI.DENCE. The DISP indicates that this library may be shared.
5. The SYSOUT DD statement assigns the SyncSort messages to the output device associated with SYSOUT class A.
6. The SORTIN DD statement gives EXPORT.SHIPPING.VOL6 as the input data set name. It is found on a standard labeled tape having the volume serial number 112233. This data set is already in existence.
7. The SORTOUT DD statement assigns the RECENT.MAJOR.EXPORTS data set name to the output file, and specifies a tape unit with the volume serial number 332211. This

data set is not yet in existence. The DCB RECFM and LRECL parameters for SORTOUT default to those of the first SORTIN file. The BLKSIZE will be selected by System Determined BLKSIZE (SDB) if active or by SyncSort if SDB is not active.

8. The three SORTWKxx DD statements reserve space on direct access devices for intermediate storage. Twenty cylinders are allocated for each SORTWK data set.
9. The SYSIN DD statement marks the beginning of the system input stream that includes the sort control statements.
10. The SORT control statement specifies that one control field will be sorted on. It begins on byte 19 of the record, is 5 bytes long, contains character data, and is to be sorted according to ascending order. The EQUALS parameter preserves the SORTIN order of records with identical data in these five bytes. The SKIPREC parameter eliminates the first 1,000 records of the SORTIN file from consideration; these records are eliminated before the INCLUDE statement takes effect.
11. The INCLUDE statement compares the 4 bytes beginning with byte 37 of the record to the hexadecimal literal, which will be padded on the right with binary zeros to the indicated (4 byte) length. The record is eliminated from the sort unless the binary data in that field is at least as great as the padded constant. The INCLUDE/OMIT statement takes effect after SKIPREC but before STOPAFT.
12. The delimiter statement marks the end of the SYSIN input stream.

A Merge without Exit Routines

Example 4

```
//EDITMERG      JOB                                1
//MERGE1        EXEC  PGM=SYNCSORT                2
//STEPLIB       DD   DSN=SORT.RESI.DENCE,DISP=SHR  3
//SYSOUT        DD   SYSOUT=A                    4
//SORTIN08      DD   DSN=SALES91,UNIT=TAPE,       5
//              VOL=SER=123456,DISP=(OLD,KEEP)
//SORTIN12      DD   DSN=SALES92,UNIT=TAPE,
//              VOL=SER=654321,DISP=(OLD,KEEP)
//SORTIN03      DD   DSN=SALES93,UNIT=3390,
//              VOL=SER=DISK11,DISP=SHR
//SORTOUT       DD   DSN=SALES.PATTERN,UNIT=3390,  6
//              VOL=SER=DISK08,DISP=(NEW,KEEP),
//              SPACE=(CYL,5),
//              DCB=(LRECL=20,RECFM=VB,
//              BLKSIZE=27980)
//SYSIN         DD   *                            7
                MERGE  FIELDS=(5,4,ZD,A)          8
                RECORD TYPE=V,LENGTH=(100,,20)    9
                INREC  FIELDS=(1,8,29,6,12,6)     10
/*                                                    11
```

Figure 168. Sample JCL/Control Stream (4)

1. The JOB statement gives EDITMERG as the jobname.
2. The EXEC statement identifies SYNCSORT as the program to be executed.
3. The STEPLIB DD statement instructs the system to look for SyncSort in the library named SORT.RESI.DENCE. The DISP shows the library may be shared.
4. The SYSOUT DD statement assigns the SyncSort messages to the output device associated with SYSOUT class A.
5. Three data sets are to be merged: SALES91, SALES92 and SALES93. SALES91 and SALES92 are found on standard labeled tapes with the volume serial numbers 123456 and 654321, respectively. The DD statement for SALES93 specifies a 3390 disk device with the volume serial number DISK11. These three data sets are already in existence, and the disk data set SALES93 may be shared. They are assigned distinct SORTINnn numbers, as required.
6. The SORTOUT DD statement assigns the name SALES.PATTERN to the output data set and specifies a 3390 disk device with the volume serial number DISK08. Five

cylinders of primary space have been allocated on this volume. The data set does not yet exist. DCB parameters are provided, preventing them from defaulting to those of the SORTIN08 file.

7. The SYSIN DD statement marks the beginning of the system input stream that includes the sort control statements.
8. The MERGE control statement specifies one control field. It begins on byte 5 (the first data byte of the record since TYPE=V is specified on the RECORD statement) and is 4 bytes long. This field contains zoned decimal data and is to be merged in ascending order.
9. The RECORD statement indicates that variable-length records are being merged and indicates the record length at various processing stages. The maximum input record length is specified as 100 bytes. Since there is no E15, the post-E15 length value is not coded and so defaults to this figure. The INREC statement reduces this maximum record length to just 20 bytes.
10. According to the RECORD control statement, the input record may be 100 bytes long. The INREC statement reduces each record to the 20 bytes crucial to this application: the 4-byte RDW and 4-byte merge control field (i.e., the first 8 bytes of the record), the 6-byte field beginning at byte 29 (the 25th data byte) and the 6-byte field beginning at byte 12 (the 8th data byte). As required, the RDW remains in the first four bytes. The records to be merged are no more than 20 bytes long and contain three fields following the RDW.
11. The delimiter statement marks the end of the SYSIN input stream.

A Copy without Exit Routines

Example 5

```
//COPYNYC      JOB                               1
//COPY1       EXEC  PGM=SYNCSORT                 2
//STEPLIB    DD   DSN=SORT.RESI.DENCE,DISP=SHR   3
//SYSOUT     DD   SYSOUT=A                       4
//SORTIN     DD   DSN=USA.OUTLETS,UNIT=TAPE,     5
//           VOL=SER=149200,DISP=(OLD,KEEP)
//SORTOUT    DD   DSN=NYC.OUTLETS,UNIT=3390,     6
//           VOL=SER=DISK08,SPACE=(CYL,5),
//           DISP=(NEW,KEEP)
//SYSIN      DD   *                               7
              SORT   FIELDS=COPY                 8
              INCLUDE COND=(56,3,CH,EQ,C'NYC')  9
/*                                                10
```

Figure 169. Sample JCL/Control Stream (5)

1. The JOB statement gives COPYNYC as the jobname.
2. The EXEC statement identifies SYNCSORT as the program to be executed.
3. The STEPLIB DD statement instructs the system to look for SyncSort in the library named SORT.RESI.DENCE. The DISP shows the library may be shared.
4. The SYSOUT DD statement assigns the SyncSort messages to the output device associated with SYSOUT class A.
5. The SORTIN DD statement indicates the file to be copied. The data set name is USA.OUTLETS, and it is found on the standard labeled tape with the volume serial number 149200. The data set is already in existence.
6. The SORTOUT DD statement names the copied file NYC.OUTLETS, and specifies a 3390 disk device with the volume serial number of DISK08. Five cylinders of primary space have been allocated on this volume. The data set does not yet exist, but is to be kept whether or not the job terminates normally. The DCB RECFM and LRECL parameters for SORTOUT default to that of the first SORTIN file. The BLKSIZE will be selected by System Determined BLKSIZE (SDB) if active or by SyncSort if SDB is not active.
7. The SYSIN DD statement marks the beginning of the system input stream that includes the sort control statements.

8. The `FIELDS` parameter specifies a copy application. This could have been coded as `MERGE FIELDS=COPY` without affecting program execution.
9. The `INCLUDE` control statement edits the `USA.OUTLETS` input file, eliminating all records which do not have the character string `NYC` in bytes 56-58. Only 'NYC' records will be copied.
10. The delimiter statement marks the end of the `SYSIN` input stream.

A Sort with an Exit Routine Already Link-edited

Example 6

//ONE#EXIT	JOB		1
//STEP1	EXEC	PGM=SYNCSORT, PARM= 'MSG=SC'	2
//STEPLIB	DD	DSN=SORT.RESI.DENCE, DISP=SHR	3
//SYSOUT	DD	SYSOUT=A	4
//MODLIB	DD	DSN=EXIT.E15, DISP=SHR	5
//SORTIN	DD	DSN=INPUT, UNIT=3390,	6
//		VOL=SER=ABCDEF, DISP=(SHR)	
//SORTOUT	DD	DSN=OUTPUT, UNIT=3390,	7
//		VOL=SER=GHIJKL, SPACE=(CYL,10)	
//		DISP=(NEW,KEEP,DELETE)	
//SORTWK01	DD	UNIT=SYSDA, SPACE=(CYL,20)	8
//SORTWK02	DD	UNIT=SYSDA, SPACE=(CYL,20)	
//SORTWK03	DD	UNIT=SYSDA, SPACE=(CYL,15)	
//SORTWK04	DD	UNIT=SYSDA, SPACE=(CYL,15)	
//SYSIN	DD	*	9
	SORT	FIELDS=(10,25,CH,A,40,10,ZD,D),	10
		FILSZ=9000	
	RECORD	TYPE=V,LENGTH=(1024,,44,192)	11
	MODS	E15=(E15,600,MODLIB,N)	12
	END		13
/*			14

Figure 170. Sample JCL/Control Stream (6)

1. The JOB statement gives ONE#EXIT as the jobname.
2. The EXEC statement identifies SYNCSORT as the program to be executed. The MSG PARM option requests that all messages be routed to the SYSOUT DD statement but only critical messages be routed to the console.
3. The STEPLIB DD statement instructs the system to look for SyncSort in the library named SORT.RESI.DENCE. The DISP shows the library may be shared.
4. The SYSOUT DD statement assigns the SyncSort messages to the output device associated with SYSOUT class A.
5. The MODLIB DD statement defines the library in which the exit routine resides; MODLIB is referenced in the MODS control statement. The data set name of the library is EXITE15, and the DISP shows that the library may be shared.

6. The SORTIN DD statement gives INPUT as the input data set name and specifies a 3390 disk with the volume serial number ABCDEF. The DISP parameter indicates that the data set is already in existence and may be shared.
7. The SORTOUT DD statement gives OUTPUT as the output data set name and specifies a 3390 disk with the volume serial number GHIJKL. Ten cylinders of primary space have been allocated on this volume. The DISP parameter shows that this data set is not yet in existence.
8. The four SORTWK statements reserve space on four temporary data sets for intermediate storage. Twenty cylinders are to be reserved on the first two data sets, fifteen on the second two data sets.
9. The SYSIN DD statement marks the beginning of the input stream that includes the sort control statements.
10. The SORT control statement specifies two sort control fields. The first begins on byte 10 (data byte 6) of the record, is 25 bytes long, contains character data, and is to be sorted in ascending order. The second control field begins on byte 40 (data byte 36) of the record, is 10 bytes long, has zoned decimal data, and is to be sorted in descending order. FILSZ instructs SyncSort to terminate abnormally unless the post-E15 file contains exactly 9,000 records.
11. The RECORD control statement shows that variable-length records are being sorted. The first LENGTH value reports that the maximum length of records in the SORTIN data set is 1024 bytes. The comma coded for the second LENGTH value shows that this maximum length is not altered by the exit routine. The comma coded for the third LENGTH value shows that this maximum length is not affected by an E35 or the INREC/OUTREC statements. The fourth LENGTH value shows that the smallest record in the input data set is 44 bytes long. The fifth LENGTH value shows that the record length that occurs most frequently in SORTIN is 192 bytes. (This value will be used to determine segment size.)
12. The MODS control statement states that the exit-type is E15. The name of the actual exit routine included at this exit is also E15. The routine requires 600 bytes of memory and resides in a library defined on the MODLIB DD statement. Finally, the N indicates that link-editing of the routine has already been performed.
13. The END control statement marks the end of the control statements.
14. The delimiter statement marks the end of the SYSIN input stream.

A Sort with an Exit Routine to be Link-edited

Example 7

//LINKEXIT	JOB		1
//STEP	EXEC	PGM=SYNCSORT	2
//SYSOUT	DD	SYSOUT=A	3
//SORTIN	DD	DSN=IN.FILE.JANUARY,	4
//		UNIT=TAPE,VOL=SER=135790,	
//		DISP=OLD,DELETE),	
//		DCB=(LRECL=200,RECFM=FB,	
//		BLKSIZE=4000),LABEL=(2,SL)	
//SORTOUT	DD	DSN=OUT.FILE.FEBRUARY,	5
//		UNIT=TAPE,VOL=SER=097863,	
//		DISP=(NEW,KEEP),LABEL=(1,SL)	
//SORTMODS	DD	DSN=A.PART.DATA.SET,DISP=OLD	6
//MODLIB	DD	DSN=EXIT.NO.ONE,DISP=SHR	7
//SYSLMOD	DD	DSN=&&LINK,UNIT=SYSDA,	8
//		SPACE=(CYL,(1,1,1))	
//SYSLIN	DD	DSN=&&TEMP,UNIT=SYSSQ,	9
//		SPACE=(TRK,1)	
//SYSPRINT	DD	SYSOUT=A	10
//SYSUT1	DD	UNIT=SYSDA,SPACE=(CYL,(1,1))	11
//SYSIN	DD	*	12
	SORT	FIELDS=(20,30,CH,A),	13
		DYNALOC=(SYSDA,6)	
	RECORD	TYPE=F,LENGTH=200	14
	MODS	E15=(EXIT1,600,MODLIB,N),	15
		E35=(EXIT2,500,SYSIN)	
	SUM	FIELDS=(1,10,ZD) TOTAL BALANCE	16
	END	ACCOUNTS FOR JANUARY BEGIN FEBRUARY	17
	.		
	.		
	.		
	Object deck EXIT2 for E35 exit routine		18
	.		
	.		
	.		
/*			19

Figure 171. Sample JCL/Control Stream (7)

1. The JOB statement gives LINKEXIT as the jobname.
2. The EXEC statement identifies SYNCSORT as the program to be executed.

3. The SYSOUT DD statement assigns the SyncSort messages to the output device associated with SYSOUT class A.
4. The SORTIN DD statement gives IN.FILE.JANUARY as the input data set name, and specifies a tape unit with the volume serial number 135790. The DISP parameter shows that the data set is already in existence.

The DCB parameter shows an LRECL of 200 bytes, a fixed blocked RECFM, and a 4000-byte BLKSIZE. The LABEL parameter shows that IN.FILE.JANUARY is the second data set on the tape, and that it has a standard label.

5. The SORTOUT DD statement gives OUT.FILE.FEBRUARY as the output data set name, and specifies a tape unit with the volume serial number 097863. The DISP parameter shows that the data set is not in existence yet.

The DCB RECFM and LRECL parameters for SORTOUT default to that of the first SORTIN file. The BLKSIZE will be selected by System Determined BLKSIZE (SDB) if active or by SyncSort if SDB is not active. The LABEL parameter shows that OUT.FILE.FEBRUARY is to be the first data set on the tape, and will have a standard label.

6. The SORTMODS DD statement defines the partitioned data set that will contain the exit routine object module that has not been link-edited and is being included in the SYSIN data stream. The DISP shows the data set may not be shared.
7. The MODLIB DD statement defines the partitioned data set in which the already link-edited exit routine resides. (Note MODLIB is referenced on the MODS control statement.) The data set name of the exit library is EXIT.NO.ONE. The DISP shows the data set may be shared.
8. The SYSLMOD DD statement defines a temporary data set called &&LINK that will contain the exit routine after it has been link-edited. A direct access device will be used with 1 cylinder reserved for primary space allocation, 1 cylinder for secondary space allocation, and 1 directory block.
9. The SYSLIN DD statement defines the temporary data set that will contain the linkage editor control statements that SyncSort will use when link-editing the exit. The name of this data set is &&TEMP. It is to be on any sequential-access device with 1 track reserved if the data set is allocated to disk.
10. The SYSPRINT DD statement defines the data set on which the linkage editor will write its messages. Whatever device is assigned to SYSOUT=A will be used.
11. The SYSUT1 DD statement defines the temporary data set that will be used as a work area by the linkage editor. It is to be on a direct access device with 1 cylinder of primary space allocated, and 1 cylinder of secondary space allocated.

12. The SYSIN DD statement marks the beginning of the input stream that includes the sort control statements and also the object deck of the exit routine to be link-edited.
13. The SORT control statement shows that one control field will be sorted on. It begins on byte 20 of the record, is 30 bytes long, contains character data, and is to be sorted according to ascending order.

The DYNALLOC parameter specifies that 6 direct access areas are to be reserved for sortwork data sets.

14. The RECORD control statement shows that fixed-length records are being sorted. The LENGTH parameter gives 200 bytes as the length of the records at input time, and, by not specifying values for l_2 and l_3 , implicitly states that the length of these records will not be changed during the sort.
15. The MODS control statement shows that the first exit-type is E15. The name of the routine for this exit is EXIT1. It will take 600 bytes in main storage, resides in a library defined on the MODLIB DD statement, and has already been link-edited.

The second exit-type is E35. The name of the routine for the exit is EXIT2, and it will take 500 bytes in main storage. The object deck for the routine is to be included in the SYSIN portion of the job stream, and, because of the absence of a letter in the last sub-parameter position for this group, the sort assumes that the routine requires link-editing and will be link-edited together with any other routines for this phase.

16. The SUM control statement's FIELDS parameter identifies one summary field. It begins on byte 1 of the record, is 10 bytes long, and has zoned decimal data. The rest of the statement is a comment.
17. The END control statement marks the end of the control statements and also contains a comment.
18. The EXIT2 object deck to be link-edited is included after the END statement in the SYSIN stream.
19. The delimiter statement marks the end of the SYSIN input stream for the sort.

Multiple Output Files

Example 8

```
//MULTOUT      JOB                               1
//              EXEC   PGM=SYNCSORT              2
//STEPLIB      DD    DSN=SORT.RESI.DENCE,DISP=SHR  3
//SYSOUT       DD    SYSOUT=A                    4
//SORTIN       DD    DSN=SALES.RECORDS,          5
//              VOL=SER=DISK1,DISP=SHR
//SORTOUT      DD    DSN=SORTED.SALES.RECORDS,   6
//              UNIT=TAPE,VOL=SER=112233,
//              DISP=(NEW,KEEP)
//SORTOFDS     DD    DSN=DOMESTIC.SALES.RECORDS,  7
//              VOL=SER=DISK8,DISP=(NEW,KEEP),
//              SPACE=(CYL,40),UNIT=SYSDA
//SORTWK01     DD    SPACE=(CYL,20),UNIT=SYSDA    8
//SORTWK02     DD    SPACE=(CYL,20),UNIT=SYSDA
//SORTWK03     DD    SPACE=(CYL,20),UNIT=SYSDA
//SYSIN        DD    *                            9
              SORT   FIELDS=(10,12,BI,A)         10
              OUTFIL  FILES=OUT,INCLUDE=ALL      11
              OUTFIL  FILES=DS,OMIT=(62,3,CH,NE,C'USA') 12
/*                                                    13
```

Figure 172. Sample JCL/Control Stream (8)

1. The JOB statement gives MULTOUT as the jobname.
2. The EXEC statement identifies SYNCSORT as the program to be executed.
3. The STEPLIB DD statement instructs the system to look for SyncSort in the library named SORT.RESI.DENCE. The DISP parameter shows that the library may be shared.
4. The SYSOUT DD statement assigns the SyncSort messages to the output device associated with SYSOUT class A.
5. The SORTIN DD statement gives SALES.RECORDS as the input data set name, and specifies a disk with the volume serial DISK1. The DISP parameter indicates that the data set is already in existence and may be shared.
6. The SORTOUT DD statement names one of the output sorted files SORTED.SALES.RECORDS, and specifies a tape device with volume serial number 112233 for storage. The DISP parameter indicates that the data set does not yet exist, but it is to be kept whether or not the job terminates normally.

7. The SORTOFDS DD statement names a second sorted output file DOMESTIC.SALES.RECORDS, and specifies a disk device with volume serial number DISK8 for storage. Forty cylinders of space have been allocated on this volume. The DISP parameter indicates that the data set does not yet exist, but is to be kept whether or not the job terminates normally.
8. The three SORTWK DD statements reserve space on direct access devices for intermediate storage. Twenty cylinders are allocated for each of the three SORTWK data sets.
9. The SYSIN DD statement marks the beginning of the input stream that includes the sort control statements.
10. The SORT control statement specifies that one control field will be sorted on. It begins on byte 10 of the record, is 12 bytes long, contains unsigned binary (BI) data and is to be sorted according to ascending order.
11. The first OUTFIL control statement is associated with the SORTOUT DD statement. The INCLUDE parameter specifies that all input records are to be included in this output file.
12. The second OUTFIL control statement is associated with the SORTOFDS DD statement. The OMIT parameter specifies that records which do not contain "USA" in bytes 62, 63 and 64 are not to be included in this file.
13. The delimiter statement marks the end of the SYSIN input stream.

Chapter 5. PARM Options

PARM options can be specified to provide processing information and to override installation defaults for JCL-initiated and program-invoked applications.

For a JCL-initiated application, specify the PARM option(s) on the EXEC statement as follows:

```
PARM='option,...'
```

Figure 173. PARM Parameter Format

For a program-invoked application, specify the PARM option(s) in a \$ORTPARM DD data set. Omit the keyword PARM= and the single quotes. PARM options for a JCL-initiated application can also be specified in a \$ORTPARM data set.

Additional MAXSORT PARMs

The MAXSORT feature, designed for large sorting applications, is initiated by the MAXSORT PARM. The following additional PARMs can be specified for a MAXSORT application: BKPTDSN, DYNATAPE, MAXWKSP, MINWKSP, NODYNATAPE, RESTART, SORTSIZE, SORTTIME and TAPENAME. These PARMs are described in “Chapter 9. MAXSORT”.

PARASORT PARM

The PARASORT feature, designed to reduce elapsed time for multi-volume and/or concatenated tape SORTIN sort applications, is initiated by the PARASORT PARM. For additional information on PARASORT, see “Chapter 10. PARASORT”.

DB2 Query Support PARM

DB2 Query Support, which allows DB2 data to be passed directly into a SORT or COPY operation without the use of setup steps or the need for user-written E15 exits, is initiated by the DB2 Query Support PARM. For additional information, see “Chapter 11. SyncSort DB2 Query Support” .

Additional Tape Sort PARMs

The following additional PARMs can be specified for a Tape Sort: OSCL, BALN and POLY. The CRCX and PEER PARMs are accepted but ignored. These PARMs are described in “Chapter 12. Tape Sort”.

Precedence Rules

There are three ways in which options can be specified, though not all options can be specified in all three ways:

- As an installation specification
- As a PARM specification
- As a SORT/MERGE control statement specification.

Note that there are six options that can be specified as a PARM option or a SORT/MERGE option. They are: CENTWIN, DYNALLOC, EQUALS/NOEQUALS, FILSZ, SKIPREC and STOPAFT.

When an option is specified in more than one way, the following precedence rules apply:

- A SORT/MERGE or PARM specification overrides an installation specification.
- A PARM specification overrides a SORT/MERGE specification except for EQUALS/NOEQUALS.

PARM Option Summary Chart

The chart on the following pages lists the PARM options. Underscored PARM options are delivered defaults which may have been altered at installation time.

PARM Option Name	Disk Sort, MAXSORT, and PARASORT	Available with Tape Sort?
BALANCE	Balances importance of CPU time, elapsed time, and I/O activity for best overall sort performance. See CPU, ELAP, and IO. Note that these options and BALANCE are all mutually exclusive.	No.
BMSG	Produces WERnnB messages.	No.
CENTWIN= <u>Q</u> /s /f	Generates a sliding (s) or fixed (f) 100-year window that determines the century to which 2-digit year data belongs. Ensures that such data is processed correctly as a 4-digit year by SORT/MERGE and INCLUDE OMIT. Also enables OUTREC processing to output a 4-digit year (yyyy) from 2-digit year input (yy).	No.
CMP= <u>CPD</u> /CLC	CMP=CPD improves performance.	No. CMP=CLC (no data validation) is the standard.
COBEXIT= COB1 / <u>COB2</u>	Specifies whether COBOL exits use OS/VSE COBOL libraries or VS COBOL II or COBOL/370 libraries.	No.
COMMAREA/ <u>NOCOMMAREA</u>	Provides a communication area between exit programs.	No
CORE/SIZE=n	Changes the amount of memory in which sort/merge can run.	Yes.
CPU	Minimizes CPU time at expense of other performance measures. See BALANCE, ELAP, and IO. Note that these options and CPU are all mutually exclusive.	No.
DEBUG	Provides a SyncSort SNAP dump in the event of a critical error.	No.
DIAG	Provides diagnostic information for certain error conditions.	Plus additional diagnostic trace.

PARM Option Name	Disk Sort, MAXSORT, and PARASORT	Available with Tape Sort?
DYNALLOC	Requests the dynamic allocation of work data sets.	No.
E15/E35=COB	Indicates a COBOL exit.	No.
ELAP	Minimizes elapsed time at expense of other performance measures. See BALANCE, CPU, and IO. Note that these options and ELAP are all mutually exclusive.	No.
EQUALS/ <u>NOEQUALS</u>	EQUALS acts to preserve the order of equal-keyed records. It is not available with PARASORT.	Yes.
EXTCOUNT	Enables special processing for applications with record counts that exceed SyncSort's default internal limit.	No.
FILSZ=n/En	Indicates the (actual or estimated) number of records <i>after</i> input processing (E14, E15, INCLUDE/OMIT, SKIPREC, STOPAFT, and Phase 1 SUM). FILSZ=n causes sort termination if n is incorrect.	FILSZ=En only. Does not take input processing (E14, E15) into account.
FLAG	FLAG and MSG control the routing of output messages.	Yes.
HBSI	Enables hiperbatch processing for SORTIN data set.	No.
HBSO	Places SORTOUT data set into hiperbatch.	No.
INCOR/INCORE= <u>ON</u> /OFF	ON permits incore and turnaround sorts.	No. INCORE=OFF is the standard.
IO	Minimizes IO activity at expense of other performance measures. See BALANCE, CPU, and ELAP. Note that these options and IO are all mutually exclusive.	No.
IOERR=ABE/ <u>NOIOERR</u>	Indicates how to handle I/O errors: user abend 999 plus dump or SyncSort error message only.	No. I/O errors generate user abend 999 plus dump.

PARM Option Name	Disk Sort, MAXSORT, and PARASORT	Available with Tape Sort?
L6=n,L7=n	Passes HISTOGRM data to optimize variable-length record sorts.	No.
<u>LIST</u> /NOLIST	LIST causes header line and control statements to be printed.	Accepted but not processed. NOLIST is the standard for invoked sorts, LIST for JCL sorts.
LOCALE= <u>NONE</u> /CURRENT /name	Controls collating based on cultural environment.	No.
MSG	MSG and FLAG control the routing of messages.	Yes.
MSGDD	Changes the DD name of the message data set.	Yes.
NULLOUT= <u>RC0</u> /RC4 /RC16	Specifies the action to be taken when SORTOUT contains no records.	No.
OVFLO= <u>RC0</u> /RC4 /RC16	Specifies the action to be taken if a summary field overflows or underflows during SUM processing.	No.
PAD= <u>RC0</u> /RC4 /RC16	Specifies the action to be taken if the non-OUTFIL SORTOUT LRECL is larger than the SORTIN/SORTINnn LRECL.	No.
PRINT121	Changes the DCB of the message data set.	No. The standard is: DCB=(LRECL=120,BLKSIZE=120, RECFM=U).
RC16=ABE/NORC16	RC16=ABE changes return code 16 to user abend 16.	No. User abend 16 is the standard.
RELEASE= <u>ON</u> /OFF	Overrides the RLSE operand in the SPACE parameter of the SORTWK DD statement(s).	No.
RESERVE=n/nK	Specifies the amount of memory reserved for the user below the 16-megabyte line.	No.
RESERVEX=n/nK	Specifies the amount of memory reserved for the user above the 16-megabyte line.	No.
RESET/ <u>NORESET</u>	Affects VSAM SORTOUT only.	No. (Tape Sort does not accept VSAM output.)
RLSOUT/ <u>NORLSOUT</u>	Determines whether excess space is released.	No. Excess SORTOUT space is not released.

PARM Option Name	Disk Sort, MAXSORT, and PARASORT	Available with Tape Sort?
SDB= ON/ OFF/ YES/ NO/ DISKONLY/ TAPEONLY/ LARGE/ SMALL/ <u>INPUT</u> / LARGEONLY/ INPUTONLY	Specifies whether system-determined blocksize should be used to select an optimum blocksize for data sets when none is provided.	No.
SECOND= <u>ON</u> /OFF	Determines whether secondary sort work space is automatically provided.	No. Secondary sort work space is not automatically provided.
SKIPREC=n	Indicates that <i>n</i> records should be skipped before the input file is sorted or copied. SKIPREC is not available for PARASORT.	No.
STOPAFT=n	Sorts or copies at most <i>n</i> records that survive input file editing (E15, INCLUDE/OMIT, SKIPREC etc.) STOPAFT is not available for PARASORT.	No.
TRUNC= <u>RC0</u> /RC4 /RC16	Specifies the action to be taken if the non-OUTFIL SORTOUT LRECL is smaller than the SORTIN/SORTINnn LRECL.	No.
UNINTDS=YES/ <u>NO</u>	Indicates if an uninitialized SORTIN or SORTINnn input file should be processed.	No.
VLTEST=(n/ <u>1</u> , <u>ON</u> /OFF / OFF4)	Indicates the type of validity testing to be done when processing variable-length records.	No.
VLTESTI=n/ <u>0</u>	Indicates action to be taken when a variable-length record does not contain all fields referenced by INCLUDE or OMIT processing.	No.
VSAMEMT= <u>NO</u> /YES	Specifies the processing of empty VSAM data sets provided as input to a sort, merge, or copy.	No.
<u>ZDPRINT</u> / NZDPRINT	Specifies whether positive summarized ZD fields will be converted to a printable format.	No.

SyncSort PARM Options

BALANCE

BALANCE

BALANCE optimizes overall performance by balancing among CPU time, sort elapsed time, and I/O activity to SORTIN, SORTOUT and SORTWK. If you wish to emphasize one performance measure at the possible expense of others, use CPU, ELAP, or IO. See CPU, ELAP, and IO, below. Note that these options and BALANCE are all mutually exclusive.

Cannot be used with Tape Sort.

BMSG

BMSG

BMSG enables class B messages. They will appear wherever the MSG PARM option indicates informational messages are to be routed.

Cannot be used with Tape Sort.

CENTWIN

CENTWIN = $\left. \begin{array}{c} 0 \\ - \\ s \\ f \end{array} \right\}$

CENTWIN defines a sliding or fixed 100-year window that determines the century to which 2-digit year data belongs when processed by SORT, MERGE, INREC, OUTREC or OUTFIL OUTREC control statements.

The 2-digit year data formats (Y2B, Y2C, Y2D, Y2P, Y2S and Y2Z) plus the full-date formats (Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y) work with CENTWIN to treat a 2-digit year value as a 4-digit year. The 2-digit and full-date year data formats can be specified on control statements as follows:

- Use SORT/MERGE control statements to correctly collate 2-digit years that span century boundaries. For information on using the 2-digit and full-date data formats for SORT/MERGE field specifications, see “CENTWIN Parameter (Optional)” on page 2.41 or on page 2.134.

- Use INCLUDE/OMIT or OUTFIL INCLUDE/OMIT control statements for correct comparisons involving 2-digit year and full-date data formats. For information on using the 2-digit year data formats for INCLUDE/OMIT processing, see "Specifying Field-to-Field Standard Comparisons for Date Fields" under the heading "INCLUDE/OMIT Control Statement" on page 2.16. For more information on specifying full-date formats, see pages 2.24-2.29.
- Use INREC/OUTREC or OUTFIL OUTREC control statements to convert 2-digit year and full-date data to 4-digit printable output. For information on using the 2-digit year data formats for OUTREC processing, see "Converting Year Data with Century Window Processing on INREC, OUTREC, or OUTFIL OUTREC" on page 2.100 and "Example 5" on page 2.120. For more information on converting full-date formats, see the descriptions of the f_1 and $f_{y2f(c)}$ parameters on pages 2.93-2.96, Table 11 on page 2.47, and Table 15 on page 2.97.

In addition, two date formats, Y2ID and Y2IP, are provided for year conversion with INREC/OUTREC and OUTFIL OUTREC. These formats work with CENTWIN to expand a 2-digit year in packed decimal format to a 4-digit year while maintaining the packed decimal format in the output field.

CENTWIN ensures that year data spanning centuries will be sequenced correctly. For example, without CENTWIN processing, an ascending sort/merge would sequence the year 01 before the year 98. With CENTWIN processing, the 01 field could be recognized as a twenty-first century date (2001) and would thus be sequenced after 98 (1998) for an ascending sort.

The CENTWIN option generates either a sliding or fixed century window, depending on which form of CENTWIN is used: CENTWIN=s or CENTWIN=f.

- CENTWIN=s specifies a sliding century window, which automatically advances as the current year changes.

The variable *s* is a number 0 through 100. This value is subtracted from the current year to set a century-window starting point. For example, in 1996 CENTWIN=20 would create the century window 1976 through 2075. Ten years later in 2006, the century starting year would slide to 1986 (2006 minus 20 = 1986) and the century window would be 1986 through 2085.

The CENTWIN delivered default is *s*=0, which means the current year is the starting year of a century window.

- CENTWIN=f specifies a fixed century window.

The variable *f* is a 4-digit year (yyyy) between 1000 and 3000. For example, CENTWIN=1976 establishes a fixed starting year 1976 for the century window 1976 through 2075. This window will not change as the current year changes.

The century window defined by CENTWIN controls processing of year-data. If a 2-digit year field (indicated by Y2B, Y2C, Y2D, Y2P, Y2S, Y2Z, Y2ID, Y2IP, Y2T, Y2U, Y2V, Y2W, Y2X, or Y2Y) has a value less than the last two digits of the century window start year, the year field will be treated as a year in the century following the year of the century window, except for 00, which is considered to be in the same century as the century window start year. All other 2-digit years will be treated as in the same century as the century window start year.

For example, consider the century window 1950 through 2049. The 2-digit year fields would be processed as follows:

Two-digit Field	Processed as Year
00	2000
01	2001
49	2049
50	1950
99	1999

An ascending sort of the above sample data would produce output data in the following sequence:

Two-digit Field	Processed as Year
50	1950
99	1999
00	2000
01	2001
49	2049

If CENTWIN has been specified on the SORT or MERGE control statement as well as in the PARM field, the PARM specification has precedence.

Cannot be used with Tape Sort.

CMP

$$\text{CMP} = \left\{ \begin{array}{l} \text{CPD} \\ \text{CLC} \end{array} \right\}$$

CMP specifies the kind of compare operation to be used for sort/merge control fields up to 16 bytes long, bearing the format code PD or ZD.

When CMP=CPD is used, ZD fields are PACKed and then compared. Invalid PD data may cause a system 0C7 abend and program termination. The integrity of fields labelled "ZD" is

only guaranteed when they contain valid ZD data. Since the zone bits (the leftmost four bits of each byte) are lost during packing, UNPKing the field later restores only valid ZD data to its original state. Leading blanks are transformed to leading zeros and alphabetic character data that packs to a valid PD field is converted to valid ZD data.

CMP=CLC uses the compare logical instruction for all PD and ZD control fields. No data validation is done and the integrity of the output is maintained.

CMP=CPD is the delivered default for the PARM. The delivered default for VLTEST is 1, and is consistent with this default. Changing the VLTEST default from 1 to any even number forces the use of CMP=CLC when sorting variable-length records.

For more detailed information and sample comparisons, see the section “Comparing PD and ZD Control Fields” on page 2.41.

Cannot be used with Tape Sort.

COBEXIT

$\text{COBEXIT} = \begin{cases} \text{COB1} \\ \text{COB2} \end{cases}$

COBEXIT indicates which libraries COBOL E15 and E35 exit routines should use when they are executed.

COBEXIT=COB1 specifies that COBOL exits should use either OS/VS COBOL libraries or no libraries at all.

COBEXIT=COB2, the delivered default, specifies that COBOL exits should use VS COBOL II or COBOL/370 libraries. VS COBOL II or COBOL/370 run-time library modules typically require more main storage than OS/VS COBOL library modules. The amount of additional storage depends on VS COBOL II or COBOL/370 installation options. When VS COBOL II or COBOL/370 run-time library modules are used, it may be necessary to account for this additional storage by adjusting the b value of the Exit-Name parameter on the MODS statement.

Cannot be used with Tape Sort.

COMMAREA

{ COMMAREA(n,x) COMMAREA(n) COMMAREA(x) COMMAREA NOCOMMAREA }

COMMAREA instructs SyncSort to provide an area for communication between exit programs. The size of this area is given as a decimal number n of bytes; x , a character string at most n bytes long, designates the initial value to be stored in this area. Regardless of the value of n , which may be between 1 and 256, x may not exceed 89 bytes in length. (Whenever x has fewer than n characters, it will be right-padded with blanks to a length of n .) If COMMAREA is specified via the EXEC statement, blanks may be included within the string x . However, if COMMAREA is specified via the \$ORTPARM DD statement, intervening blanks are *not* allowed. In neither case is a right parenthesis permitted since it delimits the COMMAREA parameter.

Both n and x are optional. If either subparameter is specified, it will determine the other: n defaults to the length of x , x defaults to n blanks. If neither x nor n is specified, n defaults to 80 bytes, x to 80 blanks.

NOCOMMAREA is the program default: no area for communication between exit programs is provided, although exit routines may still use the 19th word of the save area.

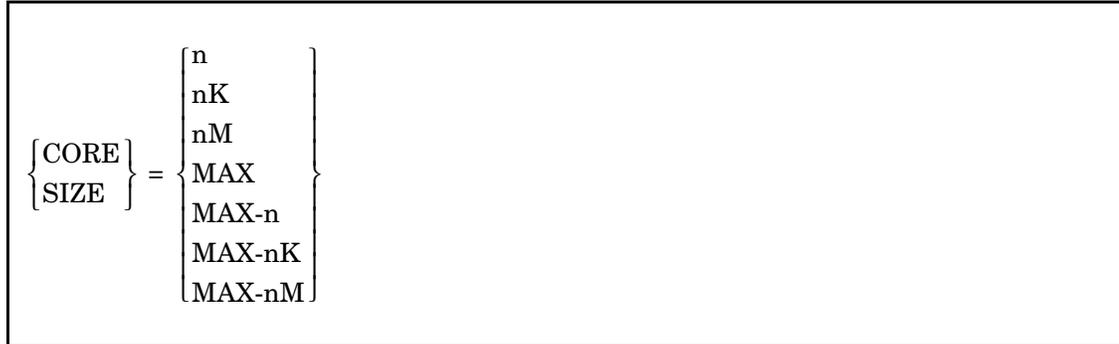
Exit program access to this communication area is described in the discussion of exit programs, see “The Exit Communication Area” on page 7.4.

Cannot be used with Tape Sort.

PARM Code	Communication Area
COMMAREA (10,DEBUG)	DEBUG..... (5 blanks)
COMMAREA (10) (10 blanks)
COMMAREA (80 blanks)
COMMAREA (,DEBUG)	DEBUG
COMMAREA (80,DEBUG)	DEBUG.... (75 BLANKS)

Figure 174. Examples: Coding the COMMAREA PARM

CORE



CORE is used to override the installation default for the amount of memory the sort/merge is allowed to use. To specify an amount of memory, choose one entry from each pair of braces.

Note that CORE and SIZE are synonymous. Note also that memory specification may be a decimal number of bytes (CORE=n), a decimal multiple of K, where K=1024 bytes (CORE=nK), or a decimal multiple of M, where M=1048576 bytes (CORE=nM).

For simplicity, the following describes only CORE, specified in units of nK.

CORE=nK	Defines a maximum memory limit of nK <i>below</i> the 16-megabyte line.
CORE=MAX	Assigns to the sort/merge all the available memory <i>above</i> and <i>below</i> the 16-megabyte line.
CORE=MAX-nK	Assigns to the sort/merge all the available memory <i>above</i> and <i>below</i> the 16-megabyte line less nK bytes, which is reserved <i>below</i> the 16-megabyte line.

Consult your systems staff for any installation-specific modifications to the handling of CORE. For example, "CORE" will be limited if a maximum memory size for SyncSort was set at installation time.

CPU

CPU

CPU minimizes the CPU time of each sort at the expense of sort elapsed time and I/O activity. See BALANCE, ELAP and IO. Note that these options and CPU are all mutually exclusive.

Cannot be used with Tape Sort.

DEBUG

DEBUG

DEBUG produces a SyncSort SNAP dump in the event that a critical error forces the sort to terminate. A SNAP dump produced in this way is of use to a SyncSort analyst in debugging complex problems. See “What to Do Before Calling SyncSort for z/OS Product Services” on page 16.72. Note that the PSW AT ENTRY TO SNAP and general registers are useless for debugging.

Cannot be used with Tape Sort.

DIAG

DIAG

DIAG turns on both the IOERR=ABE and the RC16=ABE options (see these options for explanations). When specified for a Tape Sort, DIAG also turns on a diagnostic trace and causes system ABEND 0C1 in the event of an I/O error.

DYNALLOC

$$\text{DYNALLOC} \left[= \left\{ \begin{array}{l} d \\ (d, n \left[, \text{RETRY} = \left\{ \begin{array}{l} (nn, mm) \\ \underline{\text{OFF}} \end{array} \right\} \right] [, SC=s]) \\ \text{OFF} \end{array} \right\} \right.$$

DYNALLOC requests the dynamic allocation of SORTWK data sets on device type d. Specify the device type either as a decimal number (e.g., 3380) or by the system generic name (e.g., SYSDA). Any disk device accepted for a SORTWK DD statement can be specified. Note that if VIO is specified it will be ignored, and the installation default for the DYNALLOC device type will be used in its place.

Note that the DYNALLOC parameter may be used alone, without any subparameters. In this case, the DYNALLOC installation default settings are used.

For non-MAXSORT applications, n can be 1 through 255. The value n specifies the number of SORTWK data sets that can potentially be allocated. For values of n that are 31 or less, SyncSort can automatically raise the number to 32 if the application requires. When n is 33 through 255, this value specifies the maximum number of SORTWK data sets that can be allocated.

For MAXSORT applications, n is the number of SORTWK data sets that will be allocated. As many as 32 SORTWK data sets can be specified for MAXSORT applications.

The delivered default for n is 3.

DYNALLOC=OFF can be specified to override a DYNALLOC=ON installation default.

SORTWK data sets allocated by the DYNALLOC parameter normally supplement any SORTWK data sets allocated by SORTWKnn DD statements; however, note that there is an installation option to disable DYNALLOC if SORTWKnn DD statements are present.

SyncSort uses the value specified in the RETRY parameter to request automatic DYNALLOC retry. This facility attempts to avoid a sortwork capacity exceeded condition when disk space is not immediately available to satisfy a DYNALLOC request. When RETRY is specified, SyncSort will automatically retry a specified number of times and wait a prescribed interval between DYNALLOC requests.

The nn in the first position designates the number of times SyncSort will retry a failed DYNALLOC request. The minimum allowed is 1 and the maximum is 16. The mm in the second position designates the number of minutes SyncSort waits between each DYNALLOC request. The minimum allowed is 1 and the maximum is 15. RETRY=OFF can be specified to override a RETRY=ON installation default.

In an environment where DFSMS manages temporary work data sets, the SC subparameter specifies a storage class s for SyncSort to use when dynamically allocating SORTWORK data sets. The storage administrator at your installation defines the names of the storage classes you can specify. Note that an installation written automatic class selection (ACS) routine can override the storage class you specify. If SMS is not installed or active to manage temporary work data sets, the d device specification will be used in the SORTWORK dynalloc request.

If DYNALLOC has been specified on the SORT control statement as well as in the PARM field, the PARM specification will take precedence.

Cannot be used with Tape Sort.

E15

E15=COB

E15 specifies the E15=COB option in order to include an E15 exit written in COBOL without coding C on the MODS control statement.

Cannot be used with Tape Sort.

E35

E35=COB

E35 specifies the E35=COB option in order to include an E35 exit written in COBOL without coding C on the MODS control statement.

Cannot be used with Tape Sort.

ELAP

ELAP

ELAP minimizes the elapsed time (wall clock time) of each sort at the expense of CPU time and I/O activity. See BALANCE, CPU and IO. Note that these options and ELAP are all mutually exclusive.

Cannot be used with Tape Sort.

EQUALS

{
EQUALS
NOEQUALS
}

EQUALS guarantees that the first of a series of equal-keyed records is either first-in (SORT) or from the lowest numbered SORTINnn data set (MERGE). With NOEQUALS, there is a random element to the order in which records with identical control fields will appear in the output. With or without EQUALS, MERGE preserves the order of equal-keyed records within any one data set.

When used in conjunction with SUM, EQUALS indicates which of the equal-keyed records will be preserved, containing the sum: the record occurring first in SORTIN (for a sort), or drawn from the SORTINnn data set with the lowest nn number (for a merge) will contain the totaled fields.

The EQUALS option can also be specified on the SORT/MERGE control statement. The specification on the control statement takes precedence over the specification in the PARM field.

EXTCOUNT

EXTCOUNT

EXTCOUNT enables special processing to accommodate applications that have record counts that exceed SyncSort's default internal limit.

By default, the internal limit on the number of records that can be sorted for variable-length data or for a sort application that uses the EQUALS option is 4,294,967,295 records. Specifying EXTCOUNT increases the internal limit to 140,737,488,355,327 records. Fixed-length sorts without EQUALS, and all merges and copies, have automatic support for the maximum number of records allowed by the EXTCOUNT parameter.

Note that additional SORTWK space may be required when specifying the EXTCOUNT parameter with a VL sort or a fixed-length sort with EQUALS. The additional SORTWK space is 2 bytes per record. This amount can add a significant percentage to the SORTWK space needs if the LRECL of the records is small. (Small LRECLs are typical of files with an extremely large number of records). Therefore, when using EXTCOUNT with a VL sort or a fixed-length sort with EQUALS, insure that the extra SORTWK space will be available.

Performance will usually be improved if the EXTCOUNT option is not in effect. Therefore, EXTCOUNT should be used only when appropriate to the application.

If the record limit is exceeded, SyncSort will issue a critical error message and terminate the application.

Cannot be used with Tape Sort.

FILSZ

$$\text{FILSZ} = \left\{ \begin{array}{l} n \\ \text{En} \end{array} \right\}$$

FILSZ indicates the actual (FILSZ=n) or estimated (FILSZ=En) decimal number of records to be sorted, taking into account all record additions and deletions due to an E14 or E15 exit routine, the INCLUDE/OMIT control statement, the SUM control statement and the SKIPREC and STOPAFT parameters.

FILSZ=n instructs SyncSort to terminate with an error message unless exactly n records are to be sorted. Since the number of records SUMed in Phase 1 is indeterminate and may not be reproducible, much less predictable, only the estimated En value should be used if a SUM control statement is present.

If FILSZ is specified for a Tape Sort, only the estimated En value may be given. It should indicate the number of records in the input file - not taking into account any records to be added or deleted by an E14 or E15 exit program.

The FILSZ option can also be specified on the SORT control statement. The specification in the PARM field will take precedence over that on the control statement.

FLAG

$\left. \begin{array}{l} \text{FLAG(I)} \\ \text{FLAG(U)} \\ \text{NOFLAG} \end{array} \right\}$
--

FLAG controls the routing of output messages. The MSG option, which handles messages more comprehensively, is explained later in this section. The format of the FLAG option is given below.

Specify FLAG(I) to route all messages to the data set specified by the SYSOUT DD statement and only critical messages to the console. (This is the same as the MSG=SC PARM.)

Specify FLAG(U) to route critical messages only to both the data set specified by the SYSOUT DD statement and the console. (This is the same as the MSG=CB PARM.)

Specify NOFLAG to route critical messages only to the console, no messages to the data set specified by the SYSOUT DD statement. (This is the same as the MSG=CC PARM.)

HBSI

HBSI

HBSI turns on hiperbatch processing for SORTIN data sets. To benefit from hiperbatch processing, the SORTIN data set should already reside in hiperbatch. Although hiperbatch does provide significant improvements in elapsed time, it causes some degree of degradation in other system resources. If you use HBSI and the SORTIN data set does not reside in hiperbatch, you may experience some system degradation while not realizing any of the benefits that accompany hiperbatch processing.

Cannot be used with Tape Sort.

HBSO

HBSO

HBSO turns on hiperbatch processing for SORTOUT data sets. HBSO benefits only subsequent job steps that utilize hiperbatch to access this data set.

Cannot be used with Tape Sort.

INCORE

$$\left\{ \begin{array}{l} \text{INCORE} \\ \text{INCOR} \end{array} \right\} = \left\{ \begin{array}{l} \text{ON} \\ \text{OFF} \end{array} \right\}$$

INCORE=ON requests SyncSort to perform an incore or a turnaround sort whenever possible, even if DYNALLOC is specified or SORTWKnn data sets are present. This is the delivered default.

With INCORE=OFF, SyncSort will not perform a turnaround sort even if your input data can be sorted entirely in memory. Specifying INCORE=OFF without supplying SORTWKnn data sets or requesting the DYNALLOC option results in a critical error when sorting.

Cannot be used with Tape Sort.

IO

IO

IO minimizes the I/O activity of each sort at the expense of sort elapsed time and CPU time. See BALANCE, CPU and ELAP. Note that these options and IO are all mutually exclusive.

Cannot be used with Tape Sort.

IOERR

$$\left\{ \begin{array}{l} \text{IOERR=ABE} \\ \text{NOIOERR} \end{array} \right\}$$

IOERR specifies IOERR=ABE to receive user abend 999 if an I/O error should occur. This abend will cause the job step to terminate, producing a diagnostic dump.

NOIOERR is the program default. If this option is in effect, SyncSort will, in the event of an I/O error, terminate with either a return code of 16 or a user abend 16, depending on the RC16 option that is used.

Cannot be used with Tape Sort.

L6

L6=n

L6 indicates the average number of bytes of work space each record will need, overriding (if present) the l_6 parameter of the RECORD control statement. The decimal value n of the optional L6 parameter is provided by the HISTOGRM utility program. If neither L6 nor l_6 is provided, SyncSort will estimate this value.

L6 is only used for sorting variable-length records. It is ignored by Tape Sort, merge, and copy applications.

Cannot be used with Tape Sort.

L7

L7=n

L7 indicates the segment length that SyncSort should use for maximum sorting efficiency. The decimal value n of the optional L7 parameter is provided by the HISTOGRM utility program. (A segment is a fixed-length area used to contain all or part of a variable-length record.) The L7 value overrides (if present) the l_7 parameter of the RECORD control statement. If neither L7 nor l_7 is provided, SyncSort will estimate this value.

L7 is only used for sorting variable-length records. It is ignored by Tape Sort, merge, and copy applications.

Cannot be used with Tape Sort.

LIST

$\left\{ \begin{array}{l} \underline{\text{LIST}} \\ \text{NOLIST} \end{array} \right\}$
--

LIST, the default for the sort/merge program, causes header lines and control statements to be listed with the SYSOUT data set (in all likelihood, at the printer) for both JCL- and pro-

gram-initiated executions. If NOLIST is specified, the control statements and header lines will not appear with this data set.

Tape Sort accepts but does not process this parameter.

LOCALE

$\text{LOCALE} = \left\{ \begin{array}{l} \underline{\text{NONE}} \\ \text{CURRENT} \\ \text{name} \end{array} \right\}$
--

LOCALE controls cultural environment processing, allowing you to choose an alternative set of collating rules based on a specified national language. For SORT/MERGE processing, the alternative collating applies to character (CH) fields. For INCLUDE/OMIT comparison processing, the alternative collating applies to character fields and hexadecimal constants compared to character fields.

SyncSort employs the callable services of IBM's Language Environment for z/OS to collate data in a way that conforms to the language and conventions of a selected locale. A locale defines single and multi-character collating rules for a cultural environment. Numerous pre-defined locales are available.

NONE, the default setting for LOCALE, results in normal EBCDIC collating.

CURRENT directs SyncSort to use the locale active when SyncSort begins.

name is the name of a supplied or user-defined locale that is to be active during SyncSort processing. A locale name may be up to 32 characters and is not case sensitive. The locale active just before SyncSort processing begins will be restored when SyncSort processing completes. The following is a list of locales provided with the IBM National Language Resources Feature of LE/370.

Locale Name	Language	Country
C		
DA_DK	Danish	Denmark
DE_CH	German	Switzerland
DE_DE	German	Germany
EL_GR	Greek	Greece
EN_GB	English	United Kingdom
EN_JP	English	Japan
EN_US	English	United States
ES_ES	Spanish	Spain
FI_FI	Finnish	Finland
FR_BE	French	Belgium
FR_CA	French	Canada
FR_CH	French	Switzerland
FR_FR	French	France
IS_IS	Icelandic	Iceland
IT_IT	Italian	Italy
JA_JP	Japanese	Japan
NL_BE	Dutch	Belgium
NL_NL	Dutch	Netherlands
NO_NO	Norwegian	Norway
PT_PT	Portuguese	Portugal
SV_SE	Swedish	Sweden
TR_TR	Turkish	Turkey

Table 27. Defined Locales

Notes:

1. Make sure the JCL gives SyncSort access to the library that contains the loadable locale routines. For the supplied locales, these are the dynamically loadable routines in

the IBM AD/Cycle LE/370 library. For more information, see the IBM publication *Language Environment for z/OS & VM Installation and Customization Guide, SC26-4817*.

2. If locale processing is used for fields specified in a SORT or MERGE control statement, VLTEST=1 will be forced on in addition to any other VLTEST options in effect. VLTEST=1 will cause SyncSort to terminate if a variable-length input record does not contain all SORT/MERGE control fields.
3. Although locale processing can improve performance compared to external collating routines, it should be used only when necessary. Locale processing can significantly degrade SORT/MERGE and INCLUDE/OMIT performance compared to normal collating.
4. An E61 exit cannot be used with locale processing.
5. Locale processing requires additional main storage to support the use of the IBM Language Environment facilities. For those jobs that use locale, the below-the-line region size should be increased by 1000K to accommodate the storage needs of the Language Environment modules.
6. LOCALE cannot be used with Tape Sort.

MSG



MSG indicates where SyncSort messages are to be routed. The MSG codes assume that the printer is specified for the message data set; if a device other than the printer is specified for this data set, messages described as routed to the printer will be routed to this other device instead.

AB causes all messages to be routed both to the printer and to the console.

AC causes all messages to be routed to the console, none to the printer.

- AP** causes all messages to be routed to the printer, none to the console. This is the program default.
- CB** causes only critical messages to be routed to the printer and to the console. (This is the same as the FLAG(U) option.)
- CC** causes only critical messages to be routed to the console, no messages to the printer. (This is the same as the NOFLAG option.)
- CP** causes only critical messages to be routed to the printer, no messages to the console.
- NO** causes no messages to be routed to either the printer or the console.
- PC** causes all messages to be routed to the printer and to the console.
- SC** causes only critical messages to be routed to the console, all messages to the printer. (This is the same as the FLAG(I) option.)
- SP** causes only critical messages to be routed to the printer, all messages to the console.

MSGDD

$$\text{MSGDD} = \left\{ \begin{array}{l} \text{SYSOUT} \\ \text{xxxxxxxx} \end{array} \right\}$$

The program default for the DD name of the message data set is SYSOUT. To assign a different DD name, substitute any valid DD name for *xxxxxxxx*.

NULLOUT

$$\text{NULLOUT} = \left\{ \begin{array}{l} \text{RC0} \\ \text{RC4} \\ \text{RC16} \end{array} \right\}$$

NULLOUT specifies the action to be taken when SORTOUT in a sort, merge, or copy application contains no data records.

- RC0** The delivered default instructs SyncSort to issue a return code of 0 if not overridden by a higher return code set for another reason.
- RC4** Instructs SyncSort to issue a WER461I warning message and continue processing. A return code of 4 will be issued if not overridden by a higher return code set for another reason.

RC16 Instructs SyncSort to issue a WER461A message and terminate processing with a return code of 16.

Cannot be used with Tape Sort.

OVFLO

$$\text{OVFLO} = \left\{ \begin{array}{l} \text{RC0} \\ \text{RC4} \\ \text{RC16} \end{array} \right\}$$

OVFLO specifies the action to be taken if a summary field overflows or underflows during SUM processing.

RC0 The delivered default instructs SyncSort to issue a WER049I warning message and continue processing. A return code of 0 will be returned if not overridden by a higher return code set for another reason. The WER049I will only be issued on the first occurrence of the overflow or underflow.

RC4 Instructs SyncSort to issue a WER049I warning message and continue processing. A return code of 4 will be issued if not overridden by a higher return code set for another reason. The WER049I will only be issued on the first occurrence of the overflow or underflow.

RC16 Instructs SyncSort to issue a WER049A message and terminate processing with a return code of 16.

Cannot be used with Tape Sort.

PAD

$$\text{PAD} = \left\{ \begin{array}{l} \text{RC0} \\ \text{RC4} \\ \text{RC16} \end{array} \right\}$$

PAD specifies the action to be taken if the LRECL defined in the JCL for a non-OUTFIL SORTOUT is larger than the SORTIN/SORTINnn LRECL or the internally processed record length when the SORTIN/SORTINnn LRECL is modified by features.

RC0 The delivered default instructs SyncSort to issue a WER462I message, pad fixed-length output records with binary zeros, and issue a return code of zero.

RC4 Instructs SyncSort to issue a WER462I message and pad fixed-length output records with binary zeros. A return code of 4 will be issued if not overridden by a higher return code set for another reason.

RC16 Instructs SyncSort to issue a WER462A message and terminate processing with a return code of 16.

Note that for a BetterGener application PAD will be ignored. The installation parameter SOPADGN will control processing for these applications.

PAD will be ignored in applications in which the SORTIN/SORTINnn or SORTOUT is a VSAM data set.

Cannot be used with Tape Sort.

PRINT121

PRINT121

PRINT121 changes SyncSort's DCB default for the message data set to the following:

DCB=(LRECL=121,BLKSIZE=121,RECFM=FA)

This PARM is useful when the application includes a COBOL exit which uses DISPLAY, EXHIBIT, or TRACE instructions. (These macros will otherwise cause conflicts between program and sort/merge messages.) An alternative is provided by the MSGDD parameter, used to change the name of the SyncSort message data set.

The SyncSort program default for the message file's DCB is:

DCB=(LRECL=125,BLKSIZE=882,RECFM=VBA)

PRINT121 is automatically implemented for all program-initiated sort/merges.

Cannot be used with Tape Sort.

RC16

{ RC16=ABE } { <u>NORC16</u> }

RC16=ABE will cause the sort to issue user ABEND 16 instead of return code 16. The sort step is abnormally terminated without a dump and subsequent job steps are generally flushed by the operating system. However, JCL may specify job step(s) to be executed only in the event of an ABEND.

The delivered default is NORC16; unsuccessful completion of the sort causes a return code of 16 to be passed to the operating system or the invoking program.

Cannot be used with Tape Sort.

RELEASE

$$\text{RELEASE} = \left\{ \begin{array}{l} \text{ON} \\ \text{OFF} \end{array} \right\}$$

RELEASE=ON (the program default) turns on the RLSE operand in the SPACE parameter of the SORTWK DD statements. This will cause unused space to be released from sortwork units during execution time.

Specify RELEASE=OFF to instruct SyncSort to turn off the RLSE operand in the SPACE parameter of the SORTWK DD statement. In this case, SyncSort will not release unused space from the sortwork units during sort execution.

The RELEASE parameter has no effect on program-invoked sorts.

Cannot be used with Tape Sort.

RESERVE

$$\text{RESERVE} = \left\{ \begin{array}{l} \text{n} \\ \text{nK} \\ \text{nM} \end{array} \right\}$$

RESERVE sets aside a specified amount of memory *below* the 16-megabyte line for the user. This parameter takes effect only when CORE=MAX is in effect.

The memory may be specified as a decimal number of bytes (RESERVE=n), a decimal multiple of K, where K=1024 bytes (RESERVE=nK), or a decimal multiple of M, where M=1048576 bytes (RESERVE=nM).

Cannot be used with Tape Sort.

RESERVEX

$$\text{RESERVEX} = \left\{ \begin{array}{l} \text{n} \\ \text{nK} \\ \text{nM} \end{array} \right\}$$

RESERVEX overrides the installation value that reserves a specified amount of memory *above* the 16-megabyte line for the user. This parameter takes effect only when CORE=MAX is in effect.

The memory may be specified as a decimal number of bytes (RESERVEX=n), a decimal multiple of K, where K=1024 bytes (RESERVEX=nK), or a decimal multiple of M, where M=1048576 bytes (RESERVEX=nM).

Cannot be used with Tape Sort.

RESET

{ RESET NORESET }

RESET will prevent VSAM from treating output data sets as MOD data sets, if an output VSAM file was created using the REUSE option.

Cannot be used with Tape Sort.

RLSOUT

{ RLSOUT NORLSOUT }

RLSOUT releases all excess primary and secondary space from each output DASD file when the parm is specified and DISP=NEW is specified on output data set statements.

NORLSOUT is the program default. In this case, SyncSort does not release any excess space on these output files.

Cannot be used with Tape Sort.

SDB

SDB =	}	ON
		OFF
		YES
		NO
		DISKONLY
		TAPEONLY
		LARGE
		SMALL
		<u>INPUT</u>
		LARGEONLY
		INPUTONLY

SDB specifies whether system-determined blocksize should be used to select an optimal blocksize for output data sets when none is provided. This parameter will automatically provide a blocksize that will most efficiently utilize the space on the output device.

SDB=ON enables the use of system-determined blocksize for both tape and new or previously allocated but unopened DASD output data sets except in the following conditions:

- A blocksize is found in the JCL DCB BLKSIZE specification or, in the case of a DISP=MOD tape data set, it is derived from an available tape label.
- The output file is a VSAM data set.

If the output data set is on DASD, the blocksize selected will be based upon the RECFM and LRECL, either specifically provided or determined from the usual analysis of SORTIN or RECORD statement attributes. For example, the blocksize selected for a blocked output data set assigned to a 3380 or 3390 DASD device will represent a size as close to half-track blocking as possible.

If the output file is a tape data set, the blocksize will be determined from the RECFM and LRECL in conjunction with the following rules:

- RECFM of F or FS: BLKSIZE=LRECL
- RECFM of FB or FBS and LABEL type is not AL: BLKSIZE=highest multiple of LRECL that is less than or equal to 32760.
- RECFM of FB and LABEL type is AL: BLKSIZE=highest multiple of LRECL that is less than or equal to 2048.
- RECFM of V, VS, D: BLKSIZE=LRECL +4

- RECFM of VB, VBS: BLKSIZE=32760
- RECFM of DB: BLKSIZE=2048

If SDB=OFF is specified, SyncSort will not use system-determined blocksize. The blocksize, if unavailable, will be determined from SORTIN if the SORTIN and output data set LRECLs are the same, otherwise SyncSort will select an appropriate blocksize.

If SDB=DISKONLY is specified, SyncSort will use system-determined blocksize only for disk output data sets.

If SDB=TAPEONLY is specified, SyncSort will use system-determined blocksize only for tape output data sets.

SDB=LARGE enables the use of system-determined blocksize for both tape and DASD output data sets, as with SDB=ON. Additionally, under Version 2 Release 10 of OS/390 or under z/OS, SDB=LARGE enables selection of a system-determined blocksize greater than 32760 for eligible tape output data sets if not restricted by the system BLKSZLIM value.

SDB=SMALL has the same meaning as SDB=ON.

SDB=INPUT enables the use of system-determined blocksize for both tape and DASD output data sets, as with SDB=ON. Additionally, under Version 2 Release 10 of OS/390 or under z/OS, if an input tape data set has a blocksize greater than 32760, SDB=INPUT enables selection of a system-determined blocksize greater than 32760 for eligible tape output data sets if not restricted by the system BLKSZLIM value. SDB=INPUT is the default.

SDB=LARGEONLY enables the use of system-determined blocksize for tape output data sets only, as with SDB=TAPEONLY. Additionally, under Version 2 Release 10 of OS/390 or z/OS, SDB=LARGEONLY enables selection of a system-determined blocksize greater than 32760 for eligible tape output data sets if not restricted by the system BLKSZLIM value.

SDB=INPUTONLY enables the use of system-determined blocksize for tape output data sets only, as with SDB=TAPEONLY. Additionally, under Version 2 Release 10 of OS/390 or z/OS, if an input tape data set has a blocksize greater than 32760, SDB=INPUTONLY enables selection of a system-determined blocksize greater than 32760 for eligible tape output data sets if not restricted by the system BLKSZLIM value.

Cannot be used with Tape Sort.

SECOND

SECOND = $\left\{ \begin{array}{l} \text{ON} \\ \text{OFF} \end{array} \right\}$

SECOND=ON (the program default) provides automatic secondary allocation for all sort work data sets, even if it has not been specified in the sort work JCL. With SECOND=OFF, secondary allocation is only possible if it has been specified in the sort work JCL.

Cannot be used with Tape Sort.

SKIPREC

SKIPREC=n

SKIPREC=n instructs the sort to skip a decimal number *n* of records before sorting/copying the input file. The records skipped are deleted from the input file before E15 and INCLUDE/OMIT processing is begun.

If SKIPREC=n has been specified on the SORT/MERGE control statement as well as in the PARM field, the PARM specification will take precedence.

SKIPREC is not compatible with a merge application unless using FIELDS=COPY.

Cannot be used with Tape Sort.

STOPAFT

STOPAFT=n

STOPAFT=n (a decimal number) sorts/copies at most *n* records. These will be the first *n* records after any input processing due to an E15, an INCLUDE/OMIT statement, or the SKIPREC parameter.

If STOPAFT=n has been specified on the SORT/MERGE control statement as well as in the PARM field, the PARM specification will take precedence.

STOPAFT is not compatible with a merge application, unless using FIELDS=COPY.

Cannot be used with Tape Sort.

TRUNC

$$\text{TRUNC} = \left\{ \begin{array}{l} \underline{\text{RC0}} \\ \text{RC4} \\ \text{RC16} \end{array} \right\}$$

TRUNC specifies the action to be taken if the LRECL defined in the JCL for a non-OUTFIL SORTOUT is smaller than the SORTIN/SORTINnn LRECL or the internally processed record length when the SORTIN/SORTINnn LRECL is modified by features.

RC0 The delivered default instructs SyncSort to issue a WER462I message, truncate the output records, and issue a return code of zero.

RC4 Instructs SyncSort to issue a WER462I message and truncate the output records. A return code of 4 will be issued if not overridden by a higher return code set for another reason.

RC16 Instructs SyncSort to issue a WER462A message and terminate processing with a return code of 16.

Note that for a BetterGener application TRUNC will be ignored. The installation parameter SOTRNGN will control processing for these applications. TRUNC will be ignored in applications in which the SORTIN/SORTINnn or SORTOUT is a VSAM data set.

Cannot be used with Tape Sort.

UNINTDS

$$\text{UNINTDS} = \left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$$

UNINTDS indicates how SyncSort should process a non-VSAM uninitialized DASD SORTIN or SORTINnn data set in a non-SMS environment. An uninitialized data set is one that has been created but never successfully opened and closed for output. In an SMS environment, uninitialized data sets are always processed as valid empty files.

UNINTDS=YES indicates that an uninitialized data set should be processed as an empty file. If an uninitialized multi-volume data set has the DS1IND80 and DS1IND02 flags off in the format-1 DSCB of the first volume and the number of data extents is non-zero, SyncSort will open the data set for output to set an end-of-file mark before the data set is used for input.

UNINTDS=NO indicates that SyncSort should terminate with a WER400A critical message if an uninitialized data set is provided as input on SORTIN or SORTINnn.

Cannot be used with Tape Sort.

VLTEST

$$\text{VLTEST} = \left(\begin{array}{c} \{n\} \\ \{1\} \end{array} \left\{ \begin{array}{l} \text{,ON} \\ \text{,OFF} \\ \text{,OFF4} \end{array} \right\} \right)$$

VLTEST allows you to do the following when variable-length records are processed:

- Choose the type of record length validity testing to be performed.
- Choose whether or not to verify the correct sequence of segments in variable-length spanned records.

Record length validity testing may be performed in all types of applications: sort, merge, copy, and BetterGener. Segment sequence checking may only be done during sort and merge applications.

The first subparameter of the VLTEST PARM is a number n that instructs SyncSort in the type of validity testing to be performed on variable-length records. Choosing a validity test instructs SyncSort to terminate with a critical error (outlined in WER027A, WER160A or WER167A) in the event of an illegal condition.

In particular, VLTEST instructs the sort/merge in the handling of "short" variable-length records, i.e., records not long enough to contain all of the control field(s) specified in the SORT/MERGE control statement. The delivered default for VLTEST is 1.

When VLTEST is set to an even number, SyncSort will accept short variable-length records, *padding them with binary zeros* to the length of the sort key for the sort compare process. In order to prevent system 0C7 abends due to the binary zero padding, the CMP PARM is automatically set to CMP=CLC in these cases. The binary zeros are removed from the record, restoring it to its original state, as the output record is being written.

- | | |
|------------|---|
| 0** | No validity testing of variable-length records. |
| 1 | If input record is shorter than control fields, terminate. This is the delivered default. |
| 2** | If input record is longer than maximum LRECL or l_2 value, terminate. |
| 3 | If either or both of tests 1 and 2, terminate. |
| 4** | If input record is longer than output LRECL or l_3 value, terminate. |

- 5 If input record is shorter than control fields, or longer than output LRECL or l_3 value, or both, terminate.
- 6** If input record is longer than the maximum input LRECL or l_2 value, or longer than the output LRECL, or both, terminate.
- 7 If input record is shorter than control fields, or longer than input LRECL or l_2 value, or longer than output LRECL or l_3 value, or any or all of these, terminate.

The second subparameter allows you to specify whether or not SyncSort should verify that the sequence of segments is correct in each variable-length spanned record during sort and merge applications. ON is the delivered default and signals that the segment sequence should be verified. If OFF is selected, all illogical record segments encountered in the input file will be eliminated. If OFF4 is selected, the processing described for OFF will occur, but in addition if an illogical segment is found, message WER464I will be produced and a return code of 4 will be returned if not overridden by a higher return code set for another reason.

The second subparameter does not apply during copy applications.

Note: If an illegal condition is detected during a validity test and segment sequence checking is on, message WER182A will be issued.

Cannot be used with Tape Sort.

** *These values force the use of CMP=CLC for variable-length input.*

VLTESTI

$$VLTESTI = \begin{Bmatrix} 0 \\ 1 \\ 2 \end{Bmatrix}$$

VLTESTI specifies to SyncSort how to process variable-length records that do not contain all specified INCLUDE or OMIT fields. VLTESTI applies to both regular and OUTFIL INCLUDE/OMIT processing.

The delivered default of 0 instructs SyncSort to terminate if a record does not completely contain all INCLUDE or OMIT fields. A WER250A critical error message is generated to indicate this condition.

When VLTESTI=1 is specified, a record that does not completely contain all INCLUDE/OMIT fields is treated as having failed the comparison. SyncSort will omit the record if INCLUDE is being used or include the record if OMIT has been specified.

When VLTESTI=2 is specified, SyncSort will treat comparisons to fields not completely contained within the record as false and decide a record's status for inclusion or omission from fields that are available. If all fields are not present, the record will be processed as having failed the comparison. SyncSort will omit the record if INCLUDE is being used or include the record if OMIT has been specified.

Cannot be used with Tape Sort.

VSAMEMT

$\text{VSAMEMT} = \left\{ \begin{array}{l} \text{NO} \\ \text{YES} \end{array} \right\}$
--

VSAMEMT specifies the processing of empty VSAM input data sets.

If you specify VSAMEMT=YES, an empty VSAM data set will be processed as a legitimate data set containing 0 records, and SyncSort will end with a return code of 0.

The delivered default, VSAMEMT=NO, instructs SyncSort to terminate with a WER254A critical error if an empty VSAM data set is specified for input.

Cannot be used with Tape Sort.

ZDPRINT

$\left\{ \begin{array}{l} \text{ZDPRINT} \\ \text{NZDPRINT} \end{array} \right\}$

ZDPRINT specifies if positive ZD summation results are to be converted to printable numbers. ZDPRINT, the default, enables conversion to printable format. NZDPRINT prevents the conversion.

This option determines whether the sign byte of a positive summarized ZD field will be converted to a printable format. More precisely, the option specifies whether the zone of the last digit should be changed from a hexadecimal C to a hexadecimal F.

Cannot be used with Tape Sort.

PARAMs Accepted But Not Processed by Disk Sorts

The following parameters are accepted but not processed by disk sorts: BALN, OSCL, POLY, CRCX, and PEER.

Chapter 6. Invoking SyncSort from a Program

Programming Flexibility vs. Performance

The sort/merge can be invoked by an executing program written in COBOL, PL/1 or assembler language. However, since most invoked sorts utilize inline exits (typically through the COBOL SORT verb) and so are handicapped by the calling program's I/O techniques and memory allocations, sort performance may be degraded by this mode of initiation. Whenever performance is an important consideration, the sort should be initiated through the EXEC job control statement. Additional programming flexibility is provided by exits which can be separately compiled and link-edited. These may be coded in COBOL, FORTRAN, REXX, and assembler language. Exits may also be written in PL/1 provided that SyncSort is invoked by a PL/1 program.

DD Statements

The DD statements included in the Table of DD Statements for Invoked Sort/Merge are those which may be required when invoking the sort.

DD Statement	Usage	When not required
//\$ORTPARM DD	Used to override or add to control statements or PARMs supplied from an invoking program.	If no control statements or PARMs are being overridden or added.
//SORTIN DD	SORT input data set.	If there is an E32 exit routine or if running a merge.
//SORTIN _{nn} DD //SORTIN _n DD	MERGE input data sets.	If there is an E32 exit routine of if running a sort or copy.
//SORTWK _{xxx} DD //SORTWK _n DD	Work area definition. Defines the intermediate storage for a sort.	For incore sort, MERGE, COPY, or when using DYNALLOC.
//SORTOUT DD <i>or</i> //SORTOF _{xx} DD <i>or</i> //SORTOF _x DD	Output data set. Alternate output data set(s).	If there is an E35 exit routine. When no OUTFIL control statement is present.
//SORTXSUM DD	Output data set for records deleted by SUM.	When the XSUM parameter of the SUM control statement is not used.
//SORTLIB DD	Contains modules that are required to perform a Tape Sort.	If Tape Sort is not used.
//SYSOUT DD	Message data set.	When all messages are routed to console or are disabled.

Table 28. Table of DD Statements for Invoked Sort/Merge

Invoking the Sort/Merge from an Assembler Program

Assembler invocation is accomplished by means of the ATTACH, LINK, or XCTL macro instruction. SyncSort control statements are coded as character-string operands of Assembler DC operations. The calling program passes to the sort/merge a pointer containing the address of either a 24-bit or an extended, 31-bit, parameter list. This list contains the addresses of the control statement images to be used by the sort. It may also contain other information such as the addresses of E15, E32, and E35 exit routines. Note that when using either the 24-bit or the 31-bit parameter list, the control statement images can be passed from \$ORTPARM.

Macro Instructions

The choice of macro determines the linkage relationship between the calling program and the sort/merge load module. The linkage relationship established by ATTACH precludes the use of the Checkpoint-Restart feature; do not code CHKPT/CKPT on the SORT/MERGE control statement when invoking the sort/merge with the ATTACH macro. With XCTL, care must be taken to ensure that the storage area for the parameter list and other sort control information does not reside in the module issuing the macro - XCTL will delete this

module from memory. This problem can be circumvented in two ways. Either (1) place the parameter list and additional control information in the task that attaches the module issuing the XCTL; or (2) have the module issuing the XCTL macro issue a GETMAIN macro instruction first, and place all of the sort/merge control information in the main storage area it obtains. None of the above restrictions apply when using the LINK macro.

The sort/merge DD statements are placed with the JCL of the job step that issues the macro. The EP parameter is specified as SORT whether SyncSort is to be used for sorting, merging, or copying. With ATTACH, the ECB or EXTR is usually required.

Coding the Sort/Merge Control Statements

SyncSort control statements are introduced by the invoking programs as character operands in DC operations. Although it is generally true that all control statements supported for a JCL-initiated Disk Sort/MAXSORT/Tape Sort are available to invoked applications, these exceptions should be noted:

- A MODS control statement cannot be used for an E32 exit and is not required for an E15 or E35 exit. (An E32, E15 and/or E35 exit routine may be coded in line with the invoking program with their addresses passed to the sort in the parameter list. If the 31-bit parameter list is being used, an E18 and/or E39 exit routine address may also be passed.)
- The MODS control statement is not supported for an invoked Tape Sort.
- A RECORD control statement is required if an inline E15 exit routine address is provided. Its LENGTH parameter is required whenever an inline E15 or E35 exit routine is used. For a full description of record length parameters, see “RECORD Control Statement” on page 2.125.
- The END control statement is not used.

The actual coding of the control statements is the same, except that:

- No comments or labels are permitted within the DC operand.
- Continuation characters are not called for, since the statements are not in card-image format.

For the 24-bit parameter list, each control statement DC instruction should be labeled and followed by a DC C' ' instruction so that the beginning and ending addresses of the control statement can be referenced in the parameter list.

```

SORTBEG  DC  C' SORT FIELDS=(31,5,CH,A) '
SORTEND  DC  C' '
RECVBEG  DC  C' RECORD TYPE=F,LENGTH=(80,,120) '
RECVEND  DC  C' '

```

Figure 175. Sample Control Statement Images for the 24-bit Parameter List

Note that the 24-bit invoking parameter list and pointer word, as well as the control statement images, must be below the 16 megabyte line.

As in a JCL-initiated sort/merge, control statements must begin with at least one blank.

For the 31-bit invoking parameter list, the control statement images will be pointed to by the first word of the parameter list and are organized like the control statement images for a 24-bit parameter list. Note, however, that only the first DC instruction requires a label since only the start and end of the list need be referred to. The control statements must be separated by one or more blanks; that is, each control statement must be followed by a blank. A blank before the first statement is optional; however, a blank after each statement is required. Labels, comment statements, and comment fields must not be coded. Each control statement, except the last, must have at least one parameter.

```

CARDLEN  DC  0H
          DC  Y(CARDEND-CARDBEG)
CARDBEG  DC  C' SORT FIELDS=(31,5,CH,A) '
          DC  C' '
          DC  C' RECORD TYPE=F,LENGTH=(80,,120) '
          DC  C' '
CARDEND  EQU  *

```

Figure 176. Sample Control Statement Images for the 31-bit Parameter List

The 24-Bit Parameter List

The parameter list is used to pass information from the calling program (e.g., the addresses of the DC control statement images) to the sort/merge. For the parameter list used with invoked Tape Sorts, see “Chapter 12. Tape Sort”.

In order to pass the parameter list, it is necessary to load the address of a fullword pointer into Register 1. Code X'80' in the pointer's first byte and the address of the parameter list's byte count in its last three bytes. The byte count must be located in the last 2 bytes of the first fullword entry in the parameter list. It contains the hexadecimal number of bytes *remaining in the list* -- do not include these 2 bytes in the count.

The first seven words of the parameter list are required and must be coded in the exact order shown. Note that whenever the address of an exit routine is supplied in the sixth or seventh word of the parameter list, that exit may not be specified in the MODS control statement image. Parameter list entries following the seventh fullword are optional and can be specified in any order. All values not specified in the chart as EBCDIC-coded are to be given in hexadecimal notation. For the most part, these are addresses; the *ending* address refers to the blank coded immediately after the control statement in question, as indicated in the Sample Invoked Sort.

	Byte 1	Byte 2	Byte 3	Byte 4
Optional	X'09'	Beginning address of OUTREC statement		
		Ending address of OUTREC statement		
	X'0A'	Beginning address of INREC statement		
		Ending address of INREC statement		
	X'0B'	Beginning address of first OUTFIL statement		
		Ending address of first OUTFIL statement		
	.	.		
	.	.		
	.	.		
	X'0B'	Beginning address of nth OUTFIL statement		
		Ending address of nth OUTFIL statement		
	X'F6'	Beginning address of translation table		
	X'F7'	User exit address constant		
	X'FD'	IMS flag		
	X'FE'	Pointer to STAE work area (May code zeros if none)		
X'FF'	Message option (Code in EBCDIC)			
Optional	DIAG option (Code in EBCDIC)			
	BALN, OSCL or POLY (Processed Tape Sort only; code in EBCDIC)			
	CRCX, PEER or LIST (Not processed)			
	DD name prefix to replace SORT in JCL (Code in EBCDIC)			

Table 29. (Page 2 of 2) The 24-Bit Parameter List

Note: Empty boxes indicate the contents are immaterial and not examined.

.			
.			
.			
LA	1, POINTER		Load address of pointer to parameter list
ATTACH	EP= SORT		Initiate SyncSort
BR	14		Return to invoking program
.			
.			
.			
CNOP	2,4		Align to last 2 bytes of a word
BYTECNT	DC	Y(24)	Byte count of parameter list
	DC	A(SORTBEG)	Beginning address of sort statement
	DC	A(SORTEND)	Ending address of sort statement
	DC	A(RECBEG)	Beginning address of record statement
	DC	A(RECEND)	Ending address of record statement
	DC	2F'0'	No E15 or E32 indicated. No E35.
POINTER	DC	X'80'	Indicates parameter list's pointer
	DC	AL3(BYTECNT)	Address of parameter list
SORTBEG	DC	C' SORT FIELDS=(1,16,CH,A),SKIPREC=100'	
SORTEND	DC	C' '	
RECBEG	DC	C' RECORD TYPE=F,LENGTH=80'	
RECEND	DC	C' '	

Figure 177. Sample Invoked Sort

In this example, only the required seven entries appear in the parameter list. The invoked sort skips the first 100 records in SORTIN, a data set of 80-byte fixed-length records, and sorts the remainder in ascending sequence according to the character data in its first 16 bytes.

Additional parameter-list entries may appear in any order after these required seven; the contents of the first byte of the word signals which optional parameter it is.

For invoked merge applications using the 24-bit parameter list, the number of input files must be specified on *either* the X'04' entry *or* the FILES=n parameter on the MERGE control statement. However, when using the 31-bit parameter list, the number of input files for a merge must be specified on the FILES=n parameter.

Optional Parameters	
Code in Byte 1	Contents of Bytes 2 - 4
X'00'	Indicates how much main storage SyncSort is to use. Code C'MAX' or a hexadecimal number of bytes.

Table 30. (Page 1 of 3) Optional Parameters for the 24-Bit Parameter List

Optional Parameters	
Code in Byte 1	Contents of Bytes 2 - 4
X'01'	Indicates how much main storage should be reserved for data handling by the invoking program during sort execution. SyncSort will use all available main storage less the hexadecimal number of bytes specified here. This parameter takes precedence over the X'00' entry discussed above.
X'02'	Gives the beginning address of a MODS control statement image. The last 3 bytes of the next word must contain the ending address of this image. All exits may be specified in the MODS statement image except for E32 (use entry 6 of the parameter list for this exit). An E15/E35 exit is specified in entries 6-7 <i>or</i> in the MODS image, but not in both.
X'03'	Specifies the address of a replacement for the message data set's DD name. Eight characters are used for the name; the first character must be alphabetic.
X'04'	Specifies the hexadecimal number of SORTIN files-required whenever an E32 exit supplies the input to the merge.
X'05'	SyncSort accepts the DEBUG control statement parameter but does not process it. Both this fullword (the beginning address of the DEBUG statement) and the next (the ending address) are ignored.
X'06'	Gives the beginning address of an ALTSEQ control statement image. The last 3 bytes of the next word must contain the ending address of this image.
X'07'	Gives the beginning address of a SUM control statement image. The last 3 bytes of the next word must contain the ending address of this image.
X'08'	Gives the beginning address of an INCLUDE/OMIT control statement image. The last 3 bytes of the next word must contain the ending address of this image.
X'09'	Gives the beginning address of an OUTREC control statement image. The last 3 bytes of the next word must contain the ending address of this image.
X'0A'	Gives the beginning address of an INREC control statement image. The last 3 bytes of the next word must contain the ending address of this image.
X'0B'	Gives the beginning address of an OUTFIL control statement. The last 3 bytes of the next word must contain the ending address of this image. This parameter may be specified more than once to accommodate multiple output file specifications.
X'F6'	Specifies the address of a 256-byte translation table, used to alter the collating sequence. This parameter takes precedence over the ALTSEQ control statement image.

Table 30. (Page 2 of 3) Optional Parameters for the 24-Bit Parameter List

Optional Parameters	
Code in Byte 1	Contents of Bytes 2 - 4
X'F7'	Optional user exit address constant which can be used to pass information between the invoking program, an E15 exit, and/or an E35 exit. SyncSort passes these 3 bytes to an E15 exit at offset 5 in an E15 parameter list and to an E35 exit at offset 9 in an E35 parameter list. Offset 4 in the E15 parameter list and offset 8 in the E35 parameter list will initially contain an X'00'.
X'FD'	Only the first byte is processed, flagging this as an IMS-initiated sort, processing variable-length records too short to contain all of the sort/merge control field(s).
X'FE'	If non-zero, these 3 bytes contain the address of a 104-byte STAE work area.
X'FF'	Indicates the MSG/FLAG PARM coding. For the MSG option, specify AB, AC, AP, CB, NO, PC, SC or SP in bytes 3-4. For the FLAG option, specify NOF, (I) or (U) in bytes 2-4. This parameter is coded in EBCDIC.
DIAG coded in EBCDIC <i>in the entire word</i> turns on the IOERR=ABE and RC16=ABE PARM options.	
BALN/OSCL/POLY coded in EBCDIC <i>in the entire word</i> indicates the Tape Sort technique. Ignored by Disk Sort and MAXSORT.	
CRCX/PEER/LIST coded in EBCDIC <i>in the entire word</i> is accepted but not processed by SyncSort.	
xxxx with the first character alphabetic or national (do not use DIAG, PEER, CRCX, etc.) represents the DD name prefix to be used in place of SORT in the JCL.	

Table 30. (Page 3 of 3) Optional Parameters for the 24-Bit Parameter List

Return Codes

When the sort terminates, returning control to the calling program, it places a return code in Register 15:

- 0** indicates normal termination;
- 16** indicates an unsuccessful sort.

The calling program typically tests the contents of Register 15, branching to the normal-sort or sort-error end of job routine.

Sample Assembler Invocation Using 24-Bit Parameter List

.			
.			
.			
	LA	1,PTRWORD	Load address of parameter list pointer
	LINK	EP=SORT	Initiate SyncSort
	LTR	15,15	Test return code
	BNZ	SORTERR	Branch on error condition
	B	SORTOK	Branch to normal processing
	CNOP	0,4	Fullword alignment for pointer
PTRWORD	DC	X'80'	Indicates pointer to parameter list
	DC	AL3(PARMS)	Address of parameter list
	DS	H	Unused first 2 bytes of first parameter
PARMS	DC	Y(PARSEND-PARMSBEG)	Byte count of remaining parameters
PARMSBEG	DC	A(SORTBEG)	Beginning address of sort statement
	DC	A(SORTEND)	Ending address of sort statement
	DC	A(RECBEG)	Beginning address of record statement
	DC	A(RECEND)	Ending address of record statement
	DC	F'0'	No E15/E32 exit routine
	DC	F'0'	No E35 exit routine
	DC	X'03'	Indicates SYSOUT DD name change
	DC	AL3(MSGNAME)	Address of SYSOUT DD name replacement
	DC	X'08'	Indicates INCLUDE/OMIT parameter
	DC	AL3(OMITBEG)	Beginning address of OMIT statement
	DC	A(OMITEND)	Ending address of OMIT statement
	DC	X'07'	Indicates SUM parameter
	DC	AL3(SUMBEG)	Beginning address of SUM statement
	DC	A(SUMEND)	Ending address of SUM statement
	DC	X'0B'	Indicates OUTFIL parameter
	DC	AL3(OUTBEG1)	Beginning address of first OUTFIL statement
	DC	A(OUTEND1)	Ending address of first OUTFIL statement
	DC	X'0B'	Indicates OUTFIL parameter
	DC	AL3(OUTBEG2)	Beginning address of second OUTFIL statement
	DC	A(OUTEND2)	Ending address of second OUTFIL statement
PARSEND	EQU	*	End of parameter list
SORTBEG	DC	C' SORT FIELDS=(1,20,A,35,8,A),'	Begin SORT statement image
	DC	C'FORMAT=CH'	Continue SORT statement image
SORTEND	DC	C' '	End SORT statement image
RECBEG	DC	C' RECORD TYPE=F'	Begin RECORD statement image
RECEND	DC	C' '	End RECORD statement image
OMITBEG	DC	C' OMIT COND=(21,8,PD,EQ,0)'	Begin OMIT statement image
OMITEND	DC	C' '	End OMIT statement image
SUMBEG	DC	C' SUM FIELDS=(21,8,PD)'	Begin SUM statement image
SUMEND	DC	C' '	End SUM statement image
OUTBEG1	DC	C' OUTFIL FILES=1,'	Begin first OUTFIL statement image
	DC	C' HEADER1=(50X,'CHANGES'	
	DC	C' TO W-2 FORMS',/,',	
	DC	C'50X,'JANUARY THROUGH JUNE'	
	DC	C' 1993'')'	
OUTEND1	DC	C' '	End first OUTFIL statement image

Figure 178. (Page 1 of 2) Sample Assembler Invocation Using 24-Bit Parameter List

```

OUTBEG2 DC C' OUTFIL FILES=2,'      Begin second OUTFIL statement image
        DC C'HEADER1=(19X,'EMPLOYEE',
        DC C'10X,'DEPARTMENT CODE',
        DC C'10X,'CHANGE'),'
        DC C'OMIT=(29,4,PD,NE,0)'  OMIT condition
OUTEND2 DC C' '                      End second OUTFIL statement image
MSGNAME DC CL8'MESSAGES'            SYSOUT DD name replacement
        .
        .
SORTERR DS 0H                       Error routine for unsuccessful sort
        .
        .
        BR 14                        Return
SORTOK  DS 0H                       Normal processing for successful sort
        .
        .
        BR 14                        Return

```

Figure 178. (Page 2 of 2) Sample Assembler Invocation Using 24-Bit Parameter List

This example sorts fixed-length records by the character data in its first 20 bytes and, where two records have identical data in this field, by the character data in bytes 35-42; these fields are collated in ascending order. Note the continuation of the SORT statement image using consecutive DC instructions. There is no special significance to the break after the FIELDS parameter -- a control statement image can be divided at *any* point in this way. The SORTIN file is edited by the OMIT statement, which will eliminate any records with zero in bytes 21-28 before sorting begins; these 8 bytes constitute the SUM field. SyncSort messages are written to the data set specified by the MESSAGES DD name. Two OUTFIL parameters have been specified, producing multiple output files. The first OUTFIL will receive data from every sorted input record, producing a company-wide report. The second OUTFIL will receive selected data only, as defined by the OMIT condition, producing a departmental report.

The 31-Bit Extended Parameter List

The extended parameter list allows the sort to interface with invoking programs that may require 31-bit addresses or which may use the 31-bit addressing mode (AMODE).

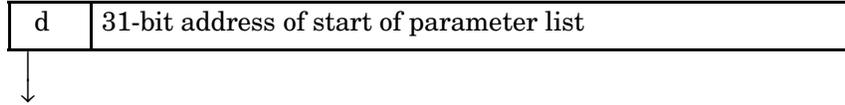
The extended parameter list is not supported for Tape Sorts. For the parameter list used with invoked Tape Sorts, see “Chapter 12. Tape Sort”.

Only the first word of the extended parameter list is required. The high order bit must be zero to identify this as a 31-bit parameter list. The subsequent words of this list are optional, and because there is no code in the high order byte, as in the 24-bit parameter list, their positional order must be maintained. Thus, when coding the list be sure to code a full-

word of zeros when omitting one of the optional parameters. The last parameter word specified in the list must be followed by the 4-byte field X'FFFFFFFF'.

The 31-bit parameter list has the following format:

REGISTER 1



		Bit 0	Bits 1 through 31
Required	+0	0	Address of halfword containing the length of control statement images (zeros if none)
Optional	+4	m	Address of user E15 or E32 (zeros if none)
	+8	m	Address of user E35 (zeros if none)
	+12		User exits address constant (zeros if none)
	+16	d	Address of ALTSEQ translation table (zeros if none)
	+20	d	Address of STAE area field (zeros if no STAE routine)
	+24	m	Address of user exit E18 (zeros if none)
	+28	m	Address of user exit E39 (zeros if none)
	+32		Call identifier (C 'nnnn')
Required	+36		X'FFFFFFFF' (required)

Table 31. 31-Bit Extended Parameter List

Note: d indicates a bit is immaterial and not examined.

The following table provides an explanation of the contents of the extended parameter list.

Address	Contents
+0	(Required) First word of the parameter list. The high order bit must be zero to identify this as an extended parameter list. The other 31 bits contain the address of a halfword which contains the length of the following control statement images. A value of 0 (zero) represents a null list and control statement images must be supplied through the \$ORTPARM DD statement.
+4	(Optional) Address of the E15 or E32 exit routine. This address may point anywhere in memory. The m bit (bit 0) means: 0=Enter the exit with 24-bit addressing in effect (AMODE 24); 1=Enter the exit with 31-bit addressing in effect (AMODE 31).
+8	(Optional) Address of the E35 exit routine. This address may point anywhere in memory. The m bit (bit 0) means: 0=Enter the exit with 24-bit addressing in effect (AMODE 24); 1=Enter the exit with 31-bit addressing in effect (AMODE31).
+12	(Optional) User exit address constant which can be used to pass information between the invoking program, an E15 exit routine, and/or an E35 exit routine. SyncSort passes these 4 bytes to an E15 exit routine at offset 4 in an E15 parameter list and/or to an E35 exit routine at offset 8 in an E35 parameter list.
+16	(Optional) Address of ALTSEQ translation table. It can point anywhere in memory and has a length of 256 bytes.
+20	(Optional) If non-zero, the address of a 112-byte STAE work area.
+24	(Optional) Address of the E18 exit routine. This address may point anywhere in memory. The m bit (bit 0) means: 0=Enter the exit with 24-bit addressing in effect (AMODE 24); 1= Enter the exit with 31-bit addressing effect (AMODE 31).
+28	(Optional) Address of the E39 exit routine. This address may point anywhere in memory. The m bit (bit 0) means: 0=Enter the exit with 24-bit addressing in effect (AMODE 24); 1= Enter the exit with 31-bit addressing in effect (AMODE 31).
+32	(Optional) Four displayable characters that will uniquely identify this particular call to SyncSort. (When this parameter is provided, SyncSort produces the WER428I message and includes these four characters in the text. This message facilitates correlation of message output when SyncSort is called multiple times by the same program.)
+36	(Required) The last parameter word specified in the list must be X'FFFFFFFF', which indicates end of list. If optional entries are omitted, this end-of-list indicator is moved up to the word immediately after the last specified parameter.

Table 32. Explanation of the Contents of the 31-Bit Extended Parameter List

Note: An optional parameter becomes required if a subsequent parameter is to appear.

Return Codes

When the sort terminates, returning control to the calling program, it places a return code in Register 15:

- 0** indicates normal termination;
- 16** indicates an unsuccessful sort.

The calling program typically tests the contents of Register 15, branching to the normal-sort or sort-error end of job routine.

The following examples demonstrate how to code an extended parameter list. In this example, all exits reside below the 16-megabyte line and should be called with 24-bit AMODE set, except the E35 exit, which should be called with 31-bit AMODE set.

```

      .
      .
      .
      LA 1,XLIST                               Point at Parameter List
      LINK EP=SORT                             Initiate SyncSort
      .
      .
      .
XLIST DC A(CNTLCARD)                          Address of Control Card Images
      DC A(E15EXIT)                            Address of E15 Exit
      DC A(E35EXIT+X'80000000')              Address of E35 Exit
      DC F'0'                                  User Address Constant
      DC A(ALTSEQ)                            Address of ALTSEQ
                                          Translation Table
      DC A(STAE)                              Address of STAE Area Field
      DC A(E18EXIT)                           Address of E18 Exit
      DC A(E39EXIT)                           Address of E39 Exit
      DC X'FFFFFFFF'                          End of Parameter List
CNTLCARD DC 0H
      DC Y(CNTLLEN)
CNTLCRD2 DC C' SORT FIELDS=(1,16,CH,A) '
      DC C' RECORD TYPE=F,LENGTH=80 '
CNTLLEN EQU *-CNTLCRD2
```

Figure 179. Sample Invoked Sort with Both 24-bit AMODE & 31-bit AMODE Set

This next example demonstrates how to code an extended parameter list in which all exits reside above the 16-megabyte line and should be called with AMODE 31 set.

```

.
.
.
LA 1,XLIST Point at Parameter List
LINK EP=SORT Initiate SyncSort
.
.
.
XLIST DC A(CNTLCARD) Address of Control Card Images
DC A(E15EXIT+X'80000000') Address of E15 Exit
DC A(E35EXIT+X'80000000') Address of E35 Exit
DC F'0' User Address Constant
DC A(ALTSEQ) Address of ALTSEQ Translation Table
DC A(STAE) Address of STAE Area Field
DC A(E18EXIT+X'80000000') Address of E18 Exit
DC A(E39EXIT+X'80000000') Address of E39 Exit
DC X'FFFFFFFF' End of Parameter List
CNTLCARD DC 0H
DC Y(CNTLLEN)
CNTLCRD2 DC C' SORT FIELDS=(1,16,CH,A) '
DC C' RECORD TYPE=F,LENGTH=80 '
CNTLLEN EQU *-CNTLCRD2

```

Figure 180. Sample Invoked Sort with 31-bit AMODE Set

Sample Assembler Invocation Using 31-Bit Parameter List

```

.
.
.
LOAD EP=E15EXIT          Load E15 exit
ST  R0,E15ADDR          Store E15 AMode+address
LA  1,XLIST             Point at parameter list
LINK EP=SORT            Initiate SyncSort
LTR  R15,R15           Test return code
BNZ  SORTERR           Branch on error condition
B    SORTOK            Branch to normal processing
.
.
.
XLIST  DC  A(CARDLEN)    Address of control statements
E15ADDR DC  A(0)         Address of E15 routine
        DC  A(0)         No E35 routine
        DC  A(0)         User exit address constant
        DC  X'FFFFFFFF'   End of parameter list
CARDLEN DS  0H          Control statement area
        DC  Y(CARDEND-CARDBEG) Length of character string
CARDBEG DC  C'SORT FIELDS=(1,20,A,35,8,A),' Begin SORT image
        DC  C'FORMAT=CH'  Continue SORT image
        DC  C'RECORD TYPE=F,LENGTH=80 ' RECORD image
        DC  C'OMIT COND=(21,8,PD,EQ,0) ' OMIT image
        DC  C'SUM FIELDS=(21,8,PD) ' SUM image
        DC  C'OUTFIL FILES=1,' First OUTFIL image
        DC  C'HEADER1=(50X,'CHANGES'
        DC  C' TO W-2 FORMS',/,/, '
        DC  C'50X,'JANUARY THROUGH JUNE'
        DC  C'1992''')' End first OUTFIL image
        DC  C'OUTFIL FILES=2,' Second OUTFIL image
        DC  C'HEADER1=(19x,'EMPLOYEE', '
        DC  C'10X,'DEPARTMENT CODE', '
        DC  C'10X,'CHANGE''')'
        DC  C',OMIT=(29,4,PD,NE,0) ' End second OUTFIL image
CARDEND EQU  *
SORTERR DS  0H          Error routine for unsuccessful sort
.
.
.
BR  14                 Return
SORTOK DS  0H          Normal processing for successful sort
.
.
.
BR  14                 Return

```

Figure 181. Sample Assembler Invocation Using 31-Bit Parameter List

This example sorts fixed-length records by the character data in its first 20 bytes and, where two records have identical data in this field, by the character data in bytes 35-42; these fields are collated in ascending order. Note the continuation of the SORT statement image using consecutive DC instructions. There is no special significance to the break after the FIELDS parameter - a control statement image can be divided at *any* point in this way. The SORTIN file is edited by the OMIT statement, which will eliminate any records with

zero in bytes 21-28 before sorting begins; these 8 bytes constitute the SUM field. Two OUTFIL parameters have been specified, producing multiple output files. The first OUTFIL will receive data from every sorted input record, producing a company-wide report. The second OUTFIL will receive selected data only, as defined by the OMIT condition, producing a departmental report.

Chapter 7. The Coding and Use of Exit Programs

What Is an Exit?

The term *program exits* refers to the various points in the sort program's executable code at which control can be passed to a user-written routine. Most exit routines take control once for every record being processed, increasing overall execution time and consuming main storage that would otherwise be used by the sort. Exits should only be coded for tasks which cannot be accomplished with SyncSort control statements.

Program exits are not allowed to take their own OS or VS checkpoints.

Program exits are labeled with a 2-digit decimal number, e.g., E35. Except for E61, the first digit (1, 2 or 3) refers to the sort/merge phase at which the routine will get control; an E61 routine can take control in Phase 1 or Phase 3. The second digit refers to the number of that exit within the phase. Whenever possible, control passes directly from Phase 1 to Phase 3, skipping the intermediate merge phase and its associated exits: E21, E25 and E27.

As indicated in the following chart, the nature of the task determines the program exit to be used.

TASK	PHASE 1							PHASE 2			PHASE 3							
	E11	E14	E15	E16	E17	E18	E61	E21	E25	E27	E31	E32	E15*	E35	E37	E38	E39	E61
Prepare for other exit routines	X							X			X							
Create input records for sort (Phase 1) and copy (Phase 3)			X										X					
Create input records for merge											X							
Add records			X										X	X				
Delete records		X	X						X				X	X				
Change records		X	X						X				X	X				
Sum records		X							X					X				
Choose action if intermediate storage insufficient				X														
Close other exit data sets					X					X					X			
Process read errors						X										X		
Process write errors																	X	
Check labels						X										X	X	
Modify a collating sequence							X											X

* E15 in Phase 3 for copy only

Table 33. Program Exits and Processing Phases

Loading the Exit Routines into Main Storage

The MODS statement identifies the exits to be taken and indicates the name of the separately compiled, user-written routine to take control at that point. The same routine (e.g., deleting selected records) could take effect in different phases, but cannot be loaded more than once in a single phase.

Note that merge and copy are executed entirely in Phase 3 and are therefore restricted in the exits which they can use. A merge application cannot use exits E11 through E27. A copy application can use exit E15 but not exits E32 or E61.

Assemble each routine as a separate program and place it in a partitioned data set or in the SYSIN input stream; SyncSort copies the SYSIN routines to the SORTMODS library for linkage editing. (If a SYSIN module is to be used at more than one exit point, each exit must have its own compiled copy of the module in SYSIN.) If SyncSort linkage edits an exit routine, the module must have an entry point whose name is that of the SyncSort exit; for example, in order to function as an E35 routine, MYEXIT must include an entry point or CSECT labeled E35.

If a routine has already been link-edited, this can be indicated in the MODS statement. When all the exits in a particular phase need to communicate with one another, the MODS statement can be used to instruct the sort to link-edit them together.

Exit Conventions

The following conventions must be observed when using exits.

- Exits provided via the MODS control statement will be entered in the addressing mode indicated by the linkage editor module attributes. Any exit linkage-edited by SyncSort will be entered in 24-bit addressing mode, except a separately linkage-edited exit E11, E21, or E31, which will be entered in the mode set by the compiler or assembler when the module was compiled or assembled.
- Exit addresses provided via the 24-bit invoking parameter list format will be entered in the 24-bit address mode.
- Exit addresses provided via the 31-bit invoking parameter list will be entered in the address mode indicated in the exit address field. That is, if bit 0 of the exit address is 0, the exit is entered in 24-bit mode; if bit 0 of the exit address is 1, the exit is entered in 31-bit mode.
- User exits may return to the sort in either 24- or 31-bit address mode.
- If an exit was entered in 24-bit address mode, the addresses passed to it will be 24-bit values that have a clean high-order byte containing binary zeros (X'00'). Addresses returned to the sort must also be 24-bit values with a high-order byte containing X'00' even though the exit could return to the sort in the 31-bit mode.

- An exit in the 31-bit mode may return an address containing a full 31-bit value. Users intending to pass only a 24-bit address must therefore make sure that the address returned has X'00' in the high-order byte. Failure to do so can have unpredictable results. Note that certain addresses within parameter lists are still explicitly restricted to 24-bit values. For example, E18 exit return parameter lists must consist of fullword entries that are 1-byte codes and 3-byte addresses.

Register Conventions

The standard operating system conventions apply to register usage. Exit routines must save and restore Registers 0 and 2-14. The sort/merge places these contents in Register 1 and 13-15 for use by the exit routine when it takes control.

Register 1	The address of a SyncSort parameter list.
Register 13	The address of a 19-word area. The first 18 words can be used to save registers, the 19th word to pass information between Assembler exits.
Register 14	SyncSort's return address, in the low-order address bits of the register. The high-order bit(s) may have undefined contents.
Register 15	The address of the entry point of the exit routine, in the low-order address bits of the register. The high-order bit(s) may have undefined contents.

The Exit Communication Area

When an exit routine is given control, Register 13 points to a 19-word area, the first 18 of which can be used to save registers. The 19th word of this area can be used to pass information between Assembler exits. For example, when the COMMAREA PARM is used, the 19th word can be set to point to the exit communication area COMMAREA provides. The first 2 bytes of this communication area give the length of the area. The user is free to change the entire communication area, including the initial halfword.

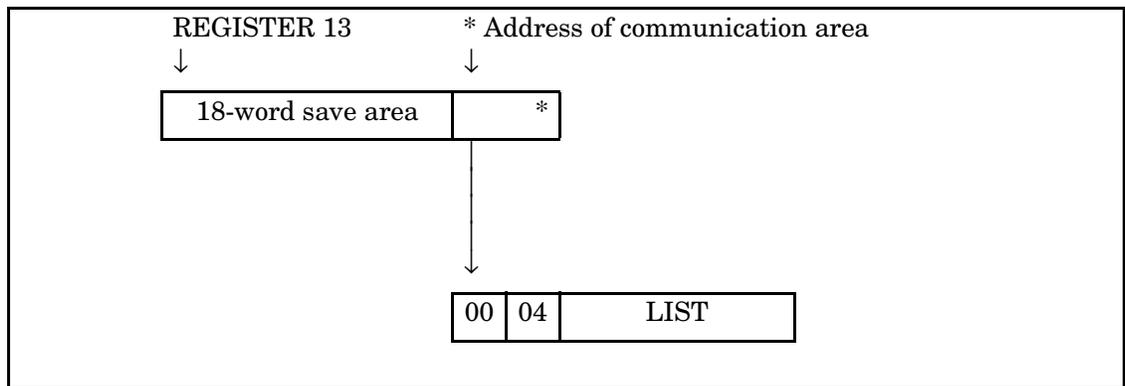


Figure 182. User Communication Area for Assembler Exit Using COMMAREA PARM

For COBOL or C exits, the address and length of this area are passed in the COBOL or C program's parameter list. In this case, there is no halfword preface - the address points directly to the communication area.

Exits E11, E21, and E31 - Preparing for Other Exit Routines

These exits are unusual in that they are entered only once, at the beginning of their associated phase. Because of this, they may be separately link-edited and are efficiently used to prepare for other exit routines (e.g., to open files or initialize variables). There are no parameter lists or return codes for these exits.

Exit E32 - Invoked Merge Only: Creating Input Records

This exit can only be used for an invoked merge and must be coded in line with the invoking program. It therefore never appears on the MODS statement. When an E32 routine is used, all SORTINnn DD statements will be ignored by the merge; the exit must supply all the input records, and the number of input files to be created must be supplied by either the invoking program's parameter list or the FILES=n parameter on the MERGE control statement.

Whenever the merge requires a new input record, SyncSort calls the E32 routine, passing it the address of a two-word parameter list in Register 1.

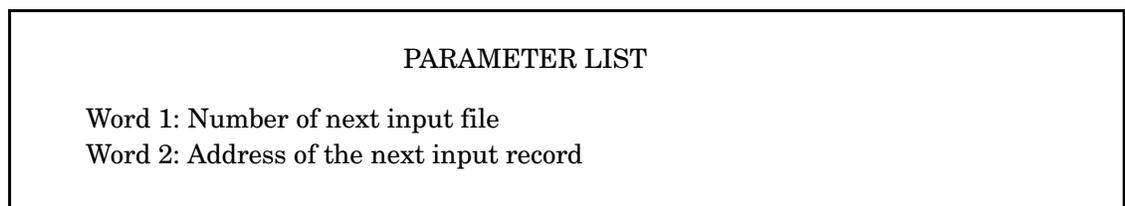


Figure 183. Parameter List for E32

The first word of the parameter list contains a hexadecimal representation of the input file SyncSort is currently processing. This is initialized as 0 for the first file and incremented by 4 every time a new SORTINnn file is to be accessed. When the E32 encounters end-of-file on a SORTINnn file it should return RC=8 to SyncSort, which will no longer request input from that file (i.e., that input file number).

The E32 routine must respond to three different cases: (a) SyncSort already has all the input records; (b) the previous record finished an input file; and (c) there is at least one more record to be added to the file with this file number. Only in the last case will the E32 supply a record address to the merge, placing it in the second word of the parameter list. The E32 also places the appropriate return code in Register 15.

Return Codes

- | | |
|-----------|--|
| 8 | <i>End of file.</i> This tells SyncSort that a particular file has been completed and to make no further request for records from that file. |
| 12 | <i>Insert this record.</i> This tells SyncSort to accept a new record from the input file requested. |
| 16 | <i>End of merge.</i> This terminates SyncSort with a critical error. |

Exits E14, E15, E25, and E35 - Deleting, Creating, Changing Records

Exit E14 - Deleting, Summarizing, Changing Records

Exit E14 may be used to change the contents of data fields, or to delete or summarize records during Phase 1. Unlike an E15 exit routine, it cannot be used to add records. An E14 exit program requires:

- at least one SORTWKxx data set, assigned to disk;
- fixed-length input records.

This exit is given control whenever SyncSort is about to *add* a record to an output sequence. Since it does not take control before the first record of that sequence, the routine always has access to a pair of sequenced records (e.g., for summarization purposes). SyncSort passes the exit program a two-word parameter list by loading its address into Register 1. The exit must not destroy the contents of this parameter list. The first word, which is on a fullword boundary, contains the address of the record about to be placed in an output sequence; the second word contains the address of the record that has just been put into the output buffer.

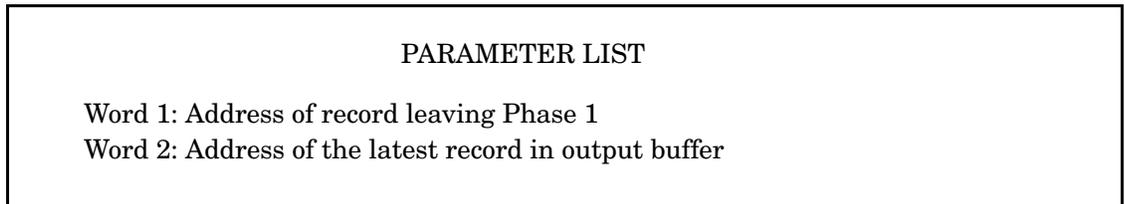


Figure 184. Parameter List for E14

There are two constraints on the type of processing the exit may accord this record pair:

- Sort control fields should *not* be changed since this may cause an out-of-sequence condition.
- If a record is to be changed, it should first be moved to a work area.

After record processing is completed, the exit routine must place the appropriate return code in Register 15. The exit must save and restore all registers except those used in linking to the sort/merge.

Return Codes

- 0** *Accept this record.* This instructs SyncSort to accept the record whose address is in the first word of the parameter list and place it in the output

buffer. The exit must also load the (work area) address of this record into Register 1 before returning control to the sort.

- 4 *Delete this record.* This instructs SyncSort to delete the record whose address is in the first word of the parameter list. Do *not* place the address of this record in Register 1. This return code might be employed, for example, after using this record to update the previous (output) record. Assuming this does not complete an output sequence for Phase 1, the next execution of the E14 will find the same address in the second word of the parameter list.

Exit E15 - Creating, Revising or Analyzing the Input File

Where an input data set already exists, this exit is used to add, delete and/or change input records. This exit is also used to analyze SORTIN via HISTOGRM (a HISTE15 application) or to create the entire input file. It can be used when sorting or copying records.

When used in conjunction with an input file, this exit is given control every time a record is brought into Phase 1 of a sort or Phase 3 of a copy. In passing control to the E15 exit routine, SyncSort places the address of a parameter list in Register 1. This parameter list is two words long, aligned on a fullword boundary. In the first word, the first byte contains X'00'; the last 3 bytes contain the address of the record just brought into Phase 1. The first word contains a zero address when there is no such record (i.e., when SORTIN end-of-file is reached or when the input data set is empty).

Word 2 contains the user address constant. On the initial call to the E15 exit, it will contain the value specified in the invoking parameter list. If this value was specified in a 24-bit invoking parameter list, it will have the high-order byte set to X'00'. If the value was omitted or SyncSort was JCL invoked, Word 2 will contain binary zeros. This word may be changed by the E15 exit whenever it is entered. If used in a sort application, the value will be returned on the subsequent call to the E15. If used in a copy application and an E35 is present, the value on the subsequent call to the E15 will reflect any modification made to the User Address Constant by the E35. In a sort application, the initial entry to the E35 will contain the value last returned from the E15.

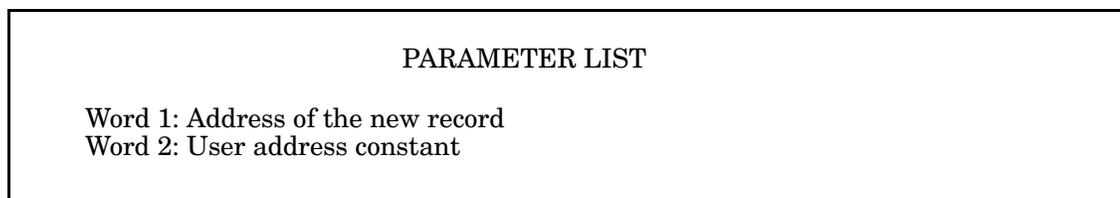


Figure 185. Parameter List for E15

E15 record processing has these two constraints:

- If a record is to be changed, it should first be moved to a work area.

- When the input data set consists of variable-length records, the first 4 bytes must contain the Record Descriptor Word, giving the length of the record.

When the program has finished processing the record, it must place the appropriate return code in Register 15.

Coding the E15 Exit Routine for an Invoked Sort or Copy

When SyncSort is initiated from an ATTACH, LINK or XCTL macro, there are two ways to include an E15 exit routine: (1) code the E15 exit routine in line with the invoking program and specify the address of its entry point in the appropriate entry of the parameter list; or (2) define the separately compiled routine in the MODS control statement. When the exit routine is coded in line with the calling program, it must supply the entire input data set; SyncSort will ignore a SORTIN DD statement, if present. Data set creation is done by supplying the sort with one record at a time, placing its address in Register 1 and a return code of 12 in Register 15. After the last record has been submitted, the exit passes a return code of 8.

Return Codes

- | | |
|-----------|--|
| 0 | <i>Accept this record.</i> This instructs SyncSort to accept the record the exit has just examined. Place the (work area) address of this record in Register 1. This return code is used when selectively editing records from an input file; it passes the (possibly altered) record back to the sort. The RECORD statement is required if the exit routine changes the maximum record length; code the old maximum length as l_1 , the new maximum as l_2 . |
| 4 | <i>Delete this record.</i> SyncSort will delete the record just examined. There is no need to load the address of this record into Register 1. |
| 8 | <i>Do not return to this exit.</i> This instructs SyncSort to close the exit for the remainder of the sort application. This return code might be used at SORTIN end-of-file (signalled by a zero address in the parameter list) to indicate that extra records will not be added at this point. There is no need to load a record address into Register 1 when passing a return code of 8. If SORTIN is present, the current input record and all subsequent records will be processed by SyncSort. |
| 12 | <i>Insert a record.</i> This tells SyncSort that the exit routine has located a record which should be added to the input data set before the record whose address appears in the parameter list. Load the address of the new record into Register 1. When SyncSort returns control to the E15, the parameter list will be unchanged. The exit routine can then add another record or process the current one. |

This return code can be used to add records to the end of the input data set or to create the entire input data set. SyncSort returns to the exit routine,

adding records without changing the parameter list (in these cases, a zero address) until a different return code (i.e., RC=8) is passed. When the input data set is created in this way, the RECORD statement is required and must specify both TYPE and LENGTH.

- 16** *Terminate SyncSort.* This tells SyncSort to terminate and return to the calling program or the supervisor. SyncSort uses a completion code of 16 to indicate that the sort was unsuccessful.

Coding a COBOL E15 Exit Routine

A COBOL E15 exit program can be indicated through the EXEC statement's PARM option (PARM='E15=COB'), the MODS control statement or the \$ORTPARM DD statement. An E15 exit cannot be coded in COBOL when running Tape Sort.

Like any other E15 exit routine, the COBOL E15 exit routine is called each time a record is brought into Phase 1 of a sort or Phase 3 of a copy. Communication between SyncSort and the COBOL exit takes place in the LINKAGE SECTION of the COBOL program. For example, records are passed to the COBOL routine in the second definition (RECORD-UP) area of the LINKAGE SECTION.

If the COBOL exit routine uses any verb (EXHIBIT, DISPLAY, TRACE) which results in output to the SYSOUT DD statement, there is a potential conflict with SyncSort's use of this DD statement. It is therefore recommended that the user separate the output by using either SyncSort's MSGDD PARM option or the COBOL compiler's SYSx parm.

If the COBOL E15 exit routine is written in OS/VS COBOL and uses OS/VS COBOL libraries or no libraries, specify the COBEXIT=COB1 PARM option. If the COBOL E15 exit routine is written in OS/VS COBOL or in VS COBOL II or COBOL/370 and uses VS COBOL II or COBOL/370 libraries, specify the COBEXIT=COB2 PARM option. VS COBOL II and COBOL/370 run-time library modules typically require more main storage than OS/VS COBOL library modules. The amount of additional storage depends on VS COBOL II of COBOL/370 installation options. When VS COBOL II or COBOL/370 run-time library modules are used, it may be necessary to account for this additional storage by adjusting the b value of the Exit-Name parameter on the MODS statement.

The LINKAGE SECTION

The LINKAGE SECTION examples that follow show the parameters required for passing fixed-length and variable-length records to the sort. The data-names and conditional names used in the examples are arbitrary but each definition is required. The complete programs from which the examples are taken follow the discussion of the exit.

Example 1: Fixed-Length Records

```
LINKAGE SECTION.  
01  EXIT-STATUS                               PIC 9 (8) COMPUTATIONAL.  
    88  FIRST-TIME                             VALUE 00.  
    88  MOST-TIME                              VALUE 04.  
    88  LAST-TIME                              VALUE 08.  
01  RECORD-UP.  
    07  FILLER                                 PIC 9(6).  
    07  R-SEQ2                                 PIC 9(2).  
    07  FILLER                                 PIC X(92).  
01  WORK                                       PIC X(100).  
01  DUMMY1                                    PIC X.  
01  DUMMY2                                    PIC X.  
01  DUMMY3                                    PIC X.  
01  DUMMY4                                    PIC X.  
01  DUMMY5                                    PIC X.  
  
01  COMM-LEN                                  PIC 9(4) COMPUTATIONAL.  
  
01  COMMUNICATION-AREA.  
  
    05  COMM-AREA OCCURS 1 TO 256 TIMES  
        DEPENDING ON COMM-LEN PIC X.
```

- For the first definition (EXIT-STATUS) specify PIC 9(8) COMPUTATIONAL. (This area defines exit status codes.) When using 88 levels to define the exit status codes, specify values 00, 04, and 08.
- For the second definition (RECORD-UP) define the SORTIN record.
- For the third definition (WORK) define the record that will be passed to SyncSort. (This is the "work area.")
- For the fourth through the eighth definitions define dummy areas.
- For the ninth definition (COMM-LEN) specify PIC 9(4) COMPUTATIONAL. This area defines the communication area length.
- For the tenth definition (COMMUNICATION-AREA) code an OCCURS clause DEPENDING ON *data-name* PIC X.

Example 2: Variable-Length Records

```
LINKAGE SECTION.
01  EXIT-STATUS                PIC 9 (8) COMPUTATIONAL.
    88  FIRST-TIME              VALUE 00.
    88  MOST-TIME               VALUE 04.
    88  LAST-TIME               VALUE 08.
01  RECORD-UP.
    05  RU      OCCURS 1 TO 100 TIMES
           DEPENDING ON LEN-RU      PIC X.
01  WORK.
    05  WK      OCCURS 1 TO 100 TIMES
           DEPENDING ON LEN-WK      PIC X.
01  IN-BUF                PIC X(100) .
01  DUMMY                  PIC X(4) .
01  LEN-RU                 PIC 9(8)   COMPUTATIONAL.
01  LEN-WK                 PIC 9(8)   COMPUTATIONAL.
01  LEN-IB                 PIC 9(8)   COMPUTATIONAL.
01  COMM-LEN               PIC 9(4)   COMPUTATIONAL.
01  COMMUNICATION-AREA.
    05  COMM-AREA OCCURS 1 TO 256 TIMES
           DEPENDING ON COMM-LEN PIC X.
```

- For the first definition (EXIT-STATUS) specify PIC 9(8) COMPUTATIONAL. (This area defines exit status codes.)
- For the second definition (RECORD-UP) code an OCCURS clause with the DEPENDING ON *data-name* option specifying (1) The minimum and maximum number of bytes the variable SORTIN records contain (do *not* include 4 bytes for the RDW) and (2) DEPENDING ON *data-name* PIC X. *Data-name* is defined in the sixth definition in the LINKAGE SECTION.
- For the third definition (WORK) code an OCCURS clause with the DEPENDING ON *data-name* option specifying (1) The minimum and maximum number of bytes for variable-length records to be passed to SyncSort (do *not* include 4 bytes for the RDW) and (2) DEPENDING ON *data-name* PIC X. *Data-name* is defined as the seventh definition in the LINKAGE SECTION.
- For the fourth definition specify a dummy level 01 *data-name* of any number of bytes. (IN-BUF is the *data-name* used in this example.) Note that the level 01 *data-name*, used here as a dummy address, has no effect on the E15 routine for variable-length records. The address is usually used as a buffer pointer in the COBOL E35 exit routine. By using it in the E15 LINKAGE SECTION, SyncSort is able to use the same parameter list for both COBOL exits E15 and E35.

- For the fifth definition specify a dummy area.
- For the sixth definition (LEN-RU) specify *data-name* PIC 9(8) COMPUTATIONAL. This is where SyncSort passes the length of the SORTIN record to the COBOL exit.
- For the seventh definition (LEN-WK) specify *data-name* PIC 9(8) COMPUTATIONAL. This is where the E15 routine passes the length of the work area record to SyncSort.
- For the eighth definition define a dummy area.
- For the ninth definition (COMM-LEN) specify PIC 9(4) COMPUTATIONAL. This area defines the communication area length.
- For the tenth definition (COMMUNICATION-AREA) code an OCCURS clause DEPENDING ON *data-name* PIC X.

The IDENTIFICATION, ENVIRONMENT, and DATA Divisions

As always, the COBOL program must contain the entries required by the compiler for these program divisions. Code the optional entries in these divisions according to the requirements of the application.

The WORKING-STORAGE SECTION

If the exit routine inserts records into the final merge and replaces records passed from SyncSort, the insertion record and the replacement record may be defined in this section. These records will be moved to the WORK area described in the LINKAGE SECTION, so be sure that the PICTURE clause or the OCCURS clause in the WORK area is correct for these records.

This section may also define the return codes as 77-level data items. Alternatively, these codes can be specified as literals in the MOVE instruction. (MOVE *literal* to RETURN-CODE.) Note that RETURN-CODE is the name of a predefined storage area in COBOL used to pass return codes to the sort; RETURN-CODE should not be defined in the exit routine.

The PROCEDURE DIVISION

Specify the USING option on the PROCEDURE DIVISION header. Each identifier specified after USING must be the same as those described in the 01-level of the LINKAGE SECTION. Taking for example the identifiers defined in the fixed-length record LINKAGE SECTION shown here, they would appear as: PROCEDURE DIVISION USING EXIT-STATUS, RECORD-UP, WORK, DUMMY1, DUMMY2, DUMMY3, DUMMY4, DUMMY5, COMM-LEN, COMMUNICATION-AREA.

The GOBACK statement is used to return control to SyncSort. Do *not* use the EXIT statement as it will cause unpredictable results. Be sure that SyncSort receives a valid return code before the GOBACK statement is executed.

EXIT-STATUS Codes (Fixed and Variable-Length Records)

- 00** *First record.* SyncSort uses this Code to indicate the first call to the COBOL exit and that the first record from SORTIN is in the RECORD-UP area.
- 04** *Most records.* This is used for all calls except the first one when there are records in the RECORD-UP area. After Code 00 has been issued, Code 04 is passed to the exit until there is no record for the sort to pass to the RECORD-UP area.
- 08** *All records passed.* This indicates that the last SORTIN record has already been processed by the exit. Do not attempt to reference the record again. No more records will be passed to the exit routine. Note that if the SORTIN data set is empty, 08 will be passed every time *including* the first time.

RETURN-CODE Codes (Fixed and Variable-Length Records)

- 0** *Accept this record.* This instructs SyncSort to accept the (unaltered) record in the RECORD-UP area.
- 4** *Delete this record.* SyncSort will delete the current record in the RECORD-UP area.
- 8** *Do not return to this exit.* This instructs SyncSort to close the exit for the remainder of the sort application. This return code might be used at SORTIN end-of-file (Exit Status Code 08) to indicate that extra records will not be added at this point. If SORTIN is present, the current input record and all subsequent records will be processed by SyncSort.
- 12** *Insert a record.* This instructs SyncSort to add the record in the WORK area to the input data set just ahead of the current record in the RECORD-UP area. When SyncSort returns control to the E15, the same record will be in the RECORD-UP area. The exit routine can then add another record from the WORK area or process the current record in RECORD-UP.
- 16** *Terminate SyncSort.* SyncSort will end its program and return to the calling program or the Supervisor. SyncSort will issue a completion code of 16 to indicate that the sort was unsuccessful.
- 20** *Replace current record.* SyncSort will replace the current record in the RECORD-UP area with the record in the WORK area. Be sure that the record in the WORK area is valid before passing it to SyncSort.

To Change a Record

In order to change the record in the RECORD-UP area, first move it to the WORK area. All changes are made to the WORK area copy, which replaces the record in RECORD-UP when 20 is moved to RETURN-CODE.

Sample COBOL E15, Fixed-Length Records

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. E15FL13C.  
  
ENVIRONMENT DIVISION.  
  
CONFIGURATION SECTION.  
  
SOURCE-COMPUTER. IBM-370.  
OBJECT-COMPUTER. IBM-370.  
  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
I-O-CONTROL.  
DATA DIVISION.  
FILE SECTION.  
  
WORKING-STORAGE SECTION.  
01  EVEN-FLAG                                PIC  9(2) VALUE ZERO.  
  
01  USER-RETURN-CODE                        PIC  9(8) COMPUTATIONAL.  
    88  ACCEPT-REC                          VALUE  0.  
    88  DELETE-REC                          VALUE  4.  
    88  END-EXIT                            VALUE  8.  
    88  INSERT-REC                          VALUE 12.  
    88  END-SORT                            VALUE 16.  
    88  REPL-REC                            VALUE 20.  
  
01  CHANGE-REC.  
    05  C-REC                                PIC  X(6) VALUE 'CHANGE'.  
    05  C-INCR                              PIC  9(4) VALUE ZERO.  
    05  C-BLANK                             PIC  X(90) VALUE ZERO.  
  
01  INSRT-REC.  
    05  I-REC                                PIC  X(6) VALUE 'INSERT'.  
    05  I-INCR                              PIC  9(4) VALUE ZERO.  
    05  I-BLANK                             PIC  X(90) VALUE SPACES.  
  
01  TOTAL.  
    05  BLANKS                              PIC  X(10) VALUE ' E15'.  
    05  TITL                                PIC  X(25) VALUE 'TOTAL RECORDS OUT'.  
    05  COUNTER                             PIC  9(8) VALUE 0.
```

Sample COBOL E15, Fixed-Length Records (Continued)

```
LINKAGE SECTION.
01  EXIT-STATUS          PIC  9(8) COMPUTATIONAL.
    88 FIRST-TIME      VALUE 00.
    88 LAST-TIME       VALUE 08.

01  RECORD-UP.
    07 FILLER          PIC  9(6) .
    07 R-SEQ1          PIC  9(2) .
    07 FILLER          PIC  X(92) .
01  WORK                PIC  X(100) .

PROCEDURE DIVISION USING EXIT-STATUS, RECORD-UP, WORK.

    IF COUNTER GREATER THAN 100
        MOVE 0 TO RETURN-CODE
        GO TO RETURN-TO-SORT.
    IF LAST-TIME GO TO RETURN-TO-SORT.
    IF R-SEQ1 EQUAL 0
        MOVE 4 TO RETURN-CODE
        GO TO RETURN-TO-SORT.
    IF R-SEQ1 EQUAL 5
        MOVE 12 TO RETURN-CODE
        MOVE INSRT-REC TO WORK
        GO TO RETURN-TO-SORT.

    IF R-SEQ1 EQUAL 6
        MOVE 20 TO RETURN-CODE
        MOVE CHANGE-REC TO WORK
        GO TO RETURN-TO-SORT.
    MOVE 0 TO RETURN-CODE.

RETURN-TO-SORT.
    ADD 1 TO COUNTER.

    IF LAST-TIME MOVE 8 TO RETURN-CODE
        DISPLAY TOTAL.
    GOBACK.
```

Sample COBOL E15, Variable-Length Records

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. E15VL19C.  
  
ENVIRONMENT DIVISION.  
  
CONFIGURATION SECTION.  
  
SOURCE-COMPUTER. IBM-390.  
OBJECT-COMPUTER. IBM-390.  
  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
I-O-CONTROL.  
  
DATA DIVISION.  
FILE SECTION.  
  
WORKING-STORAGE SECTION.  
01  EVEN-FLAG                               PIC 9(2) VALUE ZERO.  
  
01  USER-RETURN-CODE                       PIC 9(8) COMPUTATIONAL.  
    88  ACCEPT-REC                          VALUE 0.  
    88  DELETE-REC                          VALUE 4.  
    88  END-EXIT                             VALUE 8.  
    88  INSERT-REC                           VALUE 12.  
    88  END-SORT                             VALUE 16.  
    88  REPL-REC                             VALUE 20.  
  
01  CHANGE-REC.  
    05  C-REC                               PIC X(6) VALUE 'CHANGE'.  
    05  C-INCR                              PIC 9(4) VALUE ZERO.  
    05  C-BLANK                              PIC X(90) VALUE SPACES.  
  
01  INSRT-REC.  
    05  I-REC                               PIC X(6) VALUE 'INSERT'.  
    05  I-INCR                              PIC 9(4) VALUE ZERO.  
    05  I-BLANK                              PIC X(90) VALUE SPACES.  
  
01  TOTAL.  
    05  BLANKS PIC X(10) VALUE ' E15'.
```

Sample COBOL E15, Variable-Length Records (Continued)

```
05 TITL PIC X(25) VALUE 'TOTAL RECORDS OUT'.  
05 COUNTER PIC 9(8) VALUE 0.
```

LINKAGE SECTION.

```
01 EXIT-STATUS PIC 9(8) COMPUTATIONAL.  
88 FIRST-TIME VALUE 00.  
88 MOST-TIME VALUE 04.  
88 LAST-TIME VALUE 08.  
  
01 RECORD-UP.  
05 RU OCCURS 1 TO 100 TIMES  
DEPENDING ON LEN-RU PIC X.  
  
01 WORK.  
05 WK OCCURS 1 TO 100 TIMES  
DEPENDING ON LEN-WK PIC X.  
  
01 IN-BUF PIC X(100).  
01 DUMMY PIC 9(8) COMPUTATIONAL.  
01 LEN-RU PIC 9(8) COMP.  
01 LEN-WK PIC 9(8) COMP.
```

```
PROCEDURE DIVISION USING EXIT-STATUS, RECORD-UP, WORK,  
IN-BUF, DUMMY, LEN-RU, LEN-WK.
```

```
IF NOT FIRST-TIME  
ADD 1 TO COUNTER  
MOVE 0 TO RETURN-CODE.
```

```
IF COUNTER LESS THAN 50  
MOVE 54 TO LEN-WK  
ADD 1 TO I-INCR  
MOVE INSRT-REC TO WORK  
MOVE 12 TO RETURN-CODE  
GO TO RETURN-TO-SORT.
```

```
IF COUNTER LESS THAN 75  
MOVE 44 TO LEN-WK  
ADD 1 TO C-INCR  
MOVE CHANGE-REC TO WORK
```

Sample COBOL E15, Variable-Length Records (Continued)

```
MOVE 20 TO RETURN-CODE
GO TO RETURN-TO-SORT.

IF COUNTER LESS THAN 100
MOVE 80 TO LEN-WK
ADD 1 TO I-INCR
MOVE 12 TO RETURN-CODE
MOVE INSRT-REC TO WORK.
GO TO RETURN-TO-SORT.

IF COUNTER LESS THAN 200
MOVE 4 TO RETURN-CODE
GO TO RETURN-TO-SORT.

RETURN-TO-SORT.
IF LAST-TIME MOVE 8 TO RETURN-CODE
DISPLAY TOTAL.
GOBACK.
```

Coding a C E15 Exit Routine

A C E15 exit program is indicated by the MODS control statement. An E15 exit cannot be coded in C when running Tape Sort.

Like any other E15 exit routine, the C E15 exit routine is called each time a record is brought into Phase 1 of a sort or Phase 3 of a copy. SyncSort and the C exit communicate through arguments defined in the function header. For example, records are passed to the C routine by the address presented in the second argument in the function parameter list. No storage is reserved in the exit program because the records exist elsewhere.

The C E15 exit routine can be written using either the C370 V2R1 compiler with the V2R2 C370 Library, the SAA AD/Cycle C370 V1R2 Compiler and Library or using the C/C++ for MVS/ESA V3R1.1 or higher Compiler and Library. When using the LE/370 run time library modules, it may be necessary to account for this additional storage by adjusting the b value of the Exit-Name parameter on the MODS statement.

Exit Communication

The parameter list structure required for passing fixed-length and variable-length records between the sort and the exit is detailed in the following section. The parameter names used in the examples are arbitrary but each definition is required. Complete sample programs showing the use of the argument lists are presented following the discussion of the exit interface.

Fixed-Length Records - Function Definition

```
int E15exit ( int* exit_status,
             struct_ru* record_up,
             struct_ins_rep* work,
             int* dummy1, int* dummy2, int* dummy3,
             int* dummy4, int* dummy5,
             int* comm_len,
             struct_ca* communication_area)
```

The following describes the parameters used in the preceding definition.

exit_status	<p>This parameter points to a variable containing one of the following exit status codes:</p> <ul style="list-style-type: none">00 <i>First record.</i> SyncSort uses this code to indicate the first call to the C exit and that the first record from SORTIN is in the record_up area. If the SORTIN is empty or does not exist, a 08 status will be passed the first time.04 <i>Most records.</i> This is used for all calls except the first one when there are records in the record_up area. After Code 00 has been issued, Code 04 is passed to the exit until there is no record for the sort to pass to the record_up area.08 <i>All records passed.</i> This indicates that the last SORTIN record has already been processed by the exit. Do not attempt to reference the record again. No more records will be passed to the exit routine. Note that if the SORTIN data set is empty or does not exist, 08 will be passed every time including the first time.
record_up	<p>The record_up parameter contains a pointer to the record being passed to the E15 from the SORTIN. The struct_ru data type represents a structure that describes the fields within the SORTIN record.</p>
work	<p>The work parameter contains a pointer to a work area that is to be used to hold an inserted or replaced record returned from the E15. The struct_ins_rep data type represents a structure that describes the fields within the inserted or replaced record.</p>
dummy1 - dummy5	<p>These parameters define unused place holders. They are used with variable-length E15 and E35 communication. Their definition here allows a common parameter list for fixed-length and variable-length C E15 and E35 exits.</p>

comm_len	This parameter points to a variable that defines the communication area length.
communication_area	The communication_area parameter contains a pointer to the communication area. The struct_ca data type represents a structure that describes the fields in the communication area.

Variable-Length Records - Function Definition

```
int E15exit ( int* exit_status,
             void* record_up,
             void* work,
             int* dummy1, int* dummy2,
             int* len_ru,
             int* len_wk,
             int* dummy3,
             int* comm_len,
             struct_ca* communication_area)
```

The following describes the parameters used in the preceding definition.

exit_status	This parameter points to a variable containing exit status codes. See the exit_status definition for a fixed-length C E15 exit for the code definitions.
record_up	The record_up parameter contains a "universal" pointer to the record being passed to the E15 from the SORTIN. The void* pointer can be cast to point an appropriate structure to describe the record passed to the exit. This allows different record structures, as is common with variable-length records, to share a single pointer definition.
work	The work parameter contains a "universal" pointer to a work area that is to be used to hold an inserted or replaced record returned from the E15. The void* pointer can be cast to point an appropriate structure to describe the work record.
dummy1 - dummy3	These parameters define unused place holders. They are used with C E35 communication. Their definition here allows a common parameter list for C E15 and E35 exits.
len_ru	This parameter points to a variable that defines the length of the SORTIN record passed to the E15. This is the length of the record referred to in the record_up parameter.
len_wk	This parameter points to a variable that defines the length of the record to be inserted or used as a replacement for the

record_up record. This is the length of the record referred to in the work parameter. This field must be set by the exit when an insert or replace operation is performed.

comm_len	This parameter points to a variable that defines the communication area length.
communication_area	The communication_area parameter contains a pointer to the communication area. The struct_ca data type represents a structure that describes the fields in the communication area.

RETURN-CODE Codes (Fixed and Variable-Length Records)

The RETURN statement is used to return control to SyncSort. It must indicate one of the following return values to indicate the action to be taken by SyncSort.

0	<i>Accept this record.</i> This instructs SyncSort to accept the (unaltered) record in the record_up area.
4	<i>Delete this record.</i> SyncSort will delete the current record in the record_up area.
8	<i>Do not return to this exit.</i> This instructs SyncSort to close the exit for the remainder of the sort application. This return code might be used at SORTIN end-of-file (exit_status code 08) to indicate that extra records will not be added at this point. If SORTIN is present, the current input record and all subsequent records will be processed by SyncSort.
12	<i>Insert a record.</i> This instructs SyncSort to add the record in the work area to the input data set just ahead of the current record in the record_up area. When SyncSort returns control to the E15, the same record will be in the record_up area. The exit routine can then add another record from the work area or process the current record in record_up. When inserting a variable-length record, insure that its length is indicated in the len_wk parameter.
16	<i>Terminate SyncSort.</i> SyncSort will end its program and return to the calling program or the Supervisor. SyncSort will issue a completion code of 16 to indicate that the sort was unsuccessful.
20	<i>Replace current record.</i> SyncSort will replace the current record in the record_up area with the record in the work area. Be sure that the record in the work area is valid before passing it to SyncSort. When replacing a variable-length record, insure that its length is indicated in the len_wk parameter.

How to Change a Record

To change the record in the record_up area, first move it to the work area. All changes are made to the work area copy, which replaces the record in record_up when the return value from the exit is 20.

Sample C E15, Fixed-Length Records

```
#define FIRST_TIME 0
#define MOST_TIME 4
#define LAST_TIME 8
#define ACCEPT_REC 0
#define DELETE_REC 4
#define END_EXIT 8
#define INSERT_REC 12
#define END_SORT 16
#define REPL_REC 20
typedef _Packed struct record {
    char name[6];
    char code[4];
    int serial_no;
} t_ru;
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int SMPE15FB(int* exit_status,t_ru* record_up,t_ru* work,int* dummy1,
    int* dummy2,int* dummy3,int* dummy4,int* dummy5,int* comm_len,
    void* communication_area)
{
    static counter=0;
    int icode,return_code;
    char * text1="CHANGE";
    char * text2="INSERT";
    if (counter > 10) {return_code=ACCEPT_REC;
        goto return_to_sort;}
    if (*exit_status == LAST_TIME) {return_code=END_EXIT;
        goto return_to_sort;}
```

Sample C E15, Fixed-Length Records (Continued)

```
sscanf(record_up->code, "%4d", &icode);
if (icode==0) { return_code=DELETE_REC;
               goto return_to_sort;}
if (icode==5) {
               strncpy(work->name, text2, 6);
               sprintf(work->code, "%4d", icode+counter+8);
               work->serial_no=300;
               return_code=INSERT_REC;
               goto return_to_sort;}
if (icode==6) {
               strncpy(work->name, text1, 6);
               sprintf(work->code, "%4d", icode+1);
               work->serial_no=record_up->serial_no+200;
               return_code=REPL_REC;
               goto return_to_sort;}
return_code=ACCEPT_REC;
return_to_sort:
    counter++;
    if (*exit_status==LAST_TIME)
        { return_code=END_EXIT;
          printf("E15 total number of records handled:%d\n", counter);
        }
    return(return_code);
}
```

Sample C E15, Variable-Length Records

```
#define FIRST_TIME 0
#define MOST_TIME 4
#define LAST_TIME 8
#define ACCEPT_REC 0
#define DELETE_REC 4
#define END_EXIT 8
#define INSERT_REC 12
#define END_SORT 16
#define REPL_REC 20
#define MAX_RLEN 104
typedef _Packed struct record1 {
    char rec[6];
    int incr;
    char address[MAX_RLEN-14];
} t_ru1;
typedef _Packed struct record2 {
    char title[10];
    int number;
} t_ru2;
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
int SMPE15VB(int* exit_status, void* record_up, void* work, int* dummy1,
    int* dummy2, int* len_ru, int* len_wk, int* dummy3, int* comm_len,
    void* communication_area)
{
    static counter=0, i_incr=0, i_number=0;
    int return_code;
    char *text1="CHANGE E15";
    char *text2="INSERT E15";
    t_ru1 * p_record1, *pwork1;
    t_ru2 * p_record2, *pwork2;
    p_record1 = (t_ru1 *)record_up;
    pwork1 = (t_ru1 *)work;
    p_record2 = (t_ru2 *)record_up;
    pwork2 = (t_ru2 *)work;
    if (*exit_status != FIRST_TIME) {counter++;
        return_code=ACCEPT_REC;}
}
```

Sample C E15, Variable-Length Records (Continued)

```
if (counter<50) {
    if (*len_ru == 14)
    {
        *len_wk = 14;
        i_number++;
        pwork2->number=i_number;
        strncpy(pwork2->title,text2,10);
    }
    else {
        *len_wk = 54;
        i_incr++;
        pwork1->incr=i_incr;
        strncpy(pwork1->rec,text2,6);
    }
    return_code=INSERT_REC;
    goto return_to_sort;}
if (counter<75) {
    if (*len_ru == 14)
    {
        *len_wk = 14;
        pwork2->number=p_record2->number+1;
        strncpy(pwork2->title,text1,10);
    }
    else {
        *len_wk = 54;
        pwork1->incr=p_record1->incr+1;
        strncpy(pwork1->rec,text1,6);
    }
    return_code=REPL_REC;
    goto return_to_sort;}
if (counter<100) {
    if (*len_ru == 14)
    {
        *len_wk = 14;
        i_number++;
        pwork2->number=i_number;
        strncpy(pwork2->title,text2,10);
    }
}
```

Sample C E15, Variable-Length Records (Continued)

```
        else {
            *len_wk = 80;
            i_incr++;
            pwork1->incr=i_incr;
            strncpy(pwork1->rec,text2,6);
        }
        return_code=INSERT_REC;
        goto return_to_sort;}
if (counter<200) {
    return_code=DELETE_REC;
    goto return_to_sort;}
return_to_sort:
    if (*exit_status==LAST_TIME)
        { return_code=END_EXIT;
          printf("E15 total number of records handled:%d\n",counter);
        }
    return(return_code);
}
```

Exit E25 - Deleting, Changing, Summarizing Records

SyncSort gives control to exit E25 each time it is about to place a record in a Phase 2 output sequence, except for the first record of that sequence. Because all or part of the input data set may skip this phase, it may be necessary to include an E35 to do the job of the E25 during Phase 3. If it is possible to use the SUM control statement in place of the exit, this is recommended.

These constraints apply to the coding of an E25 exit routine:

- The exit may not add records.
- The exit may not change sort control fields.
- The exit may not destroy the contents of the parameter list.

SyncSort will place the address of a 2-word parameter list in Register 1 each time it passes control to the E25 routine. The first word, which is on a fullword boundary, will contain the address of the record about to leave Phase 2. The second word will contain the address of the record that has already passed into the output area. Note that the first byte of each word contains zeros.

PARAMETER LIST	
Word 1:	Address of record leaving Phase 2
Word 2:	Address of record already in output area

Figure 186. Parameter List for E25

In order to change the record leaving Phase 2, the E25 exit program must first move it to a work area. (The record in the output area may be changed, but must be left where it is.) To sum two records, place the sum in the output area record and delete the record leaving Phase 2.

After the record pair has been processed by the E25, a return code is placed into Register 15 and control returns to SyncSort.

Return Codes

- | | |
|-----------|---|
| 0 | <i>Accept this record.</i> To instruct SyncSort to accept the record leaving Phase 2, whether changed or unchanged, place return code 0 into Register 15. The (work area) address of the record to be accepted must be placed into Register 1. |
| 4 | <i>Delete this record.</i> This tells SyncSort to delete the record about to leave Phase 2. It is not necessary to place the address of this record in Register 1. The next time SyncSort returns control to the exit program, the address of a new record will be in word 1 of the parameter list but word 2 will be unchanged. (This permits further summarization, for example.) |
| 16 | <i>Terminate SyncSort.</i> SyncSort will end its program and return to the calling program or the supervisor. SyncSort will give the user a completion code of 16 to indicate that the sort was unsuccessful. |

Exit E35 - Adding, Deleting and Changing Records

When an output data set is available, the user may elect to incorporate this exit to add, delete or change records at the end of Phase 3. In the absence of an output data set, this exit has full responsibility for output processing and, under normal conditions, will delete every record passed by the sort.

E35 record processing has these constraints:

- If a record is to be changed, it should first be moved to a work area.
- The exit program may not destroy the contents of the parameter list.

- A user exit may not take checkpoints.

Coding the E35 Exit Routine for an Invoked Sort/Merge/Copy

When SyncSort is initiated from an ATTACH, LINK or XCTL macro, there are two ways to include an E35 exit routine: (1) code the E35 exit routine in line with the invoking program and specify the address of its entry point in the appropriate entry of the calling program's parameter list; or (2) define the separately compiled routine in the MODS control statement. When the exit routine is coded in line with the invoking program, it must handle all output processing; SyncSort will ignore a SORTOUT DD statement and an OUTFIL control statement, if present.

The E35 Parameter List

This exit routine is given control each time SyncSort is about to place a record in the output area after the final merge. In passing control to the E35 exit routine, SyncSort places the address of a parameter list in Register 1. The parameter list starts at a fullword boundary and is 3 words long; the first byte of each word contains binary zeros. The first word contains the address of the record about to leave Phase 3; after the last record has been passed, this word will contain zeros. The second word contains the address of the record already in the output area; when the first record is passed, this word will contain zeros.

The third word contains the user address constant. It contains either the last value set in it by an E15 exit routine or, if not modified by an E15 exit routine, the initial value from the user exit address constant provided in the invoking parameter list. If the value was obtained from the 24-bit invoking parameter list, it is limited to 24 bits with the high-order byte set to X'00'.

If the user exit address constant was not provided or if SyncSort was JCL-invoked, it will contain binary zeros. This word may be changed by the E35 exit routine whenever it is entered, and it will remain the same on all subsequent entries to the E35 exit routine.

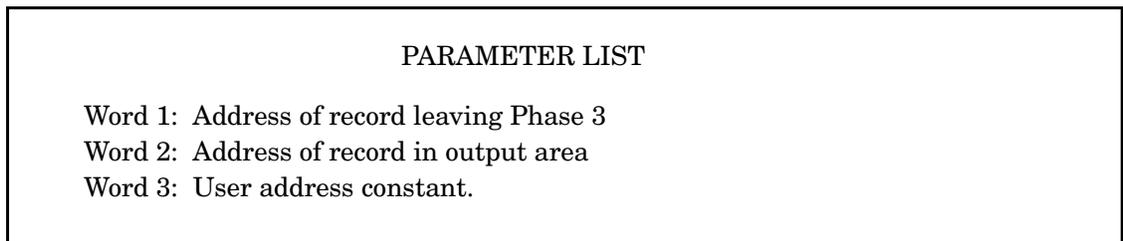


Figure 187. Parameter List for E35

Return Codes

- 0** *Accept this record.* This instructs SyncSort to accept the record now leaving Phase 3. Place the (work area) address of this record in Register 1. This return code is used when selectively editing records for output; it passes the

(possibly altered) record back to the sort. The RECORD statement is required if this exit routine changes the maximum record length.

- 4 *Delete this record.* SyncSort will delete the record leaving Phase 3. There is no need to load the address of this record into Register 1. When SyncSort returns control to the E35, the first word of the parameter list (the address of the record leaving Phase 3) will refer to a new record, but the second word (the address of the output area record) will be unchanged.
- 8 *Disconnect E35.* This instructs SyncSort to process any remaining records without showing them to the E35 exit. Register 1 is ignored for processing this return code. When this return code is used at end-of file (signalled by a zero address in the first word of the parameter list), it indicates that E35 is also finished and will not add additional records. When used before end-of-file, it indicates that SyncSort should process the "current" record passed to the E35, and any subsequent records, as if there were no E35 present. Note that when SyncSort is not creating any output files (SORTOUT or SORTOFxx) and E35 is the only "output", SyncSort terminates immediately, since any subsequent records will never be seen. Note that if an XSUM data set was being created, it will only contain records generated prior to the return code of 8.
- 12 *Insert a record.* This tells SyncSort to add a record just before the record is about to leave Phase 3. Load the address of the inserted record into Register 1. When SyncSort returns control to the E35 exit routine, the first word of the parameter list (the address of the record leaving Phase 3) will be unchanged, but the second word (the address of the output area record) will refer to the inserted record. The exit routine can then add another record or process the current one.
- 16 *Terminate SyncSort.* This tells SyncSort to end its program and return to the calling program or the supervisor. SyncSort uses a completion code of 16 to indicate that the sort was unsuccessful.

Coding a COBOL E35 Exit Routine

A COBOL E35 exit program can be indicated through the EXEC statement PARM option (PARM='E35=COB'), the MODS control statement or the \$ORTPARM DD statement. An E35 exit cannot be coded in COBOL when running Tape Sort.

Like any other E35 exit routine, the COBOL E35 is called each time a record is brought out of Phase 3. Communication between SyncSort and the COBOL exit takes place in the LINKAGE SECTION of the COBOL program. For example, records are passed to the COBOL routine in the second definition (RECORD-UP) area of the LINKAGE SECTION. No storage is reserved in the exit program because the records exist elsewhere.

If the COBOL exit routine uses any verb (EXHIBIT, DISPLAY, TRACE) which results in output to the SYSOUT DD statement, there is a potential conflict with SyncSort's use of this DD statement. It is therefore recommended that the user separate the output by using either SyncSort's MSGDD PARM option or the COBOL compiler's SYSx parm.

If the COBOL E35 exit routine is written in OS/VS COBOL and uses OS/VS COBOL libraries or no libraries, specify the COBEXIT=COB1 PARM option. If the COBOL E35 exit routine is written in OS/VS COBOL or in VS COBOL II or COBOL/370 and uses VS COBOL II or COBOL/370 libraries, specify the COBEXIT=COB2 PARM option. VS COBOL II or COBOL/370 run-time library modules typically require more main storage than OS/VS COBOL library modules. The amount of additional storage depends on VS COBOL II or COBOL/370 installation options. When VS COBOL II or COBOL/370 run-time library modules are used, it may be necessary to account for this additional storage by adjusting the b value of the Exit-Name parameter on the MODS statement.

The LINKAGE SECTION

The LINKAGE SECTION examples that follow show the parameters required for passing fixed-length and variable-length records to the sort. The data-names and conditional names used in the examples are arbitrary but each definition is required. The complete programs from which the examples are taken follow the discussion of the exit.

Example 1: Fixed-Length Records

```
LINKAGE SECTION.  
01  EXIT-STATUS          PIC  9(8)          COMPUTATIONAL.  
    88  FIRST-TIME      VALUE 00.  
    88  MOST-TIME       VALUE 04.  
    88  LAST-TIME       VALUE 08.  
  
01  RECORD-UP.  
    05  RU              PIC  X(100).  
  
01  WORK.  
    05  WK              PIC  X(100).  
  
01  IN-BUF.  
    05  IB              PIC  X(100).  
  
01  DUMMY1              PIC  X(4).  
01  DUMMY2              PIC  X.  
01  DUMMY3              PIC  X.  
01  DUMMY4              PIC  X.  
  
01  COMM-LEN            PIC  9(4)          COMPUTATIONAL.  
  
01  COMMUNICATION-AREA.  
  
    05  COMM-AREA OCCURS 1 TO 256 TIMES  
        DEPENDING ON COMM-LEN PIC X.
```

The PICTURE and VALUE clauses for (1) the record passed from SyncSort, (2) the record WORK area, and (3) the record in the output buffer are application-specific.

- For the first definition (EXIT-STATUS) specify PIC 9(8) COMPUTATIONAL. When using 88 levels to define exit status codes, specify values 00, 04, and 08.
- For the second definition (RECORD-UP) define the record leaving Phase 3.
- For the third definition (WORK) define the record that SyncSort is to put in the output data set. This is the "work" area.
- For the fourth definition (IN-BUF) define the record in the output data set.
- For the fifth definition define a dummy area with PIC X(4).
- For the sixth through the eighth definition define dummy areas.

- For the ninth definition (COMM-LEN) specify PIC 9(4) COMPUTATIONAL. This area defines the communication area length.
- For the tenth definition (COMMUNICATION-AREA) code an OCCURS clause DEPENDING ON data-name PIC X.

Example 2: Variable-Length Records

```

LINKAGE SECTION.
01  EXIT-STATUS                                PIC  9(8) COMPUTATIONAL.
    88  FIRST-TIME                             VALUE 00.
    88  MOST-TIME                              VALUE 04.
    88  LAST-TIME                              VALUE 08.

01  RECORD-UP.
    05  RU                                     OCCURS 1 TO 100 TIMES
                                           DEPENDING ON LEN-RU          PIC X.

01  WORK.
    05  WK                                     OCCURS 1 TO 100 TIMES
                                           DEPENDING ON LEN-WK          PIC X.

01  IN-BUF.
    05  IB                                     OCCURS 1 TO 100 TIMES
                                           DEPENDING ON LEN-IB          PIC X.

01  DUMMY                                     PIC X(4)
01  LEN-RU                                    PIC  9(8) COMPUTATIONAL.
01  LEN-WK                                    PIC  9(8) COMPUTATIONAL.
01  LEN-IB                                    PIC  9(8) COMPUTATIONAL.
01  COMM-LEN                                  PIC  9(4) COMPUTATIONAL.

01  COMMUNICATION-AREA.
    05  COMM-AREA OCCURS 1 TO 256 TIMES
           DEPENDING ON COMM-LEN PIC X.

```

- For the first definition (EXIT-STATUS) specify PIC 9(8) COMPUTATIONAL. When using 88 levels to define exit status codes, specify values 00, 04, and 08.
- For the second definition (RECORD-UP) code an OCCURS clause with the DEPENDING ON *data-name* option specifying (1) the minimum and maximum number of bytes of your variable-length records leaving Phase 3 (do not include 4 bytes for the RDW) and (2) DEPENDING ON *data-name* PIC X. *Data-name* is defined in the sixth definition in the LINKAGE SECTION.

- For the third definition (WORK) code an OCCURS clause with the DEPENDING ON *data-name* option specifying (1) the minimum and maximum number of bytes for variable-length records you will pass to SyncSort (do *not* include 4 bytes for the RDW) and (2) DEPENDING ON *data-name* PIC X. *Data-name* is defined as the seventh definition in the LINKAGE SECTION. This area is used for the "work" area.
- For the fourth definition (IN-BUF) define records in the output area. Code an OCCURS clause with the DEPENDING ON *data-name* option specifying (1) the minimum and maximum number of bytes for variable-length records in the output data set (do *not* include 4-bytes for the RDW) and (2) DEPENDING ON *data-name* PIC X. *Data-name* is defined as the eighth definition in the LINKAGE SECTION.
- For the fifth definition define a dummy area with PIC X(4).
- For the sixth definition (LEN-RU) specify PIC 9(8) COMPUTATIONAL. SyncSort will pass the length of the record leaving Phase 3 in this area.
- For the seventh definition (LEN-WK) specify PIC 9(8) COMPUTATIONAL. The E35 routine passes SyncSort the length of the record in the work area in this section.
- For the eighth definition (LEN-IB) specify PIC 9(8) COMPUTATIONAL. SyncSort passes the length of the record in the output area in this section.
- For the ninth definition (COMM-LEN) specify PIC 9(4) COMPUTATIONAL. This area defines the communication area length.
- For the tenth definition (COMMUNICATION-AREA) code an OCCURS clause DEPENDING ON *data-name* PIC X.

The IDENTIFICATION, ENVIRONMENT, and DATA Divisions

As always, the COBOL program must contain the entries required by the compiler for these program divisions. Code the optional entries in these divisions according to the requirements of the application.

The WORKING-STORAGE SECTION

If the exit routine inserts records into the final merge and replaces records passed from SyncSort, the insertion record and the replacement record may be defined in this section. These records will be moved to the WORK area described in the LINKAGE SECTION, so be sure that the PICTURE clause or the OCCURS clause in the WORK area is correct for these records.

This section may also define the return codes as 77-level data items. Alternatively, these codes can be specified as literals in the MOVE instruction. (MOVE *literal* to RETURN-CODE.) Note that RETURN-CODE is the name of a predefined storage area in COBOL

used to pass return codes to the sort; RETURN-CODE should not be defined in the exit routine.

The PROCEDURE DIVISION

Specify the USING option on the PROCEDURE DIVISION header. Each identifier specified after USING must be the same as those described in the 01-level of the LINKAGE SECTION. Taking for example the identifiers defined in the fixed-length record LINKAGE SECTION shown here, they would appear as: PROCEDURE DIVISION USING EXIT-STATUS, RECORD-UP, WORK, IN-BUF, DUMMY1, DUMMY2, DUMMY3, DUMMY4, COM-LEN, COMMUNICATION-AREA.

The GOBACK statement is used to return control to SyncSort. Do *not* use the EXIT statement as it will cause unpredictable results. Be sure that SyncSort receives a valid return code before the GOBACK statement is executed.

EXIT-STATUS Codes (Fixed and Variable-Length Records)

- 00** *First Record.* SyncSort uses this Code to indicate the first call to the COBOL exit and that the first record to leave Phase 3 is in the RECORD-UP area.
- 04** *Most records.* This is used for all calls except the first one when there are records in the RECORD-UP area. After Code 00 has been issued, Code 04 is passed to the exit until there is no record for the sort to pass to the RECORD-UP area.
- 08** *All records passed.* This indicates that the last record has already been processed by the exit. Do not attempt to reference the record again. No more records will be passed to the exit routine. Note that if SyncSort is not passing any records to Phase 3, 08 will be passed every time *including* the first time.

RETURN-CODE Codes (Fixed and Variable-Length Records)

- 0** *Accept this record.* This instructs SyncSort to accept the (unaltered) record in the RECORD-UP area.
- 4** *Delete this record.* SyncSort will delete the current record in the RECORD-UP area.
- 8** *Disconnect E35.* This instructs SyncSort to process any remaining records without showing them to the E35 exit. Register 1 is ignored for processing this return code.

When this return code is used at end-of-file (signalled by EXIT-STATUS LAST-TIME), it indicates that the E35 is also finished and will not add

additional records. When used before end-of-file, it indicates that SyncSort should process the "current" record passed to the E35, and any subsequent records, as if there were no E35 present. Note that when SyncSort is not creating any output files (SORTOUT or SORTOFxx) and E35 is the only "output," SyncSort terminates immediately, since any subsequent records will never be seen. Also note that if an XSUM data set was being created, it will only contain records generated prior to the return code of 8.

- 12** *Insert a record.* This instructs SyncSort to add the record in the WORK area to the input data set just ahead of the current record in the RECORD-UP area. When SyncSort returns control to the E35, the same record will be in the RECORD-UP area. The exit routine can then add another record from the WORK area or process the current record in RECORD-UP.
- 16** *Terminate SyncSort.* SyncSort will terminate and return to the calling program or the Supervisor. SyncSort will issue a completion code of 16 to indicate that the sort was unsuccessful.
- 20** *Replace current record.* SyncSort will replace the current record in the RECORD-UP area with the record in the WORK area. Be sure that the record in the WORK area is valid before passing it to SyncSort.

To Change a Record

In order to change the record in the RECORD-UP area, first move it to the WORK area. Make the changes there and then pass return code 20 in RETURN-CODE. The altered record in the WORK area will replace the record in RECORD-UP.

Sample COBOL E35, Fixed-Length Records

```
IDENTIFICATION DIVISION.

PROGRAM-ID. E35FL101.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-390.
OBJECT-COMPUTER. IBM-390.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.

DATA DIVISION.
FILE SECTION.

WORKING-STORAGE SECTION.
01  EVEN-FLAG                               PIC 9(2)    VALUE ZERO.

01  USER-RETURN-CODE                       PIC 9(8)    COMPUTATIONAL.
    88  ACCEPT-REC                          VALUE      0.
    88  DELETE-REC                          VALUE      4.
    88  END-EXIT                             VALUE      8.
    88  INSERT-REC                           VALUE     12.
    88  END-SORT                              VALUE     16.
    88  REPL-REC                              VALUE     20.

01  CHANGE-REC.
    05  C-REC                                PIC X(6)    VALUE 'CHANGE'.
    05  C-INCR                               PIC 9(4)    VALUE ZERO.
    05  C-BLANK                              PIC X(90)   VALUE SPACES.

01  INSRT-REC.
    05  I-REC                                PIC X(6)    VALUE 'INSERT'.
    05  I-INCR                               PIC 9(4)    VALUE ZERO.
    05  I-BLANK                              PIC X(90)   VALUE SPACES.

01  TOTAL.
    05  BLANKS                              PIC X(10)   VALUE 'E35'.
    05  TITL                                PIC X(25)
        VALUE 'TOTAL RECORDS HANDLED'.
    05  COUNTER                              PIC 9(8)    VALUE 0
```

Sample COBOL E35, Fixed-Length Records (Continued)

```
LINKAGE SECTION.  
01  EXIT-STATUS                      PIC 9(8)    COMPUTATIONAL.  
    88  FIRST-TIME                    VALUE 00.  
    88  MOST-TIME                     VALUE 04.  
    88  LAST-TIME                     VALUE 08.  
  
01  RECORD-UP.  
    05  RU                            PIC X(100).  
01  WORK.  
    05  WK                            PIC X(100).  
01  IN-BUF.  
    05  IB                            PIC X(100).  
01  DUMMY1                           PIC X(4).
```

```
        PROCEDURE DIVISION USING EXIT-STATUS, RECORD-UP, WORK,  
                                IN-BUF, DUMMY.
```

```
IF NOT FIRST-TIME  
    ADD 1 TO COUNTER  
    MOVE 0 TO RETURN-CODE.
```

```
IF COUNTER LESS THAN 50  
    ADD 1 TO I-INCR  
    MOVE INSRT-REC TO WORK  
    MOVE 12 TO RETURN-CODE  
    GO TO RETURN-TO-SORT.
```

```
IF COUNTER LESS THAN 75  
    ADD 1 TO C-INCR  
    MOVE CHANGE-REC TO WORK  
    MOVE 20 TO RETURN-CODE  
    GO TO RETURN-TO-SORT.
```

Sample COBOL E35, Fixed-Length Records (Continued)

```
IF COUNTER LESS THAN 100
  ADD 1 TO I-INCR
  MOVE INSRT-REC TO WORK
  MOVE 12 TO RETURN-CODE
  GO TO RETURN-TO-SORT.

IF COUNTER LESS THAN 200
  MOVE 4 TO RETURN-CODE
  GO TO RETURN-TO-SORT.

RETURN-TO-SORT.
  IF LAST-TIME MOVE 8 TO RETURN-CODE
  DISPLAY TOTAL.
GOBACK.
```

Sample COBOL E35, Variable-Length Records

```
IDENTIFICATION DIVISION.

PROGRAM-ID. E35VL101.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-390.
OBJECT-COMPUTER. IBM-390.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.

DATA DIVISION.
FILE SECTION.

WORKING-STORAGE SECTION.
01  EVEN-FLAG                               PIC 9(2) VALUE ZERO.

01  USER-RETURN-CODE                       PIC 9(8) COMPUTATIONAL.
    88 ACCEPT-REC                           VALUE 0.
    88 DELETE-REC                           VALUE 4.
    88 END-EXIT                             VALUE 8.
    88 INSERT-REC                           VALUE 12.
    88 END-SORT                             VALUE 16.
    88 REPL-REC                             VALUE 20.

01  CHANGE-REC.
    05 C-REC                                PIC X(6) VALUE 'CHANGE'.
    05 C-INCR                               PIC 9(4) VALUE ZERO.
    05 C-BLANK                              PIC X(90) VALUE SPACES.

01  INSRT-REC.
    05 I-REC                                PIC X(6) VALUE 'INSERT'.
    05 I-INCR                               PIC X(4) VALUE ZERO.
    05 I-BLANK                              PIC X(90) VALUE SPACES.

01  TOTAL.
    05 BLANKS                               PIC X(10) VALUE ' E35'.
    05 TITL                                 PIC X(25)
        VALUE 'TOTAL RECORDS HANDLED'.
    05 COUNTER                              PIC 9(8) VALUE 0.
```

Sample COBOL E35, Variable-Length Records (Continued)

```
LINKAGE SECTION.
01  EXIT-STATUS                               PIC 9(8)  COMPUTATIONAL.
    88  FIRST-TIME                             VALUE 00.
    88  MOST-TIME                              VALUE 04.
    88  LAST-TIME                              VALUE 08.

01  RECORD-UP.
    05  RU  OCCURS 1 TO 100 TIMES
          DEPENDING ON LEN-RU                 PIC X.

01  WORK.
    05  WK  OCCURS 1 TO 100 TIMES
          DEPENDING ON LEN-WK                 PIC X.

01  IN-BUF.
    05  IB  OCCURS 1 TO 100 TIMES
          DEPENDING ON LEN-IB                 PIC X.

01  DUMMY PIC X(4) .

01  LEN-RU                                     PIC 9(8)  COMPUTATIONAL.
01  LEN-WK                                     PIC 9(8)  COMPUTATIONAL.
01  LEN-IB                                     PIC 9(8)  COMPUTATIONAL.
```

Sample COBOL E35, Variable-Length Records (Continued)

```
PROCEDURE DIVISION USING EXIT-STATUS, RECORD-UP, WORK,  
IN-BUF, DUMMY, LEN-RU, LEN-WK, LEN-IB.  
  
IF NOT FIRST-TIME  
  ADD 1 TO COUNTER  
  MOVE 0 TO RETURN-CODE.  
  
IF COUNTER LESS THAN 50  
  MOVE 54 TO LEN-WK  
  ADD 1 TO I-INCR  
  MOVE INSRT-REC TO WORK  
  MOVE 12 TO RETURN-CODE  
  GO TO RETURN-TO-SORT.  
  
IF COUNTER LESS THAN 75  
  MOVE 44 TO LEN-WK  
  ADD 1 TO C-INCR  
  MOVE CHANGE-REC TO WORK  
  MOVE 20 TO RETURN-CODE  
  GO TO RETURN-TO-SORT.  
  
IF COUNTER LESS THAN 100  
  MOVE 80 TO LEN-WK  
  ADD 1 TO I-INCR  
  MOVE INSRT-REC TO WORK  
  MOVE 12 TO RETURN-CODE  
  GO TO RETURN-TO-SORT.  
  
IF COUNTER LESS THAN 200  
  MOVE 4 TO RETURN-CODE  
  GO TO RETURN-TO-SORT.  
  
RETURN-TO-SORT.  
  IF LAST-TIME MOVE 8 TO RETURN-CODE  
  DISPLAY TOTAL.  
GOBACK.
```

Coding a C E35 Exit Routine

A C E35 exit program is indicated by the MODS control statement. An E35 exit cannot be coded in C when running Tape Sort.

Like any other E35 exit routine, the C E35 exit routine is called each time a record is brought out of Phase 3. Communication between SyncSort and the C exit takes place through arguments defined in the function header. For example, records are passed to the C routine by an address presented in the second argument in the function parameter list. No storage is reserved in the exit program because the records exist elsewhere.

The C E35 exit routine can be written using either the C370 V2R1 compiler with the V2R2 C370 Library, the SAA AD/Cycle C370 V1R2 Compiler and Library or the C/C++ for MVS/ESA V3R1.1 Compiler and Library. When using the LE/370 run time library modules, it may be necessary to account for this additional storage by adjusting the b value of the Exit-Name parameter on the MODS statement.

Exit Communication

The parameter list structure required for passing fixed-length and variable-length records between the sort and the exit is detailed in the following section. The parameter names used in the examples are arbitrary but each definition is required. Complete sample programs showing the use of the argument lists are presented following the discussion of the exit interface.

Fixed-Length Records - Function Definition

```
int E35exit ( int* exit_status,
             struct_ru* record_up,
             struct_ins_rep* work,
             struct_in_buf* in_buf,
             int* dummy1, int* dummy2, int* dummy3, int* dummy4,
             int* comm_len,
             struct_ca* communication_area)
```

The following describes the parameters used in the preceding definition.

exit_status

This parameter points to a variable containing one of the following exit status codes:

- 00** *First record.* SyncSort uses this Code to indicate the first call to the C exit and that the first record to leave

Phase 3 is in the record_up area. If there are no records to pass to the exit, a 08 status will be passed to the exit on the first call.

- 04** *Most records.* This is used for all calls except the first one when there are records in the record_up area. After Code 00 has been issued, Code 04 is passed to the exit until there is no record for the sort to pass to the record_up area.
- 08** *All records passed.* This indicates that the last record has already been processed by the exit. Do not attempt to reference the record again. No more records will be passed to the exit routine. Note that if SyncSort is not passing any records to Phase 3, 08 will be passed every time including the first time.

record_up	The record_up parameter contains a pointer to the record leaving Phase 3. The struct_ru data type represents a structure that describes the fields within the record.
work	The work parameter contains a pointer to a work area that is to be used to hold an inserted or replaced record returned from the E35. The struct_ins_rep data type represents a structure that describes the fields within the inserted or replaced record.
in_buf	The in_buf parameter contains a pointer to the record that SyncSort is to put in the output data set. Until a record has been accepted or inserted, this pointer will be null. A record at this address can be modified if required.
dummy1 - dummy4	These parameters define unused place holders. They are used with variable-length C E35 communication. Their definition here allows a common parameter list for fixed and variable-length C E15 and E35 exits.
comm_len	This parameter points to a variable that defines the communication area length.
communication_area	The communication_area parameter contains a pointer to the communication area. The struct_ca data type represents a structure that describes the fields in the communication area.

Variable-Length Records - Function Definition

```
int E35exit ( int* exit_status,
             void* record_up,
             void* work,
             void* in_buf,
             int* dummy1,
             int* len_ru,
             int* len_wk,
             int* len_ib,
             int* comm_len,
             struct_ca* communication_area)
```

The following describes the parameters used in the preceding definition.

exit_status	This parameter points to a variable containing exit status codes. See the <code>exit_status</code> definition for a fixed-length C E35 exit for the code definitions.
record_up	The <code>record_up</code> parameter contains a "universal" pointer to the record leaving Phase 3. The <code>void*</code> pointer can be cast to point an appropriate structure to describe the record passed to the exit. This allows different record structures, as is common with variable-length records, to share a universal pointer.
work	The <code>work</code> parameter contains a "universal" pointer to a work area that is to be used to hold an inserted or replaced record returned from the E35. The <code>void*</code> pointer can be cast to point an appropriate structure to describe the work record.
in_buf	The <code>in_buf</code> parameter contains a "universal" pointer to the record that SyncSort is to put in the output data set. Until a record has been accepted or inserted, this pointer will be null. The <code>void*</code> pointer can be cast to point an appropriate structure to describe the work record.
dummy1	This parameter defines an unused place holder.
len_ru	This parameter points to a variable that defines the length of the record leaving Phase 3. This is the length of the record referred to in the <code>record_up</code> parameter.
len_wk	This parameter points to a variable that defines the length of the record to be inserted or used as a replacement for the <code>record_up</code> record. This is the length of the record referred to in the <code>work</code> parameter.

len_ib	This parameter points to a variable that defines the length of the record that SyncSort is to put in the output data set. This is the length of the record referred to in the in_buf parameter.
comm_len	This parameter points to a variable that defines the communication area length.
communication_area	The communication_area parameter contains a pointer to the communication area. The struct_ca data type represents a structure that describes the fields in the communication area.

RETURN-CODE Codes (Fixed and Variable-Length Records)

0	<i>Accept this record.</i> This instructs SyncSort to accept the (unaltered) record in the record_up area.
4	<i>Delete this record.</i> SyncSort will delete the current record in the record_up area.
8	<i>Disconnect E35.</i> This instructs SyncSort to process any remaining records without showing them to the E35 exit. When this return code is used at end-of-file (signalled by exit_status 08), it indicates that the E35 is also finished and will not add additional records. When used before end-of-file, it indicates that SyncSort should process the "current" record passed to the E35, and any subsequent records, as if there were no E35 present. Note that when SyncSort is not creating any output files (SORTOUT or SORTOFxx) and E35 is the only "output," SyncSort terminates immediately, since any subsequent records will never be seen.
12	<i>Insert a record.</i> This instructs SyncSort to add the record in the work area to the input data set just ahead of the current record in the record_up area. When SyncSort returns control to the E35, the same record will be in the record_up area. The exit routine can then add another record from the work area or process the current record in record_up.
16	<i>Terminate SyncSort.</i> SyncSort will terminate and return to the calling program or the Supervisor. SyncSort will issue a completion code of 16 to indicate that the sort was unsuccessful.
20	<i>Replace current record.</i> SyncSort will replace the current record in the record_up area with the record in the work area. Be sure that the record in the work area is valid before passing it to SyncSort.

Change a Record

In order to change the record in the record_up area, first move it to the provided work area. Make the changes there and then pass return code 20. The altered record in the work area will replace the record in record_up.

Sample C E35, Fixed-Length Records

```
#define FIRST_TIME 0
#define MOST_TIME 4
#define LAST_TIME 8
#define ACCEPT_REC 0
#define DELETE_REC 4
#define END_EXIT 8
#define INSERT_REC 12
#define END_SORT 16
#define REPL_REC 20
#include <decimal.h>
typedef _Packed struct record {
    char rec[6];
    decimal(7,0) incr;
    char address[90];
} t_ru;
int counter,i_incr;
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int SMPE35FB(int* exit_status,t_ru* record_up,t_ru* work,t_ru* in_buf,
    int* dummy1,int* dummy2,int* dummy3,int* dummy4,int* comm_len,
    void* communication_area)
{
    int return_code;
    char *text1="CHANGE";
    char *text2="INSERT";
    if (*exit_status != FIRST_TIME) {counter++;
        return_code=ACCEPT_REC;
    }
}
```

Sample C E35, Fixed-Length Records (Continued)

```
if (counter<50) {
    i_incr++;
    work->incr=i_incr;
    strncpy(work->rec,text2,6);
    return_code=INSERT_REC;
    goto return_to_sort;}
if (counter<75) {
    work->incr=record_up->incr+1d;
    strncpy(work->rec,text1,6);
    return_code=REPL_REC;
    goto return_to_sort;}
if (counter<100) {
    i_incr++;
    work->incr=i_incr;
    strncpy(work->rec,text2,6);
    return_code=INSERT_REC;
    goto return_to_sort;}
if (counter<200) {
    return_code=DELETE_REC;
    goto return_to_sort;}
return_to_sort:
    if (*exit_status==LAST_TIME)
        { return_code=END_EXIT;
          printf("E35 total number of records handled:%d\n",counter);
        }
    return(return_code);
}
```

Sample C E35, Variable-Length Records

```
#define FIRST_TIME 0
#define MOST_TIME 4
#define LAST_TIME 8
#define ACCEPT_REC 0
#define DELETE_REC 4
#define END_EXIT 8
#define INSERT_REC 12
#define END_SORT 16
#define REPL_REC 20
#define MAX_RLEN 104
typedef _Packed struct record1 {
    char rec[6];
    int incr;
    char address[MAX_RLEN-14];
} t_ru1;
typedef _Packed struct record2 {
    char title[10];
    int number;
} t_ru2;
int counter,i_incr,i_number;
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int SMPE35VB(int* exit_status,void* record_up,void* work,void* in_buf,
    int* dummy,int* len_ru,int* len_wk,int* len_ib,int* comm_len,
    void* communication_area)
{
```

Sample C E35, Variable-Length Records (Continued)

```
int return_code;
char *text1="CHANGE E35";
char *text2="INSERT E35";
t_ru1 * p_record1,*pwork1;
t_ru2 * p_record2,*pwork2;
p_record1 = (t_ru1 *)record_up;
pwork1=(t_ru1 *)work;
p_record2 = (t_ru2 *)record_up;
pwork2=(t_ru2 *)work;
if (* exit_status != FIRST_TIME) {counter++;
    return_code=ACCEPT_REC;}
if (counter<50) {
    if (*len_ru == 14)
    {
        *len_wk = 14;
        i_number++;
        pwork2->number=i_number;
        strncpy(pwork2->title,text2,10);
    } else
    {
        *len_wk = 54;
        i_incr++;
        pwork1->incr=i_incr;
        strncpy(pwork1->rec,text2,6);
    }
    return_code=INSERT_REC;
    goto return_to_sort;}
if (counter<75) {
    if (*len_ru == 14)
    {
        *len_wk = 14;
        pwork2->number=p_record2->number+1;
        strncpy(pwork2->title,text1,10);
    } else
    {
        *len_wk = 54;
        pwork1->incr=p_record1->incr+1;
        strncpy(pwork1->rec,text1,6);
    }
    return_code=REPL_REC;
    goto return_to_sort;}
```

Sample C E35, Variable-Length Records (Continued)

```
if (counter<100) {
    if (*len_ru == 14)
    {
        *len_wk = 14;
        i_number++;
        pwork2->number=i_number;
        strncpy(pwork2->title,text2,10);
    } else
    {
        *len_wk = 80;
        i_incr++;
        pwork1->incr=i_incr;
        strncpy(pwork1->rec,text2,6);
    }
    return_code=INSERT_REC;
    goto return_to_sort;}
if (counter<200) {
    return_code=DELETE_REC;
    goto return_to_sort;}
return_to_sort:

    if (*exit_status==LAST_TIME)
    { return_code=END_EXIT;
      printf("E35 total number of records handled:%d\n",counter);
    }
    return(return_code);
}
```

Exit E16-Taking Action on Insufficient Intermediate Storage

Exit E16 is given control in the event that the input data set is unable to fit into intermediate storage. There is no parameter list. The E16 return code tells SyncSort how to respond to the insufficient SORTWK problem.

Return Codes

- 0** *Sort present records only.* This instructs SyncSort to process only those records presently contained on the intermediate storage devices. The sort will receive message WER054I RCD IN xxxxxxxx, OUT yyyyyyyy if the data is read from SORTIN directly, or WER055I INSERT xxxxxxxx, DELETE yyyyyyyy if the data receives input exit (E14 or E15) or INCLUDE/OMIT processing. The message's RCD IN or INSERT xxxxxxxx figure indicates how many records have been sorted. For a sort with no

exits or INCLUDE/OMIT processing, the remaining records may be sorted by running another job using the SKIPREC=n parameter on the SORT control statement, skipping the xxxxxxxx number of records. The new sort will start just where the last one left off. The final output is obtained by running a MERGE with the two SORTOUT data sets.

- 4 *Try to sort all records.* This tells SyncSort to continue to read in records from the input data set. If there are very few records left, the sort may complete successfully. If there are too many records to continue the sort, SyncSort will terminate with a SORT CAPACITY EXCEEDED message.

- 12 *Terminate SyncSort.* SyncSort will terminate immediately with a SORT CAPACITY EXCEEDED message.

Exits E17, E27, and E37 - Closing Data Sets

These exits are unusual in that they are entered only once, at the end of their associated phase. Because of this, they may be efficiently used to clean up after other exit routines (e.g., to close data sets). There are no parameter lists or return codes for these exits.

Exits E18, E38, and E39 - Checking Labels, Processing Read or Write Errors, End-of-File Routines, Special VSAM Processing

These exits are mainly used for I/O error recovery routines. However, they may also be used to check labels, to do end-of-file processing, and to provide various information to the VSAM access method.

Exit E18 and E38 Programs

Exit E18 is *only* used for sorts and exit E38 *only* for merges or copies. Each exit is entered exactly once, at the start of SORTIN processing. At this time, SyncSort checks Register 1 for the address of a user parameter list specifying the various open and error exit routines the user wishes SyncSort to include. SyncSort will then enter these routines at the appropriate times during execution. Because use to these exits forces the use of BSAM for the input file(s), performance may be adversely affected.

The format of the parameter list is given below. More information on the DCB fields can be found in the appropriate IBM publication.

Byte 1	Byte 2	Byte 3	Byte 4
01	SYNAD field		
02	EXLST field		
03	00	00	EROPT code
04	EODAD field		
00	00	00	00

Table 34. Parameter List for E18 and E38

The parameter list must begin on a fullword boundary and consist of an integral number of words. With the exception of the required fullword of zeros used to indicate the end of the parameter list, entries are optional. The first byte of each word identifies the parameter:

SYNAD field Indicated by 01 in byte 1. The SYNAD field contains the address of a synchronous read error routine, assembled as part of the exit program. Note that you may not use Register 13 as a save area pointer on entry to your routine. You must either provide your own save area or use the SYNADAF macro instruction.

EXLST field Indicated by 02 in byte 1. The EXLST field contains the address of a list of pointers to user routines that perform operations such as label checking. Note that in the event that the list contains a DCB-exit entry, it will not be entered during concatenated SORTIN processing.

EROPT code Indicated by 03 in byte 1. Bytes 2 and 3 contain zeros, byte 4 the EROPT code. This code tells SyncSort what action to take if it discovers an uncorrectable read error on a non-VSAM input file.

X'00' Follow the EROPT code in the DCB parameter of the DD statement that describes the data set containing the error.

X'20' Terminate the program.

X'40' Skip the block containing the error.

X'80' Accept the block containing the error.

EODAD field Indicated by 04 in byte 1. The EODAD field contains the address of an end-of-file routine. It can only be used with an E18 exit.

VSAM Input to E18 and E38

With VSAM input, these exits can be used to pass the addresses of various VSAM exits or to insert passwords into VSAM input ACB's. When control is returned to the sort, Register 1 must contain the address of a parameter list:

X'05'	3 byte address of VSAM exit list
X'06'	3-byte address of password list
F'0'	Fullword of zeros

Table 35. Sample VSAM Parameter List for E18 and E38

If both address entries are present, they may be in either order. Only one need be present. (QSAM parameters will be ignored.) The password list referenced in the parameter list is found in the exit routine and is formatted as follows:

2 bytes on a halfword boundary:

Number n of entries in list

Followed by n 16-byte entries:

8-byte DDname
8-byte Password

The exit routine must not alter this list. The sort may destroy the last byte of the DD name field.

The exit list is built using the VSAM EXLST macro, which provides the addresses of the VSAM exit routines. VSAM branches directly to the routines which must return to VSAM via the address in Register 14.

To do EODAD processing with E38, write a LERAD exit and check for X'04' in the FDBK field of the RPL: this indicates input EOD. This field is needed by the merge, so it should not be altered when returning to VSAM.

The following example shows how to code the return to the sort.

```

ENTRY E18
.
.
.
E18    LA    1,PARMLST
      ST    1,24(13)    R13 points to sort's save area
      LM    14,12,12(13)
      BR    14
      CNOP  0,4
PARMLST DC  X'01'
      DC  AL3(BSAMERR)
      DC  X'02'
      DC  AL3(EXLST)
      DC  X'03'
      DC  X'000080'    EROPT code
      DC  X'04'
      DC  AL3(BSAMEOD)
      DC  X'05'
      DC  AL3(VSAMEXL)
      DC  X'06'
      DC  AL3(PWDLST)
      DC  A(0)
.
.
.
VSAMEXL EXLST SYNAD=SYNAD, LERAD=LERAD
PWDLST  DC  H'2'
      DC  CL8'SORTIN'    SORTIN ddname
      DC  CL8'INPASS'    SORTIN password
      DC  CL8'SORTOUT'    SORTOUT ddname
      DC  CL8'OUTPASS'    SORTOUT password
SYNAD   ...    VSAM synch error rtn
LERAD   ...    VSAM logic error rtn
BSAMERR ...    BSAM error rtn
EXLST   ...    EXLST address list
BSAMEOD ...    BSAM end of data rtn

```

Figure 188. Sample E18 Program

Exit E39 Programs

Exit E39 is used mainly for SORTOUT write error routines. The exit is entered once at the beginning of merge or copy processing or the start of sort Phase 3. At this time SyncSort checks Register 1 for the address of a user parameter list specifying the various routines

the user wishes SyncSort to include. SyncSort will then enter these routines at the appropriate times during execution. The use of an E39 exit forces the use of BSAM on the output file; this may degrade performance somewhat.

The format of the parameter list is given below.

Byte 1	Byte 2	Byte 3	Byte 4
01	SYNAD field		
02	EXLST field		
00	00	00	00

Table 36. Parameter List for E39

The parameter list must begin on a fullword boundary and consist of an integral number of words. With the exception of the required fullword of zeros used to indicate the end of the parameter list, entries are optional. The first byte of each word identifies the parameter:

SYNAD field Indicated by 01 in byte 1. The SYNAD field contains the address of a synchronous write error routine, assembled as part of the exit program. Note that you may not use Register 13 as a save area pointer on entry to your routine. You must either provide your own save area or use the SYNADAF macro instruction.

EXLST field Indicated by 02 in byte 1. The EXLST field contains the address of a list of pointers to user routines that perform operations such as label checking. If the EXLST field is specified, CHECKPOINT processing will not be performed by SyncSort.

Exit E39 may be used to supply a VSAM exit list or password list for the output file in the same manner as described for exits E18 and E38. Note that unlike E18 there is no EODAD field with this exit.

Exit E61 - Modifying the Collating Process

Exit 61 is used to alter the collating of all control fields specified as having an o (order) value of E in the SORT/MERGE control statement. Note that an E61 exit routine is called in Phase 1 for sort applications and in Phase 3 for merge applications. Each time SyncSort encounters an order E control field, it moves a copy of the control field to a work area and passes the copy's address to the exit routine. Thus, the E61 exit program processes a control field image while leaving the original control field intact. An order E control field is collated in ascending order according to its f (format) code and its E61 image. In order to code an effective E61 routine, the user must be familiar with the standard data formats used by the operating system.

For all order E control fields except BInary fields, the number of bytes in the control field image will be the number specified as the l (length) value on the SORT/MERGE control

statement. BInary fields are left and right padded with zeros to the nearest byte boundary. For example, a control field designated as 5.3,1.4,BI,E receives three bits of padding on the left, one on the right, producing an image 2 bytes long.

An E61 exit can process only the first 256 bytes of the control field image in a single pass. If a control field image is more than 256 bytes long, the exit will be entered more than once for that control field.

If AC is specified as the format of a control field on the SORT or MERGE statement, SyncSort will translate the field to ASCII before the E61 routine is given control. In order to use an E61 routine to modify what would be an AC control field, specify the field as CH in the SORT or MERGE statement and translate the image to ASCII after it is altered by the E61 exit routine.

There is no advantage to coding an E61 exit if the ALTSEQ control statement can provide the needed collating modification. ALTSEQ changes the installation's alternate collating sequence, used for all control fields specified with the format code AQ.

An E61 exit cannot be used with locale processing (LOCALE option enabled).

The Parameter List

Each time your routine is executed, SyncSort will place the address of a three-word parameter list in Register 1. The parameter list will be on a fullword boundary. The first word contains the number of the control field within the record in byte 4. The second word contains the address of the control field in the work area in bytes 2, 3, and 4. The third word contains the length of the control field in bytes 3 and 4. All values are given in hexadecimal and the unused bytes are filled with zeros.

Byte 1	Byte 2	Byte 3	Byte 4
00	00	00	Number of control field
00	Address of control field in work area		
00	00	Length of control field	

Table 37. Parameter List for E61

Lengthening a Control Field Image

The length of the control field image is completely determined by the length and format code of the control field. Therefore, in order to provide a 12-byte PD image of 5 bytes from the original record, it is necessary for the SORT/MERGE control statement to reference a 12-byte PD control field that contains the 5 desired bytes. The extra 7 bytes are used to contain the "lengthened" image.

Shortening a Control Field Image

The length of the control field image is completely determined by the length and format code of the control field. To shorten a control field image, specify the full length of the original control field as the l (length) value in the SORT/MERGE control statement. Then shorten each image by the same number of bytes and pad it uniformly to the length of the original field. Be sure to pad each control field image with the same leading or trailing character, and replace data in the control field image with the same type of data as that in the actual control field.

Reversing a Collating Sequence

Every order E control field is collated according to its image and format code, in ascending order. To collate the field in apparent descending order, complement the control field image according to its format code before returning control to SyncSort. For a BI or CH field, for example, complement the image with hexadecimal FF's before returning control to the sort.

Coding REXX Exits

The exit routines E15 and E35 can be coded in REXX.

REXX Variables Provided by SyncSort for z/OS

SyncSort for z/OS provides a number of special REXX variables to facilitate the development of REXX exits. These variables offer a simple, efficient means of establishing communication between the exit and the sort/merge.

To load these variables, the following command must be used when the exit is called.

```
ADDRESS 'SYNCREXX' 'GIVE'
```

When the exit completes its work, the exit should use the following sequence of commands to return the variables to SyncSort for z/OS.

```
ADDRESS 'SYNCREXX' 'TAKE'  
RETURN
```

The following table describes the special REXX variables.

Variable	Function
SYRECORD	<p>When the exit is entered, SYRECORD contains the current data record. The exit can accept the record, modify it or add a new record; SYACTION should be set accordingly.</p> <p>If SYRECORD is null, then SyncSort for z/OS has no data remaining. When this happens, the exit can either CLOSE or continue to INSERT new records.</p>
SYACTION	<p>This variable must be set before the exit returns control to SyncSort for z/OS. It describes the disposition of the current record. Possible values for SYACTION are as follows:</p> <p>ACCEPT: Retain the current record with no modification.</p> <p>REPLACE: Replace the current record with the contents of the SYRECORD.</p> <p>DELETE: Delete the current record.</p> <p>INSERT: Insert the contents of the SYRECORD before the current record.</p> <p>CLOSE: Do not return to the exit.</p> <p>ABEND: Terminate SyncSort for z/OS.</p> <p>If an E15 is providing all the input (SORTIN not present), the only valid values for SYACTION are INSERT, CLOSE or ABEND.</p>
SYEXITYP	<p>This variable will automatically be set to E15 or E35, depending on which type of exit is being called.</p>
SYGBLN1... ...SYGBLN8	<p>These eight special variables are global variables. The user may set these to any value provided that the value does not exceed 15 characters in length. SyncSort for z/OS will insure that these variables are preserved across calls to the exit.</p>
SYGBLSTR	<p>This is an additional global variable. The user may set this to any value, provided the string does not exceed 1024 characters in length. SyncSort for z/OS will insure that this variable is preserved across calls to the exit.</p>

Table 38. REXX Variables Provided by SyncSort for z/OS

Sample REXX Exit

The following example illustrates a REXX exit that will count the number of records that are passed to the exit:

```
address 'SYNCREXX' 'GIVE'  
if sygbln1='SYGBLN1' then sygbln1=0  
if LENGTH(syrecord) > 0  
  then do  
    syaction='REPLACE'  
    sygbln1=sygbln1 + 1  
  end  
else do  
  syaction='CLOSE'  
  say 'REXX' syexityp 'counted' sygbln1 'records'  
end  
address 'SYNCREXX' 'TAKE'  
return
```

Figure 189. Sample REXX Exit Code

Chapter 8. The Flow of the Sort

This chapter briefly outlines the flow of control in the standard Disk Sort, incore sort, merge and copy. It describes the order in which SyncSort will process and act on the PARMs, control statements and exit routines provided by the user. Note that all executions begin with Phase 0 processing and that a given SyncSort execution will skip steps where appropriate (e.g., will skip a "Variable-length record sampling" step if sorting fixed-length records or HISTOGRM length values are supplied). No attempt has been made to indicate which steps are required of all Disk Sorts, incore sorts, etc., or to indicate the nature or timing of any abend processing.

Phase 0

- Process PARMs, merging EXEC and \$ORTPARM PARM specifications. The EXEC statement/invoking program's parameter list overrides the installation defaults. \$ORTPARM overrides the EXEC statement/invoking program's parameter list.
- Process control statements (from the \$ORTPARM DD statement and either the SYSIN DD statement or the invoking program's parameter list).
- Link-edit user exits (if necessary).
- Validate SORTIN/SORTINnn and SORTOUT/SORTOFxx/SORTOFx/SORTXSUM DCB attributes.

- If MERGE or COPY, GO TO →

Non-Sorting
Phase 3

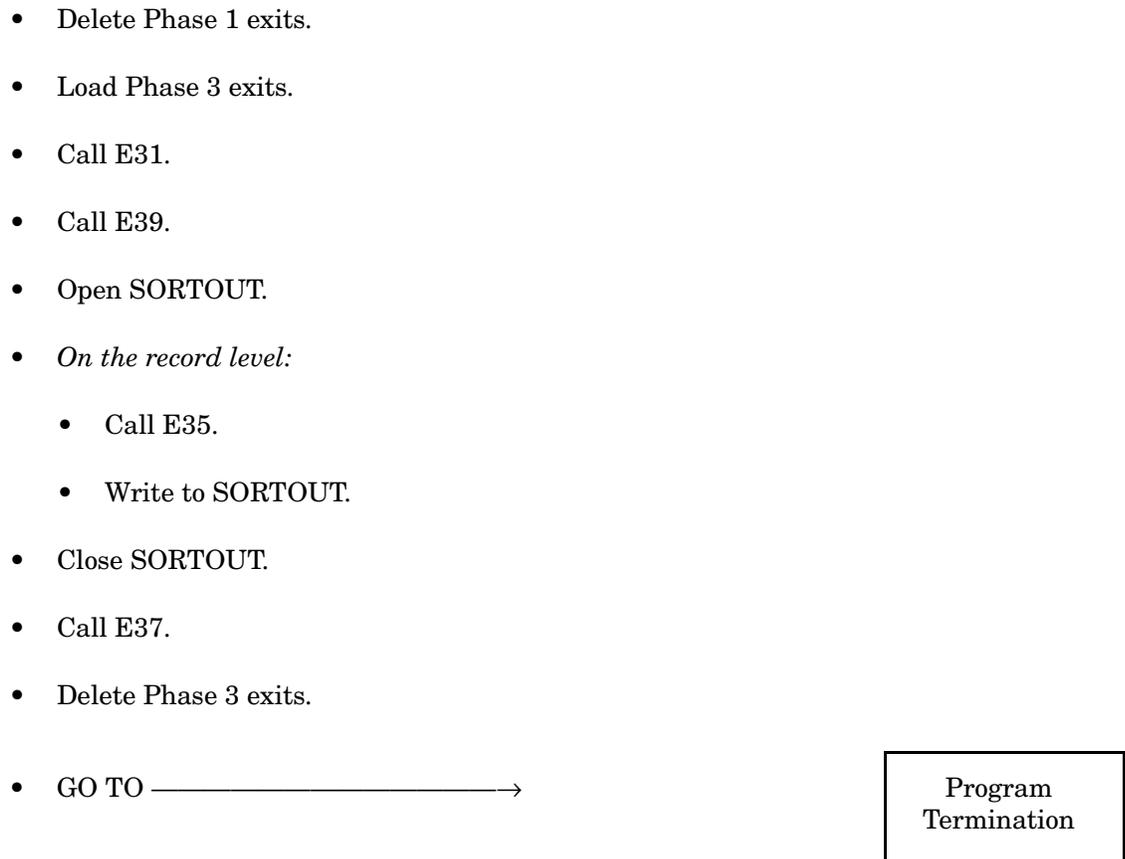
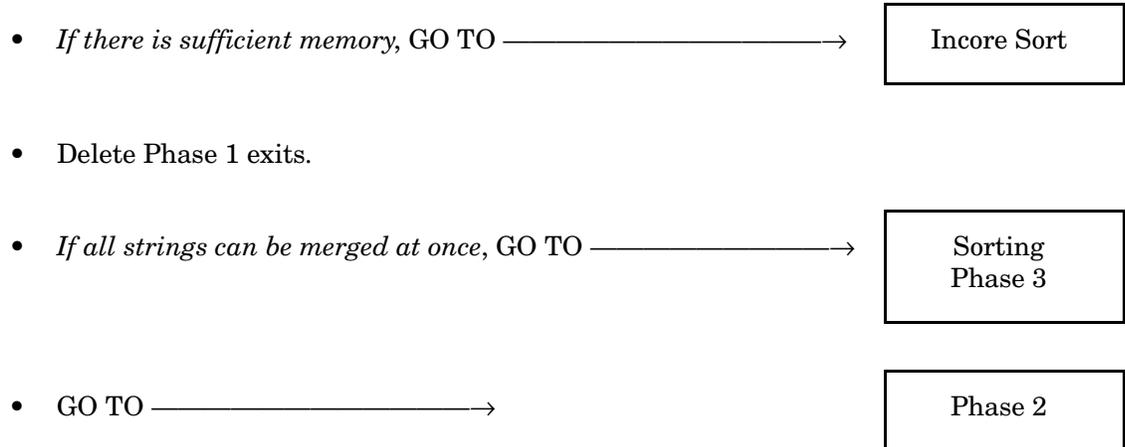
- variable-length record sampling: open SORTIN, do the sampling, close SORTIN.

- GO TO →

Phase 1

Phase 1

- Load Phase 1 exits.
- Call E11.
- Call E18.
- Open SORTIN.
- Perform SKIPREC processing.
- *On the record level:*
 - Read from SORTIN or DB2 database for DB2 query.
 - Call E15.
 - Perform INCLUDE/OMIT processing.
 - Perform INREC processing.
 - Perform STOPAFT processing.
 - Call E61.
 - Perform SUM processing.
 - Call E14.
- Call E16.
- Close SORTIN.
- Call E17.



- Load Phase 2 exits.
- Call E21.
- *On the record level:*
 - Call E25.
 - Perform SUM processing.
- Call E27.
- Delete Phase 2 exits.
- GO TO —————→

Sorting Phase 3

Sorting Phase 3

- Load Phase 3 exits.
- Take checkpoint.
- Call E31.
- Call E39.
- Open SORTOUT, SORTOFxx, SORTOFx and SORTXSUM.
- *On the record level:*
 - Perform SUM processing.
 - Write to SORTXSUM.
 - Perform OUTREC processing.
 - Call E35.
 - *If SORTOUT, SORTOFxx or SORTOFx are present, then for each output data set:*
 - Perform STARTREC/ENDREC processing.
 - Perform INCLUDE/OMIT parameter processing.

- Perform SortWriter functions.
- Perform OUTREC processing.
- Perform ANSI control character processing.
- Write to SORTOUT, SORTOFxx, or SORTOFx.
- Call E37.
- Close all data sets.
- Delete Phase 3 exits.
- GO TO →

Program
Termination

Non-Sorting
Phase 3

- Load user exits for the merge/copy.
- Call E31.
- Call E38.
- Call E39.
- Open all data sets.
- Perform SKIPREC processing (for a copy).
- *On the record level:*
 - Read from SORTIN/SORTINnn or DB2 database for DB2 query or (for a merge) call E32 for a record.
 - Call E15 (for a copy).
 - Perform INCLUDE/OMIT processing.
 - Perform INREC processing.
 - Perform STOPAFT processing (for a copy).
 - Call E61 (for a merge).

- Perform SUM processing (for a merge).
- Write to SORTXSUM (for a merge).
- Perform OUTREC processing.
- Call E35.
- *If SORTOUT, SORTOFxx or SORTOFx are present, then for each output file:*
 - Perform STATREC/ENDREC processing.
 - Perform INCLUDE/OMIT parameter processing.
 - Perform SortWriter functions.
 - Perform OUTREC processing.
 - Perform ANSI control character processing.
 - Write to SORTOUT, SORTOFxx or SORTOFx.
- Call E37.
- Close all data sets.
- Delete user exits.
- GO TO —————→

Program Termination

Program Termination

- Print SyncSort messages.
- END.

Chapter 9. MAXSORT

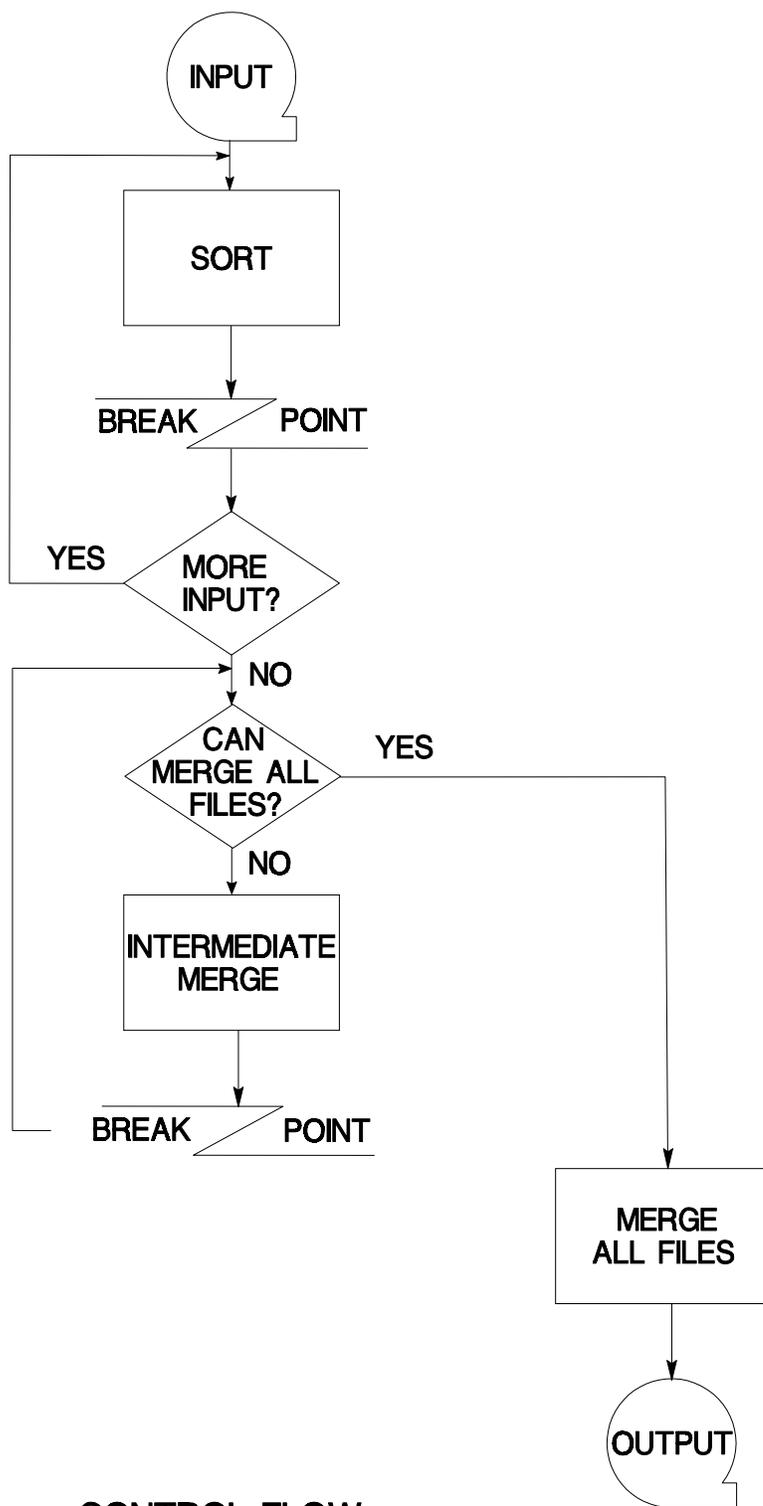
MAXSORT: A Maximum Capacity Sort

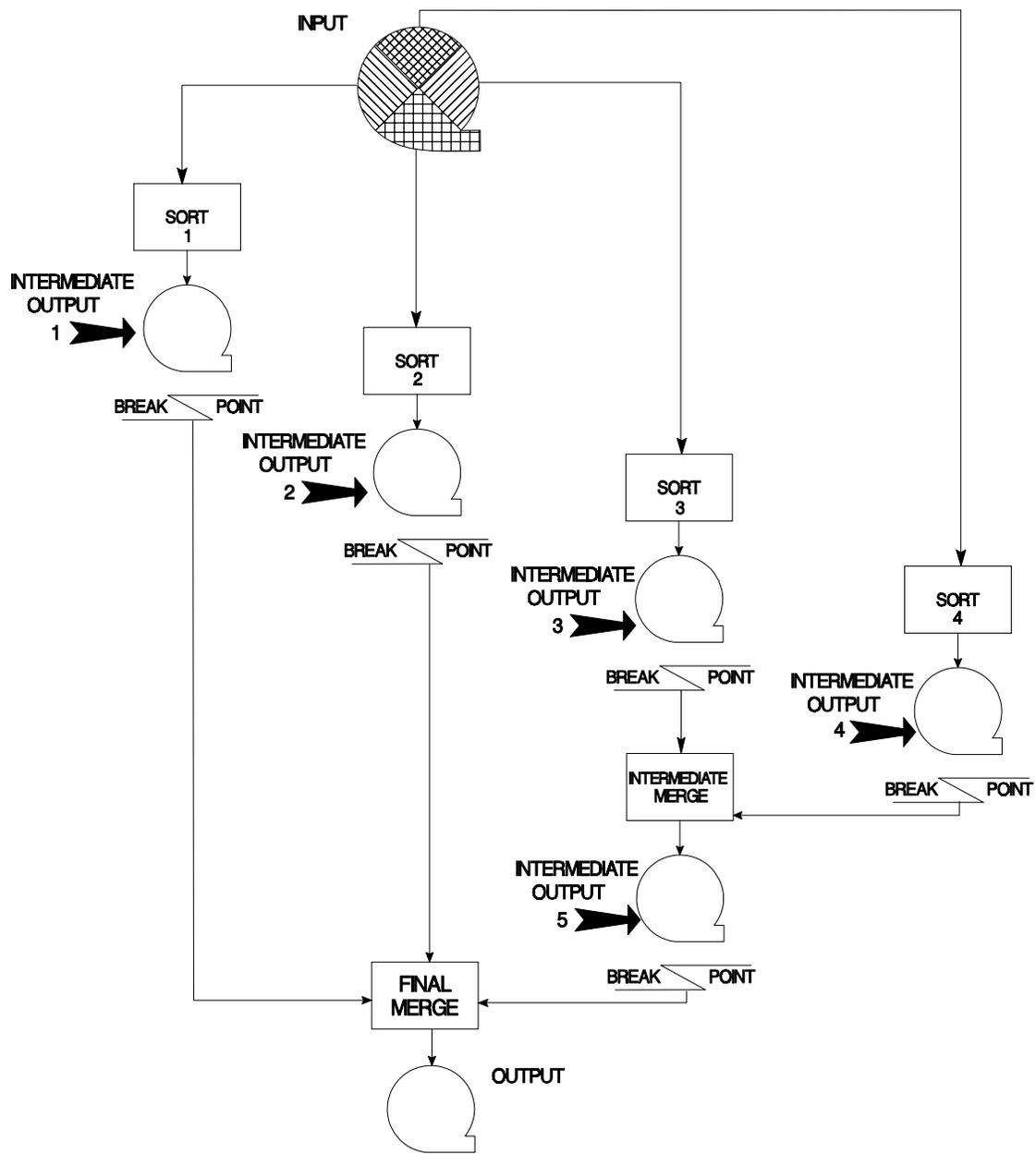
MAXSORT is a maximum capacity sort designed to sort amounts of data that are too large for an ordinary sorting technique to process.

MAXSORT breaks up the sorting process into small, individual sorts. At the end of each individual sort a natural breakpoint occurs. At this time, the sorted data is written out on intermediate storage devices and it becomes possible to stop the program without losing the results of the previous processing. At each breakpoint, an operator may intervene to change program options.

When all the input to the sort has been read and has become individual sorted data sets, this output becomes input to one or more merges. If all the data sets can be merged at once, one final merge is performed. If all the data sets cannot be merged at once, some of them will be combined in one or more intermediate merges. Then, when all the data sets can be merged at one time, the final merge is performed and the final sorted output is produced.

The diagrams on the following two pages illustrate the MAXSORT technique.





DATA FLOW

MAXSORT's Advantages

- Without MAXSORT, overlarge sorts require tape work areas or force the user to segment the input and execute multiple disk sorts. With MAXSORT, any input data set can be handled by *one* sort execution using disk work space.
- MAXSORT requires *less* disk space than ordinary sorts. Because MAXSORT stores the output of each individual sort on tape, the same disk SORTWK files can be used over and over again.
- Since the output of each individual sort is a completely sorted data set, the original job may be interrupted for higher priority jobs without wasting processing time.
- If a system or program failure occurs, whatever data sets have already been produced are still usable. The job can be restarted at the last breakpoint, and all previously produced data sets can be used without resorting.

Job Control Language

MAXSORT and Disk Sort have similar JCL requirements. To initiate MAXSORT using job control statements, specify PARM='MAXSORT' on the EXEC statement. A program-initiated sort requests MAXSORT by using a PARM card image in the data set defined by the \$ORTPARM DD statement. In either case, it may be necessary to request additional main storage in order to use the MAXSORT technique.

Sample EXEC Statement

```
//stepname EXEC { PGM=SYNCSORT  
                 PGM=SORT  
                 PGM=IERRCO00  
                 PGM=IGHRCO00  
                 PGM=ICEMAN }
```

Figure 190. MAXSORT EXEC Statement

DD Statements

MAXSORT's DD statement requirements are summarized in the following table. As many as three additional types of DD statements may be needed. Note that SORTWK files *must* be allocated only to disk devices.

MAXSORT DD Statements		
//\$ORTPARM	DD	Used to override PARM or control statement information.
//SYSIN	DD	Control statement data set. Required unless the address of a 24-bit or 31-bit extended parameter list is supplied by an invoking program.
//SYSOUT	DD	Message data set. Required unless all messages are routed to console.
//SORTWKnn	DD	Disk work area definition. Required unless DYNALLOC is specified or MAXSORT is restarted at a MERGE breakpoint.
//SORTIN	DD	SORT input data set. Required unless there is an E15. Ignored if the invoking program supplies an inline E15 exit routine; optional if the MODS statement activates an E15 exit routine.
//SORTOUT	DD	Output data set. Required unless there is an E35. Ignored if the invoking program supplies an inline E35 exit routine; optional if the MODS statement activates an E35 exit routine.
//SORTXSUM	DD	Output data set for records deleted by SUM. Required if XSUM parameter used.
//SORTBKPT	DD	Breakpoint data set. Must be DASD. Required.
//SORTOU00	DD	Required if intermediate output is on tape.
//SORTOUnn	DD	Required if intermediate output is on disk or if intermediate output is on tape and DYNATAPE is not specified.
//SORTCKPT	DD	Checkpoint data set. Required if Checkpoint-Restart is used.
//SORTMODS	DD	Required if user exits are in SYSIN and if user exits are to be linkage-edited at execution time.
//SYSLIN	DD	
//SYSLMOD	DD	
//SYSUT1	DD	
//SYSPRINT	DD	
//ddname	DD	Required for exits unless the exit is inline in LINKLIB/JOBLIB/STEPLIB or in SYSIN.

Table 39. MAXSORT DD Statements

When the RELEASE=ON parameter is active (either via specification or by default) at the conclusion of the sort portion of a MAXSORT, most of the allocated SORTWK space is free (however, in the case of invoked sorts or SORTWKs defined as OLD, the data set is returned to the size allocated at MAXSORT initiation rather than the minimum size possible).

The SORTBKPT, SORTOU00, SORTOU_{nn} and SORTCKPT DD statements are discussed below. Refer to “Chapter 4. JCL and Sample JCL/Control Statement Streams” for a discussion of the other DD statements, which are specified for MAXSORT just as they would be specified for Disk Sort.

SORTBKPT DD Statement

The required SORTBKPT statement defines the breakpoint data set on which the sort control information is stored. At the end of each individual sort or merge, a breakpoint is reached and the information in the breakpoint data set is automatically amended. However, when MAXSORT is program-invoked or when any exit other than an E35 exit is being used, breakpoints cannot be taken. Nevertheless, the SORTBKPT statement must be specified for all MAXSORTs, even those for which a breakpoint restart is not possible.

Allocating Disk Space for the Breakpoint Data Set

The breakpoint data set must be allocated on a direct access device. It is recommended that space for the breakpoint data set be allocated in the job step preceding the sort step. Or, the breakpoint data set may be pre-allocated in a separate job.

Sample Allocation of the Breakpoint Data Set

The following example illustrates how disk space for a breakpoint data set might be allocated as part of a MAXSORT job control stream. These two statements would follow the JOB statement:

```
//ALLOC      EXEC PGM=IEFBR14
//BKPTDATA  DD      DSN=BKPT.DATA,DISP=(NEW,CATLG),UNIT=SYSDA,
//          SPACE=(1000,(100,50))
```

Figure 191. Sample Job Step Allocating Disk Space for a Breakpoint Data Set

In this example, the name of the breakpoint data set is BKPT.DATA. Because this data set must be kept until MAXSORT has completed, DISP=(NEW,CATLG) has been specified. Supplying approximately 100K bytes of primary space and allowing for secondary allocation should be adequate.

Sample SORTBKPT DD Statement

The sample SORTBKPT DD statement which follows identifies the breakpoint data set for which disk space has already been allocated.

```
//SORTBKPT  DD      DSN=BKPT.DATA,DISP=(OLD,KEEP)
```

Figure 192. Sample SORTBKPT DD Statement

This SORTBKPT DD statement defines the breakpoint data set for which disk space has previously been allocated. The data set name must be the same name which was specified when the space for the breakpoint data set was allocated. DISP=(OLD,KEEP) is specified

so that this statement does not have to be changed if MAXSORT is restarted. The DCB information should *not* be coded because MAXSORT supplies these values. If a VOLSER was coded when the disk space for the breakpoint data set was allocated, the same VOLSER must be specified in the SORTBKPT DD statement.

SORTOU00 DD Statement

The SORTOU00 DD statement is required for every MAXSORT application in which the intermediate output is stored on tape.

The SORTOU00 DD statement defines the unit to be used for the output of the individual sorts and intermediate merges. MAXSORT will create and name these data sets which will eventually be merged to produce the final sorted output.

The data set names generated will have one of the two formats. If the TIMESTMP option is not specified (the installation default), data set names will have the format:

TDS.jobname. $\left\{ \begin{array}{l} \text{Snn} \\ \text{Mnn} \end{array} \right\}$

If the TIMESMP option *is* specified at installation time, data set names will have the format:

TDS.Ddddhhmm.jobname. $\left\{ \begin{array}{l} \text{Snn} \\ \text{Mnn} \end{array} \right\}$
--

In either case, the S or M indicates whether the output is from the sort phase or from the merge phase; nn is the relative number of the data set (01 to 99). The Ddddhhmm time stamp refers to the time (Julian day, hour and minute) the sort began. The prefix default (TDS.) can be changed by specifying the BKPTDSN PARM option.

The following rules should be observed in coding the SORTOU00 DD statement:

- Specify DISP=(NEW,KEEP) and a permanent DSNAME and VOL=PRIVATE so that a scratch tape is used and the volumes are unloaded. Failure to specify this can result in the rewinding of the scratch tape and overwriting of the intermediate sort output.
- Specify the DEFER option in the UNIT parameter so that mount messages to the operator that do not pertain to MAXSORT are suppressed.
- SORTOU00 and SORTIN cannot share the same tape unit. However, SORTOU00 and SORTOUT *may* share the same tape unit unless the DYNATAPE PARM is specified. SORTIN and SORTOUT may always share the same unit.

- If DYNATAPE is in effect, the tape unit name must be the same as the unit name specified in the TAPENAME PARM.

Sample SORTOU00 DD Statement

```
//SORTOU00 DD DSN=PERM.OU00,DISP=(NEW,KEEP),
//          UNIT=(TAPE,,DEFER),VOL=PRIVATE
```

Figure 193. Sample SORTOU00 DD Statement

SORTOU_{nn} DD Statements

The SORTOU_{nn} DD statements allocate the tape units used as input to the merge phase. They are required unless the DYNATAPE option (available only under z/OS) is used.

Although it is not necessary to specify the SORTOU_{nn} DD statements when DYNATAPE is used, it is a good idea to pre-allocate *at least two* SORTOU_{nn} data sets when the DYNATAPE option is specified. This ensures that the minimum required number of tape units will be available for the merge phase. When additional units are available, DYNATAPE will provide performance benefits.

The following rules should be observed in coding the SORTOU_{nn} DD statements:

- At least *two* tape units must be allocated in the absence of DYNATAPE. However, allocating more units will make the merge phase complete more quickly.
- Each statement must be allocated to a unique tape drive. If DYNATAPE is in effect, the tape unit name must be the same as the unit name specified in the TAPENAME PARM.
- All the tape units must operate at the same recording density. For best performance, multiple density units should be run at the highest density.
- For each SORTOU_{nn} statement, replace the 'nn' with a two digit number between 01 and 99. The numbers need not be consecutive.
- Specify DISP=(NEW,KEEP), a permanent DSNAME and VOL=PRIVATE so that a scratch tape is used and the tape volumes are unloaded.
- Specify the DEFER option in the UNIT parameter so that mount messages to the operator that do not pertain to MAXSORT are suppressed.

Sample SORTOU_{nn} DD Statements

```
//SORTOU01 DD DSN=PERM.OU01,DISP=(NEW,KEEP),
//          UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SORTOU02 DD DSN=PERM.OU02,DISP=(NEW,KEEP),
//          UNIT=(TAPE,,DEFER),VOL=PRIVATE
```

Figure 194. Sample SORTOU_{nn} DD Statements

Using Disk for Intermediate Output

In most cases, tape units will be used to store the intermediate output of the individual sorts and intermediate merges. However, it may be desirable in some circumstances to place intermediate output on disk (e.g., to take advantage of a mass storage subsystem). Assigning the intermediate output to a mass storage subsystem will not compromise the sort's efficiency. Because these files will be written and read sequentially, paging will be minimal.

If disk is used for intermediate storage, the following rules should be observed:

- The SORTOU00 statement should *not* be coded.
- SORTOU_{nn} statements may be allocated to real or MSS virtual volumes.
- To determine how many SORTOU_{nn} DD statements to supply, divide the total number of bytes of sort input data by the number of bytes of SORTWK space and add 2 to the result. This figure is the number of SORTOU_{nn} statements to supply. If too few SORTOU_{nn} DD statements are supplied, MAXSORT will terminate for restart at the point at which a new DD statement is needed.
- Each SORTOU_{nn} DD statement must allocate enough primary and secondary space to hold *all* of the data written during that intermediate sort.
- Track overflow is not supported for disk SORTOU_{nn} data sets. Record lengths must not exceed the track capacity unless VS or VBS records are being processed.

SORTCKPT DD Statement

This DD statement is required in order to restart a MAXSORT which is task-invoked or includes a user exit routine because in these cases MAXSORT cannot be restarted from a breakpoint. The standard OS/VS Checkpoint-Restart feature is used. Both automatic Checkpoint-Restart and deferred Checkpoint-Restart capabilities are supported (see “Chapter 13. Performance Considerations”). Checkpoints are taken at the end of each intermediate sort or merge.

The SORTBKPT DD statement must be specified in addition to the SORTCKPT DD statement even when MAXSORT cannot be restarted from a breakpoint.

A MAXSORT with an E35 exit does *not* require the SORTCKPT DD statement.

Control Statements

Control statements will only be accepted at the initial execution of MAXSORT. If MAXSORT is restarted, the control statements cannot be changed. Except for MERGE and OUTFIL, all SyncSort control statements are fully supported for MAXSORT.

PARM Options

The MAXSORT parameters described below may be specified on the EXEC statement, the \$ORTPARM DD statement, PARMTBLE or PARMEXIT, and may be listed in any order.

BKPTDSN

$\text{BKPTDSN} = \left\{ \begin{array}{l} \text{cc...c.} \\ \underline{\text{TDS.}} \end{array} \right\}$
--

The BKPTDSN PARM is used to change the prefix of the data set names for the output of the individual sorts and intermediate merges. TDS. is the delivered default. The last character of the prefix *must* be a period. If the TIMESTMP option was specified at installation time, up to 21 characters may precede that period. Otherwise, up to 31 characters may be specified before the final period.

DYNATAPE

$\left\{ \begin{array}{l} \text{DYNATAPE} \\ \underline{\text{NODYNATAPE}} \end{array} \right\}$
--

DYNATAPE instructs MAXSORT to dynamically allocate any tapes needed as input for the merge phase. DYNATAPE may be used instead of (or as a supplement to) SORTOUNn DD statements.

NODYNATAPE, the default, disables dynamic allocation.

MAXSORT

MAXSORT

This parameter is *required* in order to execute MAXSORT.

MAXWKSP

$$\text{MAXWKSP} = \left\{ \begin{array}{l} \text{MAX} \\ \text{nM} \\ \text{n} \end{array} \right\}$$

This option specifies the *maximum* amount of disk SORTWK space that MAXSORT can use. When this parameter is used, MAXSORT will release excess space in order to meet the figure specified by the user.

When the RELEASE=ON parameter is active (either via specification or by default) at the conclusion of the sort portion of a MAXSORT, most of the allocated SORTWK space is freed (however, in the case of invoked sorts or SORTWKs defined as OLD, the data set is returned to the size allocated at MAXSORT initiation rather than the minimum size possible).

If MAX, the default value, is specified, all primary and secondary space which has been allocated will be acquired. The MAXWKSP value may also be specified as a decimal number of cylinders (n) or as a decimal number of megabytes (nM) of work space.

If MAXWKSP is specified as n cylinders, MAXSORT will convert the specification to an actual byte value. MAXSORT will multiply by n the capacity of a cylinder on the disk allocated to the lowest-numbered SORTWKnn DD statement.

Note: MAXWKSP should be specified as greater than or equal to MINWKSP, if specified.

MINWKSP

$$\text{MINWKSP} = \left\{ \begin{array}{l} 8 \\ \text{nM} \\ \text{n} \end{array} \right\}$$

This option specifies the *minimum* amount of disk SORTWK space that MAXSORT can use. If the MINWKSP value exceeds the primary allocation and sufficient secondary allocation cannot be obtained to meet the MINWKSP value at the time of execution, the sort terminates. It can be restarted later when more space is available.

The MINWKSP value may be specified as a decimal number of cylinders (n) or a decimal number of megabytes (nM) of work space.

The default MINWKSP value is 8 cylinders.

If MINWKSP is specified as n cylinders, MAXSORT will convert the specification to an actual byte value. MAXSORT will multiply by n the capacity of a cylinder on the disk allocated to the lowest-numbered SORTWKnn DD statement.

Note: MINWKSP should be specified as less than or equal to MAXWKSP, if specified.

RESTART

$\text{RESTART}=\left\{\begin{array}{l} \underline{\text{LAST}} \\ \text{NO} \\ \text{id} \end{array}\right\}$
--

This parameter specifies the point at which restart is to occur.

LAST, the default value, requests that the sort start at the most recent breakpoint.

NO specifies that the SORTBKPT data set is to be cleared so that it can be used for a new job. (Be sure to specify NO only when the SORTBKPT data set is empty or should be destroyed.)

To restart at a particular breakpoint, code its id number. The breakpoint id number is provided by message WER350I.

SORTSIZE

$\text{SORTSIZE}=\left\{\begin{array}{l} \text{n} \\ \text{nM} \\ \text{nT} \end{array}\right\}$
--

This option is accepted but ignored. Its function has been replaced by SyncSort internal techniques.

SORTTIME

$\text{SORTTIME}=\left\{\begin{array}{l} \text{n} \\ \underline{1440} \end{array}\right\}$
--

The SORTTIME parameter terminates the sort at the next breakpoint after n minutes of clock time have elapsed. (The sort may be restarted later.) The default is 1440 minutes (24 hours).

If this parameter is omitted or 1440 is specified, the sort will not terminate prematurely.

This parameter may be specified with operator communication at installation time. If operator communication is specified, the sort will be interrupted at the next breakpoint after the specified amount of time has elapsed and the operator will be asked whether to terminate the sort or continue until the next breakpoint.

TAPENAME

TAPENAME={ name TAPE

This parameter specifies the tape unit generic name for dynamic tape allocation. The default TAPENAME is TAPE.

If the TAPENAME parameter is specified, the same unit generic name must be specified for both the SORTOU00 and the SORTOU n n DD statements.

The tape unit generic name must be a valid unit name at your installation.

Exit Programs

All the exits available for Disk Sort are supported for MAXSORT. However, since MAXSORT never runs out of work space on even the largest sorts, an E16 exit routine will never be called. Exit routines may be written in COBOL, C, Assembler language, or REXX. All exits should be prelink-edited for maximum efficiency.

The following rules must be observed when MAXSORT includes an exit routine:

- Exit programs are not allowed to take their own z/OS checkpoints.
- MAXSORT may take system checkpoints when the following exits are active: E14, E15, E25, E35 and E61. Since a checkpoint may be taken between any two calls of these exits, these routines should be coded accordingly. Any restrictions that apply to system Checkpoint-Restart, such as restrictions on the use of data sets, are applicable to the coding of these exit routines.

Invoking MAXSORT from a Program

MAXSORT can be invoked from programs written in COBOL, PL/1 or Assembler language. However, this is the least efficient method of executing MAXSORT and performance benefits will be realized if MAXSORT is initiated through job control language.

When MAXSORT is invoked from a program, the MAXSORT PARM should be specified in the \$ORTPARM DD statement. The SYSIN DD statement is ignored.

Restarting MAXSORT

A JCL-initiated MAXSORT can be restarted from a breakpoint if necessary. When MAXSORT is restarted from a breakpoint, the following PARM options cannot be modified: CMP=CPD/CLC, EQUALS, E15/E35=COB, FILSZ, LOCALE, MAXSORT, STOPAFT and TAPENAME. Other PARM options will be accepted if they are specified on the EXEC statement. Only the CORE parameter can be passed through \$ORTPARM.

SyncSort control statements cannot be modified when MAXSORT is restarted. However, the l5, l6 and l7 values on the LENGTH parameter of the RECORD control statement can be altered.

Restarting MAXSORT with Exit Routines or an Invoked MAXSORT

When MAXSORT includes an exit routine or is invoked from a program, it cannot be restarted from a breakpoint. Instead, it can be restarted from a checkpoint using the standard OS/VS Checkpoint-Restart feature. Checkpoints are taken at the end of each intermediate sort or merge.

When MAXSORT is restarted from a checkpoint, modified PARM options cannot be specified on the EXEC statement. Only the CORE parameter can be passed through \$ORTPARM.

To specify that checkpoints be taken for a MAXSORT with an exit routine or for an invoked MAXSORT, the following rules must be observed:

- Include the SORTCKPT DD statement in the JCL (in addition to the SORTBKPT DD statement).
- Assign a permanent data set name to every SORTWKnn DD statement and specify DISP=(NEW,DELETE,KEEP).
- Specify RD=R and MSGLEVEL=1 on the JOB statement.
- Specify the CKPT parameter on the SORT/MERGE control statement.

MAXSORT's Operator Interface

If MAXSORT's operator interface options are enabled when SyncSort is installed, they will permit operator communication at selected breakpoints (e.g., at the first breakpoint after SORTTIME has expired, or when tape drives are dynamically allocated under DYNATAPE.) Operator communication allows the operator to examine the environment at execution time to decide whether or not to terminate MAXSORT at that breakpoint. If the operator decides to terminate the sort, it can be restarted later at that breakpoint. All the previously produced sorted data sets can be used without resorting.

Operator communication with MAXSORT is *not* a delivered default - these options must be enabled at SyncSort installation time.

For example, if MAXSORT's assigned block of computer time (its SORTTIME value) has been exhausted and SyncSort was installed to permit operator intervention at such times, message WER375D is generated.

```
WER375D PAYROLL.SORTSTEP - MAXSORT BKPT PAYROLL S12
WER375D TIME ESTIMATE: 30 MINUTES UNTIL NEXT NOTIFICATION
WER375D REPLY 'GO' TO CONTINUE, 'STOP' TO TERMINATE
```

Figure 195. Example: Operator Notification at SORTTIME Expiration

The operator receiving this message can decide to terminate the sort or allow it to continue, basing his decision on scheduling priorities and the estimated time of the sort. When another 30 minutes have passed, the operator will be asked again whether or not MAXSORT should be terminated.

When DYNATAPE is specified and operator communication has been enabled at installation time, message WER376D may be generated to report the results of the dynamic allocation attempt.

```
WER376D PAYROLL.SORTSTEP - MAXSORT BKPT PAYROLL.S01
WER376D 4 TAPE UNITS ALLOCATED TO PAYROLL
WER376D 6 TAPE UNITS NEEDED FOR BEST PERFORMANCE
WER376D TIME ESTIMATE USING 4 TAPE UNITS--
WER376D 20 MINUTES TO NEXT BREAKPOINT
WER376D REPLY 'GO' TO CONTINUE, 'STOP' TO TERMINATE, 'NN' # UNITS
```

Figure 196. Example: Operator Notification with DYNATAPE

If the operator responds 'GO', MAXSORT will execute with four tape units. If the operator responds 'STOP', MAXSORT will terminate. If the operator responds with a number ('NN'), MAXSORT will try to allocate that total number of tape drives. Ideally, the operator should specify six for 'NN' because MAXSORT needs six tape units for best performance. If the operator requests additional tape units, message WER376D will be reissued. The operator will again be prompted for a 'GO', 'STOP' or 'NN' reply. In this way, the operator can balance the requirements of MAXSORT against the requirements of other jobs that are executing at the same time.

When DYNATAPE is specified, there may not be enough tape units available for dynamic allocation. In this case, message WER377D is generated.

```
WER377D PAYROLL.SORTSTEP - MAXSORT BKPT PAYROLL.S01
WER377D INSUFFICIENT TAPE UNITS AVAILABLE
WER377D 2 TAPE UNITS ALLOCATED TO PAYROLL
WER377D 4 TAPE UNITS NEEDED TO CONTINUE EXECUTION
WER377D REPLY 'RETRY' TO GET UNITS, 'STOP' TO TERMINATE
```

Figure 197. Example: Operator Notification of Insufficient Tape Units under DYNATAPE

The operator receiving message WER377D can wait until additional tape drives have been released and then reply 'RETRY'. Or, the operator can answer 'STOP' to terminate the job and then restart it later when more tape drives become available.

If the DYNATAPE and TAPENAME PARMs have been specified and all tape units on the system within the TAPENAME class have already been allocated, message WER378D is generated.

```
WER378D NO ADDITIONAL TAPE UNITS EXIST FOR GENERIC CLASS 2400-3
```

Figure 198. Example: Operator Notification of Insufficient Tape Units for TAPENAME Class

Message WER378D is followed by message WER376D if the number of tape drives allocated is sufficient for execution, or by message WER377D if it is not sufficient.

Sample MAXSORT JCL/Control Streams

The following examples illustrate how the JCL could be coded for typical 100-megabyte MAXSORTs.

Example 1: A 1-Gigabyte MAXSORT with only Minimal Disk Space Available

An installation is running a 1-gigabyte sort and has a restricted amount of disk space available for SORTWK across three volumes (WORK1, WORK2 and WORK3).

The JCL for this job follows.

```
//ALLOC      EXEC  PGM=IEFBR14                      1
//BKPTDATA   DD    DSN=BKPT.DATA,DISP=(NEW,CATLG),   2
//           UNIT=SYSDA,SPACE=(1000,(100,50))
//SORT       EXEC  PGM=SYNCSORT,PARM='MAXSORT,MINWKSP=600' 3
//SORTBKPT   DD    DSN=BKPT.DATA,DISP=(OLD,KEEP)     4
//SYSOUT     DD    ...                               5
//SORTIN     DD    ...
//SORTOUT    DD    ...
//SORTWK01   DD    UNIT=SYSDA,SPACE=(CYL,(150,20)),   6
//           VOL=SER=WORK1
//SORTWK02   DD    UNIT=SYSDA,SPACE=(CYL,(150,20)),
//           VOL=SER=WORK2
//SORTWK03   DD    UNIT=SYSDA,SPACE=(CYL,(150,20)),
//           VOL=SER=WORK3
//SORTOU00   DD    DSN=PERM.OU00,DISP=(NEW,KEEP),     7
//           UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SORTOU01   DD    DSN=PERM.OU01,DISP=(NEW,KEEP),     8
//           UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SORTOU02   DD    DSN=PERM.OU02,DISP=(NEW,KEEP),
//           UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SORTOU03   DD    DSN=PERM.OU03,DISP=(NEW,KEEP),
//           UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SYSIN      DD    *                                9
              SORT FIELDS=(1,10,CH,A)
/*
```

Figure 199. Sample JCL Control Stream for a 1-Gigabyte MAXSORT

1. This job step is run in order to allocate the disk space for the breakpoint data set via IBM utility program IEFBR14.
2. This statement allocates the space for the breakpoint data set. Specify (NEW,CATLG) because this data set must be saved.

3. The EXEC statement initiates the regular SyncSort program, and the MAXSORT PARM is specified, as required. This job requests a minimum of 600 cylinders of disk space for SORTWKnn data sets. If that much space cannot be obtained during the job, the program will terminate.
4. The SORTBKPT DD statement is required for all MAXSORTs. It identifies the breakpoint data set which was allocated in the first job step. DISP=(OLD,KEEP) is specified so that this statement can be reused if MAXSORT is restarted.
5. These DD statements are coded just as they would be for an ordinary sort.
6. The SORTWKnn DD statements must be allocated to disk or MAXSORT will terminate. In this case, 450 cylinders of primary space have been allocated. Secondary allocation could provide up to 300 cylinders on each volume if that amount of free space exists. Since the MINWKSP PARM specifies at least 600 cylinders, this program will terminate unless 150 cylinders of secondary space can be obtained.
7. The SORTOU00 DD statement is required for this job because the intermediate sort output will be stored on tape. DISP=(NEW,KEEP), a permanent DSN and VOL=PRIVATE are specified to ensure that the system unloads each output tape. The DEFER option in the UNIT parameter is specified so that mount messages to the operator that do not pertain to MAXSORT are suppressed.
8. The SORTOU01, SORTOU02 and SORTOU03 DD statements allocate the tape units used as input to the merge phase. Permanent DSNAMES, DISP=(NEW,KEEP), VOL=PRIVATE and the DEFER option in the UNIT parameter are all specified just as they were for the SORTOU00 DD statement.
9. The sort control statements are included here.

Example 2: Restarting the MAXSORT in Example 1 from a Breakpoint

Example 1 can be restarted from a breakpoint simply by submitting the original job control stream *without* the job step which allocated space for the breakpoint data set. The job will be restarted from the last breakpoint because RESTART=LAST is the default; it is not necessary to specify RESTART=LAST on the EXEC statement.

The JCL for this job follows.

```
//SORT      EXEC  PGM=SYNCSORT, PARM='MAXSORT, MINWKSP=600 '  
//SORTBKPT  DD   DSN=BKPT.DATA, DISP=(OLD, KEEP)  
//SYSOUT    DD   ...  
//SORTIN    DD   ...  
//SORTOUT   DD   ...  
//SORTWK01  DD   UNIT=SYSDA, SPACE=(CYL, (150, 20)), VOL=SER=WORK1  
//SORTWK02  DD   UNIT=SYSDA, SPACE=(CYL, (150, 20)), VOL=SER=WORK2  
//SORTWK03  DD   UNIT=SYSDA, SPACE=(CYL, (150, 20)), VOL=SER=WORK3  
//SORTOU00  DD   DSN=PERM.OU00, DISP=(NEW, KEEP),  
//          UNIT=(TAPE, , DEFER), VOL=PRIVATE  
//SORTOU01  DD   DSN=PERM.OU01, DISP=(NEW, KEEP),  
//          UNIT=(TAPE, , DEFER), VOL=PRIVATE  
//SORTOU02  DD   DSN=PERM.OU02, DISP=(NEW, KEEP),  
//          UNIT=(TAPE, , DEFER), VOL=PRIVATE  
//SORTOU03  DD   DSN=PERM.OU03, DISP=(NEW, KEEP),  
//          UNIT=(TAPE, , DEFER), VOL=PRIVATE  
//SYSIN     DD   *  
             SORT FIELDS=(1, 10, CH, A)  
/*
```

Figure 200. Sample JCL Control Stream for Restarting a 1-Gigabyte MAXSORT

The JCL is identical to the JCL in Example 1 except that the step which allocated the disk space for the breakpoint data set is not resubmitted.

Example 3: A 1-Gigabyte MAXSORT with Dynamic Tape Allocation

This example is identical to Example 1 with one difference: the DYNATAPE PARM requests dynamic tape allocation.

The JCL for this job follows.

```
//ALLOC      EXEC  PGM=IEFBR14
//BKPTDATA   DD    DSN=BKPT.DATA,DISP=(NEW,CATLG),UNIT=SYSDA,
//           SPACE=(1000,(100,50))
//SORT       EXEC  PGM=SYNCSORT,PARM='MAXSORT,MINWKSP=600,
//           DYNATAPE'
//SORTBKPT   DD    DSN=BKPT.DATA,DISP=(OLD,KEEP)
//SYSOUT     DD    ...
//SORTIN     DD    ...
//SORTOUT    DD    ...
//SORTWK01   DD    UNIT=SYSDA,SPACE=(CYL,(150,20)),VOL=SER=WORK1
//SORTWK02   DD    UNIT=SYSDA,SPACE=(CYL,(150,20)),VOL=SER=WORK2
//SORTWK03   DD    UNIT=SYSDA,SPACE=(CYL,(150,20)),VOL=SER=WORK3
//SORTOU00   DD    DSN=PERM.OU00,DISP=(NEW,KEEP),
//           UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SORTOU01   DD    DSN=PERM.OU01,DISP=(NEW,KEEP),
//           UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SORTOU02   DD    DSN=PERM.OU02,DISP=(NEW,KEEP),
//           UNIT=(TAPE,,DEFER),VOL=PRIVATE
//SYSIN      DD    *
              SORT FIELDS=(1,10,CH,A)
/*
```

Figure 201. Sample JCL Control Stream for a 1-Gigabyte MAXSORT

The DYNATAPE PARM requests that tape units be obtained dynamically. Because DYNATAPE has been specified, the SORTOU01, SORTOU02 and SORTOU03 DD statements specified in Example 1 do not have to be supplied. They will be created and dynamically allocated when needed. If enough tape units are available at the time the job is run, the sort will be successfully completed in one step.

However, there may not be enough tape devices available under dynamic allocation at execution time. In that case, the job will terminate and can be restarted at a later time when more tape units are available.

For best results, code two SORTOU_n DD statements in addition to specifying the DYNATAPE PARM as the above example illustrates. This approach ensures that MAXSORT will have the minimum two tape units needed for the merge phase *and* also allows MAXSORT to take advantage of the additional tapes available under dynamic allocation.

Tuning MAXSORT

MAXSORT's performance can be optimized by controlling the intermediate sorts which it processes. A balance should be achieved between the number and duration of intermediate sorts. Limiting the number of sorts reduces the required tape mounts and restricting the duration of sorts decreases the interval between breakpoints.

A good rule of thumb is that each intermediate sorted data set should create from one to five volumes of input data, and the only way to determine the amount of input data is by controlling the amount of SORTWK space used. This is illustrated in Figure 202.

```
1 tape volume contains 250 megabytes
1 3390 cylinder can hold approximately 800,000 bytes
SORTIN: 750 megabytes (3 tape volumes)
OBJECTIVE: Each intermediate sort processes one input volume,
           i.e. 3 intermediate sorts should be run.

To determine SORTWK allocation (for 3390 SORTWKs):
  Divide one input volume  250,000,000
  by cylinder capacity     800,000

Quotient:   313 cylinders.

To ensure 3 intermediate sorts:

  Allocate 313 cylinders
  Set MAXWKSP=250M
```

Figure 202. Calculating MAXWKSP

If only 50 cylinders were allocated in the preceding example, 19 intermediate sorts would be performed, increasing the required tape mounts and potential for error. If 800 cylinders were allocated, most of the input would be processed by the first intermediate sort, delaying the first breakpoint and introducing the potential for losing data. It is crucial, therefore, to allocate a balanced amount of DASD space that will divide your file into reasonably sized segments to minimize the possibility of system error and to enhance your performance.

Before tuning MAXSORT then, a number of individual environmental elements should be considered. A study of disk and tape availability, input data file size, and virtual storage limitations will help you optimize the balance of performance and reliability.

Chapter 10. PARASORT

PARASORT: Parallel Input Processing for Elapsed Time Improvement

PARASORT improves elapsed time performance for sorts whose input is a multi-volume tape data set and/or concatenated tape data sets. Reduced elapsed time can help critical sort applications achieve batch window goals.

The performance improvement from PARASORT is a result of processing the SORTIN input volumes in a parallel fashion. Depending upon the resources provided, elapsed time can be reduced up to 20% for 2-way input and up to 33% for 4-way input.

PARASORT requires additional tape units for the application. You will need from two to eight times the current number of tape units, depending upon resource availability and the degree of improvement desired. PARASORT automatically manages the tape units and minimizes the use of the tape drive resources by deallocating excess tape drives during initialization and releasing all the extra units at the end of the sort input phase.

PARASORT Applicability

Certain SyncSort facilities or application characteristics cannot be used with PARASORT. The following are incompatible with a PARASORT application:

- A SORTIN record format (RECFM) of VS or VBS.
- An ASCII tape data set specified for SORTIN.

- An exit routine other than a pre-linked or inline E35 exit.
- EQUALS specified either as an installation or run-time option.
- SKIPREC or STOPAFT options specified.
- SEQNUM specified on INREC.
- CKPT (checkpoint) option in effect.
- The MAXSORT option specified.
- The OUTREC CONVERT feature to convert VL records to FL format.
- Certain unusual sort key types, feature combinations, or long sort keys in excess of 800 bytes.
- FIELDS=COPY specified.

Job Control Language

The JCL for PARASORT is similar to the JCL for a standard disk sort. The primary difference is that PARASORT JCL must specify additional tape units to allow parallel input processing of the SORTIN data set. For details on SORTIN JCL for PARASORT, see “SORTIN DD Statement with PARASORT” on page 10.3.

To initiate PARASORT using job control statements, specify PARM='PARASORT' on the EXEC statement. A program initiated sort requests PARASORT by using a PARM card image in the data set defined by the \$ORTPARM DD statement.

Sample EXEC Statement

<pre>//stepname EXEC</pre>	<pre>{ PGM=SYNCSORT PGM=SORT PGM=IERRCO00 PGM=IGHRCO00 PGM=ICEMAN }</pre>	<pre>,PARM='PARASORT'</pre>
----------------------------	---	-----------------------------

Figure 203. PARASORT EXEC Statement

DD Statements

PARASORT's DD statement requirements are summarized in the following table. One additional DD type (SORTPARn) is required compared to a conventional disk sort.

PARASORT DD Statements		
//SORTPARM	DD	Used to override PARM or control statement information.
//SYSIN	DD	Control statement data set. Required unless the address of a 24-bit or 31-bit extended parameter list is supplied by an invoking program.
//SYSOUT	DD	Message data set. Required unless all messages are routed to console.
//SORTWK _{xxx}	DD	Disk work area definition. Required unless DYNALLOC is specified.
//SORTIN	DD	SORT input data set. Required.
//SORTPAR _n	DD	Defines additional tape units for parallel reading of SORTIN. Required.
//SORTOUT	DD	Output data set. Required unless there is an E35. Ignored if the invoking program supplies an inline E35 exit routine; optional if the MODS statement activates an E35 exit routine.
//SORTXSUM	DD	Output data set for records deleted by SUM. Required if XSUM parameter used.
//ddname	DD	Required unless E35 user exit is in LINKLIB/JOBLIB/STEPLIB.

Table 40. PARASORT DD Statements

The SORTIN and SORTPAR_n DD statements are discussed below. For a discussion of the other DD statements, which are specified for PARASORT just as for a non-PARASORT Disk Sort, see “Chapter 4. JCL and Sample JCL/Control Statement Streams”.

SORTIN DD Statement with PARASORT

The SORTIN DD statement is required for a PARASORT application. It must define either a single multi-volume tape data set or several concatenated tape data sets, which can be single or multi-volume. If the SORTIN is concatenated, each data set must meet the normal SORTIN concatenation requirements and must be able to use the same device type so that UNIT=AFF=SORTIN can be specified for all data sets in the concatenation. For a discussion of normal SORTIN JCL, see “Chapter 4. JCL and Sample JCL/Control Statement Streams”

For optimal performance, data sets that reside on tapes, such as 3480s, that can be read only in a single direction should have two units allocated. If the data set is on a tape that supports bidirectional processing, a single unit is sufficient. In all cases DEFER mounting must be specified.

SORTIN data sets may not be passed data sets or have PASS specified on their DD statement.

Either the catalog or specific list of volume serial numbers must be specified. The volume serial list must accurately reflect the volumes in the data set. If extra volumes are specified (as may happen if an old data set is rewritten with less data) an error message will be generated. A volume sequence number may not be specified.

The following example SORTIN DD statements for PARASORT illustrate three different SORTIN cases:

- A multi-volume cataloged data set
- A multi-volume uncataloged data set
- Concatenated single and multi-volume uncataloged data sets

Note that each example includes a *y* on the UNIT specification (for example, UNIT=(3480,*y*,DEFER)). The *y* is either 1 or 2 and indicates the number of units to be allocated for these devices. For optimal performance, data sets that reside on tapes that can be read only in a single direction, such as 3480s, should have two units allocated. If the data set is on a tape that supports bidirectional processing, a single unit is sufficient. In all cases DEFER mounting must be specified.

Note also that each example includes an alternative UNIT specification: UNIT=(xxxxx1,*y*,DEFER). This specification applies if special esoteric names are available. The xxxxx1 is a special esoteric unit name established especially for PARASORT. These esoteric names may have been created at your site for use with PARASORT. Contact the systems programmer responsible for SyncSort installation to determine if they are available. For information on how to select a special esoteric name, see “Special Channel Separated Esoteric Names” on page 10.7

Example 1

SORTIN consists of a single multi-volume cataloged data set.

```
//SORTIN DD DSN=INPUT.FILE,DISP=(OLD,KEEP),  
// UNIT=(,y,DEFER)  
// or if a special esoteric name is available  
// UNIT=(xxxxx1,y,DEFER)
```

Figure 204. Sample SORTIN DD Statement

Example 2

SORTIN consists of a single multi-volume uncataloged data set.

```
//SORTIN DD DSN=INPUT.FILE,DISP=(OLD,KEEP),
//   UNIT=(3480,y,DEFER),VOL=SER=(VOL001,VOL002,...,VOL00N)
//   or if special esoteric name is available
//   UNIT=(xxxxx1,y,DEFER),VOL=SER=(VOL001,VOL002,...,VOL00N)
```

Figure 205. Sample SORTIN DD Statement

This example presumes the file is standard label and the first file is on VOL001.

For uncataloged data sets, the unit and volser list must be specified.

Example 3

SORTIN consists of a concatenation of single and multi-volume uncataloged data sets.

```
//SORTIN DD DSN=INPUT.FILE1,DISP=(OLD,KEEP),
//   UNIT=(3480,y,DEFER),VOL=SER=(VOL001,VOL002,VOL003)
//   or if special esoteric names are available
//   UNIT=(xxxxx1,y,DEFER),VOL=SER=(VOL001,VOL002,VOL003)
//   DD DSN=INPUT.FILE2,DISP=(OLD,KEEP),
//   UNIT=AFF=SORIN,VOL=SER=(VOL101,VOL102)
//   DD DSN=INPUT.FILE3,DISP=(OLD,KEEP),
//   UNIT=AFF=SORIN,VOL=SER=(VOL201)
```

Figure 206. Sample SORTIN DD Statement

It is also possible to include a DD DUMMY allocation in the SORTIN concatenation. This is necessary when modifying production JCL that is a prototype for the maximum number of possible concatenated input files. If a particular execution uses less than the maximum number, place the DD DUMMY specification at the appropriate point. This specification should contain a DCB specification that matches that of the first SORTIN data set.

SORTPARn DD Statements

The SORTPARn DD statements define units that will be used to perform the parallel reading of the input file. Up to four SORTPARn DD statements may be provided, with a minimum of two required. The number of SORTPARn DD statements that you provide may be limited by the tape channel capacity at your installation. See “Special Channel Separated Esoteric Names” on page 10.7 for information on how to determine if your choice is limited.

The n in the SORTPARn is replaced with numbers 1 through 4. The numbers must start at 1 and be numbered consecutively.

The required SORTPAR1 must be coded in one of the following two ways, depending on whether the SORTIN data set is cataloged or not.

- If the SORTIN DD is defined as a single **cataloged** data set or as a series of concatenated data sets where the first data set of the concatenation is cataloged, then the SORTPAR1 DD must be coded as follows:

```
//SORTPAR1 DD DSN=*.SORTIN,DISP=OLD,  
//          UNIT=AFF=SORTIN
```

Figure 207. Sample SORTPAR1 DD Statement

- If the SORTIN DD is defined as a single **non-cataloged** data set or as a series of concatenated data sets where the first data set of the concatenation is non-cataloged, then the SORTPAR1 DD must be coded as follows:

```
//SORTPAR1 DD DSN=*.SORTIN,DISP=OLD,UNIT=AFF=SORTIN,  
//          VOL=SER=(VOL001,VOL002,...,VOL00n)
```

Figure 208. Sample SORTPAR1 DD Statement

where the VOL=SER list contains the identical volumes specified on the SORTIN DD specification. If the SORTIN DD is a series of concatenations, the VOL=SER list is the volumes that comprise the first data set in the concatenation.

The remaining SORTPARnn DDs are coded as shown on the following prototype SORTPARnn DD statement:

```
//SORTPARn DD DSN=*.SORTIN,DISP=(,KEEP,KEEP),  
//          VOL=PRIVATE,UNIT=(xxxx,y,DEFER)  
//          or if special esoteric names are available  
//          VOL=PRIVATE,UNIT=(xxxxxn,y,DEFER)
```

Figure 209. Prototype SORTPARn DD Statement

The xxxx is a unit type or generic name compatible with the device associated with SORTIN. If special channel separated esoteric names have been made available, see “Special Channel Separated Esoteric Names” on page 10.7.

The y is either 1 or 2 and indicates the number of units to be allocated for these devices. For optimal performance, data sets that reside on tapes that can be read only in a single direction, such as 3480s, should have two units allocated. If the data set is on a tape that supports bidirectional processing, a single unit is sufficient. In all cases DEFER mounting must be specified.

The number of SORTPARn data sets to allocate depends on several factors:

- The total number of volumes to be read from SORTIN and its concatenations.
There is no need to allocate more SORTPARn data sets than total volumes in the SORTIN file. Note that if more SORTPARn data sets are allocated than there are volumes, the excess SORTPARn data sets will be deallocated at PARASORT's initiation.
- The degree of performance improvement desired.
Typically, two SORTPARn datasets will provide up to 20% elapsed time improvement; three, up to 25%; and four, up to 33%.
- The degree of channel contention, which may reduce the number of SORTPARn DD statements used.
The use of special esoteric unit names will ensure that this contention is eliminated, but your choice for the number of SORTPARn DD statements may be limited.
- Resource availability.
System constraints may limit the number available to a particular job.

Special Channel Separated Esoteric Names

For optimal PARASORT performance, SyncSort must be able to read each SORTPARn input DD simultaneously, with no channel contention. To ensure this, your system programming staff may have defined special esoteric unit names for use with PARASORT. Before creating a PARASORT application, you should contact your system programmer to verify that this work has been done and that the special names are available for use. Note, however, that even if the work has been done, certain categories may be unavailable due to limited channel capacity.

To use special esoteric names, do the following:

1. Decide whether you would like to specify 4-way (up to SORTPAR4), 3-way (up to SORTPAR3) or 2-way (up to SORTPAR2) input.
2. In the table of special esoteric names provided by your systems programmer, find the name that corresponds to the tape type of the SORTIN data sets.

The following is a sample table of special esoteric names. This table is for illustration only; the names at your site may be different.

	3420	3480/90	3490E	3590
4-WAY	PAR241	PAR441	PARE41	NOT
INPUT	PAR242	PAR442	PARE42	POSSIBLE
	PAR243	PAR443	PARE43	
	PAR244	PAR444	PARE44	
3-WAY	PAR231	PAR431	PARE31	NOT
INPUT	PAR232	PAR432	PARE32	POSSIBLE
	PAR233	PAR433	PARE33	
2-WAY	PAR221	PAR421	PARE21	PAR921
INPUT	PAR222	PAR422	PARE22	PAR922

Figure 210. Sample Esoteric Unit Name Table

- Use the name from the table at your site for your SORTIN and SORTPARn names. The following example JCL is for input from 3480 cartridges with at least 4 volumes:

```
//SORTIN DD DSN= . . . , DISP=(OLD,KEEP) , UNIT=(PAR441,2,DEFER)
//SORTPAR1 DD DSN=* .SORTIN, DISP=OLD,
// UNIT=AFF=SORTIN
//SORTPAR2 DD DSN=* .SORTIN, DISP=(,KEEP,KEEP) ,
// VOL=PRIVATE, UNIT=(PAR442,2,DEFER)
//SORTPAR3 DD DSN=* .SORTIN, DISP=(,KEEP,KEEP) ,
// VOL=PRIVATE, UNIT=(PAR443,2,DEFER)
//SORTPAR4 DD DSN=* .SORTIN, DISP=(,KEEP,KEEP) ,
// VOL=PRIVATE, UNIT=(PAR444,2,DEFER)
```

Sortwork Considerations

The amount of sortwork space required is the same as if the application were run as a conventional sort. What should be modified, if sortworks are provided via JCL rather than DYNALLOC, is the number of SORTWKxx DD statements. Try to provide a total number of SORTWKxx DDs that is two to three times the number of SORTPARns specified. This would typically require an adjustment in primary and secondary space amounts so that the total space allocated is similar to that of the original application. This subdivision of SORTWORK space will provide an opportunity for additional channel path availability. This parallelism in SORTWORK channel paths is also a key to improving sort elapsed time performance.

Operations Notes

Since the SORTIN tape volumes will be read in any order and on a SORTPAR_n tape unit location determined by the PARASORT logic, it is possible to receive messages on the console log that indicate out of sequence processing of the volumes of the SORTIN data set. Messages such as the following may be generated:

IEC712I ... SORTPAR_n READ - NOT FIRST VOLUME OF DATA SET

or

IEC710I ... SORTPAR_n ANOTHER VOLUME EXPECTED

These messages can be disregarded since this type of processing is deliberate with PARASORT.

Chapter 11. SyncSort DB2 Query Support

SyncSort can directly retrieve data from a DB2 database based on a user-provided query. An SQL SELECT statement is used to specify the criteria of the request. The query of the DB2 database replaces SyncSort's SORTIN or E15 processing. SORT or COPY functions, but not MERGE, can be used with DB2 queries. All SyncSort features performed after E15 processing are available for use with the DB2 query facility. Refer to “Chapter 8. The Flow of the Sort” for a summary of SyncSort's features and flow of control during processing.

The SyncSort DB2 Query facility improves performance over DB2's DSNTIAUL program by allowing DB2 data to be passed directly into a SORT or COPY operation, without the use of setup steps or the need for user-written E15 exits.

Restrictions

The following cannot be used with the DB2 Query facility. If specified, they will cause SyncSort to terminate with a return code of 16:

- E15 exit
- The SKIPREC parameter
- The MAXSORT feature
- The PARASORT feature
- MERGE

The following will be ignored if used with the DB2 Query facility:

- A SORTIN data set
- The TYPE parameter and the L1 and L2 values of the LENGTH parameter of a RECORD statement

Job Control Language

The JCL for the DB2 Query facility is similar to the JCL of a standard disk sort. The primary difference is that the DB2 query JCL must also contain an additional SORTDBIN DD specification to define the DB2 query with an SQL SELECT statement.

To initiate a SORT or COPY with the DB2 Query facility using job control statements, specify PARM='DB2=dsn' on the EXEC statement. The dsn referred to in the DB2 parameter is the DB2 subsystem name to be accessed. When a SORT or COPY DB2 Query application is invoked from a program, specify the DB2 parameter in the \$ORTPARM DD statement.

Note: In order to issue the first query to the DB2 subsystem identified in the DB2=parm, you must have BINDADD authority so the plan can be added to the subsystem.

Sample EXEC Statement

The following shows a sample EXEC statement with the DB2 PARM:

<pre>//stepname EXEC { PGM=SYNCSORT PGM=SORT PGM=IERRCO00 PGM=IGHRCO00 PGM=ICEMAN }</pre>	<pre>,PARM='DB2=dsn'</pre>
---	----------------------------

Figure 211. DB2 Query EXEC Statement

DD Statements

The DD statements used with the DB2 Query facility are summarized in the following table. Note that the SORTDBIN DD statement is unique to the DB2 Query facility.

DB2 Query DD Statements		
//\$ORTPARM	DD	Used to override PARM or control statement information.
//SYSIN	DD	Control statement data set. Required unless the address of a 24-bit or 31-bit extended parameter list is supplied by an invoking program.

DB2 Query DD Statements		
//SYSOUT	DD	Message data set. Required unless all messages are routed to console.
//SORTWK _{xxx}	DD	Disk work area definition. Required unless DYNALLOC is specified.
//SORTDBIN	DD	Data set which defines the DB2 query. Contains the SQL SELECT statement to be used for this application.
//SORTOUT	DD	Output data set. Required unless there is an E35. Ignored if the invoking program supplies an inline E35 exit routine; optional if the MODS statement activates an E35 exit routine.
//SORTXSUM	DD	Output data set for records deleted by SUM. Required if XSUM parameter used.
//SORTMODS	DD	Required if user exits are in SYSIN and if user exits are to be linkage-edited at execution time.
//SYSLIN	DD	
//SYSLMOD	DD	
//SYSUT1	DD	
//SYSPRINT	DD	
//ddname	DD	Required for exits unless the exit is inline in LINKLIB/JOBLIB/STEPLIB or in SYSIN.

Table 41. DB2 Query DD Statements

SORTDBIN DD Statement

The SORTDBIN DD statement is required for a DB2 query application. The data set defined by the SORTDBIN contains the SQL SELECT statement that describes the criteria of the query.

The SORTDBIN DD record format must be F or FB, and the record length must be 80.

The SORTDBIN data set must be formatted in accordance with the following rules:

- Only a SELECT statement or \$ELECT statement (for trial run described below) is accepted. Any other SQL statements will cause the job to terminate with a WER468A error message. For details on the facilities and syntax of a SELECT statement refer to the IBM publication *DB2 Universal Database for OS/390 SQL Reference (GC26-9014)*.
- The maximum supported length of a SELECT statement for this feature is 32765 characters.
- The SELECT statement may not use the '--' convention of two consecutive hyphens to denote that the remainder of a card image is a comment.
- The SELECT statement may be terminated with a semicolon. Any characters found after the semicolon will be considered comments.

- Only columns 1 through 72 of each record will be read. Columns 73 through 80 will be ignored.
- Long fields (LONG VARCHAR and LONG VARGRAPHIC) and large object fields (BLOB, CLOB, and DBCLOB) are not supported. If they are specified, the application will terminate with a WER468A error message.

Operation

Using the query provided in the SORTDBIN data set, SyncSort will access the DB2 database specified in the DB2 EXEC PARM and will process as fixed-length records the rows returned from the query. The records will be processed as if read from a SORTIN or retrieved from an E15 exit. All SyncSort features available after E15 processing in the flow of control can be used with a SORT or COPY application.

The record used within SyncSort is constructed from the fields in the query as follows:

- The order of the fields in the record is the same as the order specified by the SELECT statement.
- The data format of the fields within the record is the same format returned by DB2.
- A fixed-length field is the same length as returned by DB2.
- Variable-length character data is stored in a fixed-length field. The field length is equal to the maximum length of the field plus two bytes for a leading field length descriptor variable. The field length descriptor contains a binary value describing the number of bytes of data provided for this field. If an instance of the field is shorter than the maximum, the remaining bytes will be set to binary zeros.
- Any fields defined to allow nulls will cause the creation of two fields within the record constructed by SyncSort. The first will be the data field and the second will be a one byte indicator field. If the value of the field is null, by default the field will be filled with binary zeros (X'00') and the indicator field will contain a '?' to signify the field is null. If the value of the field is not null, the indicator will be set to binary zeros. If binary zeros would not be an appropriate fill value for a null field, use of SQL functions such as VALUE or COALESCE on the SELECT statement should be considered. For instance, if the field to be retrieved is packed decimal, it is usually best to create a null value of the proper PD format. This ensures that if the field is used later as a sort key or in other data conversion features, it would contain appropriate high or low values such as PD zeros or nines as specified in the VALUE or COALESCE function.

Record Description

Information about the record created by the query will be displayed in the SyncSort message data set. For each column selected, the report will display the start and end position

within the record, the DB2 data type, the equivalent SyncSort data type, and whether null values are allowed. A data type whose length is not implied by the format will have a field length appended to its description. Note that the length displayed for a VARCHAR DB2 data type is two bytes shorter than would be indicated by the field start and end positions. The extra two bytes described in the start and end positions are for the field length descriptor, which is contained in the first two bytes of the field.

Record Description: Trial Mode Execution

When first developing an application, knowledge of the actual input record layout built from the query is required. This can be obtained from a trial mode execution. The trial mode execution uses a query provided in the SORTDBIN data set to generate a report of the input record layout in the SyncSort message data set (SYSOUT). The trial mode execution does not perform any other processing or request data from DB2. To request trial mode execution, modify the SQL SELECT keyword in the SORTDBIN data set to \$SELECT. This indicates a trial mode execution is to be performed. For trial mode, execute SyncSort with JCL of the following form:

```
//          EXEC PGM=SYNCSORT, PARM= ' DB2=DSN1 '
//STEPLIB DD   DSN=DB2 .SDSNLOAD, DISP=SHR
//          DD   DSN=SORT .RESI .DENCE, DISP=SHR
//SYSOUT   DD   SYSOUT=A
//SORTDBIN DD *
$SELECT FIRSTNME, LASTNAME, WORKDEPT, HIREDATE, EDLEVEL, SALARY
  FROM DSN.EMPTAB
  WHERE EDLEVEL>10
/*
```

Figure 212. Sample JCL for Trial Mode Execution

Note that only the SYSOUT and SORTDBIN DD are required for a trial mode execution. An actual execution of the application will require other DDs as documented for SORT or COPY applications.

The STEPLIB DD statement specifies where the SyncSort and DB2 products can be found. The STEPLIB DD statement would be needed if these products could not be found in the standard system libraries.

The DB2 EXEC statement parameter would be set to the DB2 subsystem name to be accessed.

The SYSOUT data set will contain an input record layout report as shown in the following sample:

```

SYNCSORT FOR Z/OS 1.1AN TPF0 U.S. PATENTS: 4210961, 5117495 (C) 2001 SYNCSORT INC. DATE=2001/225 TIME=14.40.32
                                Z/OS 1.1.0 CPU MODEL 2064
PRODUCT LICENSED FOR CPU SERIAL NUMBER 12345 LICENSE/PRODUCT EXPIRATION DATE: 31 MAR 2002
DB2 QUERY OPTION SELECTED
QUERY STATEMENTS:
SELECT FIRSTNME, LASTNAME, WORKDEPT, HIREDATE, EDLEVEL, SALARY
FROM DSN.EMPTAB
WHERE EDLEVEL>10
INPUT RECORD DESCRIPTION:

COLUMN          START   END     DB2          SYNCSORT      NULL VALUE
NAME            POSITION POSITION DATA TYPE    DATA TYPE
-----
FIRSTNME              1      14   VARCHAR(12)  CH            DISALLOWED
LASTNAME             15      31   VARCHAR(15)  CH            DISALLOWED
WORKDEPT             32      34   CHAR(3)      CH            ALLOWED
NULLINDICATOR        35      35   DATE         CH            ALLOWED
HIREDATE              36      45   DATE         CH            ALLOWED
NULLINDICATOR        46      46   SMALLINT     CH            ALLOWED
EDLEVEL              47      48   SMALLINT     BI            ALLOWED
NULLINDICATOR        49      49   DECIMAL(9,2) CH            ALLOWED
SALARY               50      54   DECIMAL(9,2) PD            ALLOWED
NULLINDICATOR        55      55   DECIMAL(9,2) CH

```

```

WER467I DB2 QUERY TRIAL MODE SUCCESSFULLY EXECUTED
WER169I RELEASE 1.1A BATCH 0999 TPF LEVEL 0
WER052I END SYNCSORT - JOBNAME, SORTSQL, , DIAG=E4FF, E28C, C888, 6044, AC66, 48A3, 0E88, 6E64

```

Figure 213. Sample SYSOUT

You would create SyncSort control statements with field specifications based on the input record layout and place the control statements in the data set specified by the SYSIN DD statement. You would then create a set of JCL statements for the application.

Sample SyncSort DB2 Query Application

In this example, a query is made for employee data. The example specifies how the application is to sort and format the data into a report. The formatting adds headers, field spacing, and converts a date to printable forms.

```
//SORTSQL EXEC PGM=SYNCSORT, PARM='DB2=DSN1 '  
//STEPLIB DD DSN=DB2.SDSNLOAD, DISP=SHR  
// DD DSN=SORT.RESI.DENCE, DISP=SHR  
//SYSOUT DD SYSOUT=A  
//SORTOF1 DD DSN=OUT1, DISP=(NEW, CATLG), UNIT=3390, SPACE=(CYL, 1)  
//SORTWK01 DD UNIT=SYSDA, SPACE=(CYL, 20)  
//SORTWK02 DD UNIT=SYSDA, SPACE=(CYL, 20)  
//SORTDBIN DD *  
SELECT FIRSTNME, LASTNAME, WORKDEPT, HIREDATE, EDLEVEL, SALARY  
FROM DSN.EMPTAB  
WHERE EDLEVEL>10  
//SYSIN DD *  
SORT FIELDS=(3, 12, CH, A)  
OUTFIL FILES=1,  
HEADER1=(2/, 20:'EMPLOYEE INFORMATION',  
2/, 1:'FIRSTNAME', 14:'LASTNAME', 29:'WORKDEPT',  
41:'HIRE DATE', 54:'LEVEL', 65:'SALARY',  
/, 1:'-----', 14:'-----', 29:'-----',  
40:'-----', 54:'-----', 65:'-----'),  
OUTREC=(1:3, 12, C' ', 17, 15, C' ', 32, 3, 7C' ',  
36, 10, C' ', 47, 2, BI, M0, 5C' ', 50, 5, PD, M2)
```

Figure 214. Sample SyncSort DB2 Query Application

The following describes the JCL statements:

- The EXEC statement identifies SYNCSORT as the program to be executed. The DB2 PARM defines the DB2 subsystem to be accessed.
- The STEPLIB DD statement instructs the system as to where the SyncSort and DB2 products can be found.
- The SYSOUT DD statement assigns the SyncSort messages to the output device associated with SYSOUT class A.
- The SORTOF1 DD statement gives OUT1 as the output data set name and specifies a 3390 disk. One cylinder of primary space has been allocated on this volume. The DISP parameter shows that this data set is not yet in existence.
- The two SORTWK statements reserve space on four temporary data sets for intermediate storage. Twenty cylinders are to be reserved on the data sets.

- The SORTDBIN DD statement marks the beginning of the input stream that contains the SQL SELECT statement that describes the criteria of the query.
- The SYSIN DD statement marks the beginning of the input stream that includes the sort control statements. A sort will be performed and a report will be generated. The records read from the DB2 database under control of the query specified in the SORTDBIN data set will be formatted and presented in this report. Fields will be converted to printable format when necessary.

The SYSOUT will contain a report on the execution of the application. The report displays the control statements followed by the query record layout and SyncSort messages with information on the particular execution. The following is a sample report:

SYNCSORT FOR Z/OS 1.1AN TPF0 U.S. PATENTS: 4210961, 5117495 (C) 2001 SYNCSORT INC. DATE=2001/225 TIME=14.40.32

Z/OS 1.1.0 CPU MODEL 2064

PRODUCT LICENSED FOR CPU SERIAL NUMBER 12345

LICENSE/PRODUCT EXPIRATION DATE: 31 MAR 2002

SYSIN :

```

SORT FIELDS=(3,12,CH,A)                                00048000
OUTFIL FILES=1,                                        00049002
HEADER1=(2/,20:'EMPLOYEE INFOMATION',                 00050002
2/,1:'FIRSTNAME',14:'LASTNAME',29:'WORK DEPT',        00051002
41:'HIRE DATE',54:'LEVEL',65:'SALARY',                00052002
/,1:'-----',14:'-----',29:'-----',             00053002
40:'-----',54:'-----',65:'-----'),              00054002
OUTREC=(1:3,12,C' ',17,15,C' ',32,3,7C' ',           00060002
36,10,C' ',47,2,BI,M0,5C' ',50,5,PD,M2)              00070002

```

DB2 QUERY OPTION SELECTED

QUERY STATEMENTS:

```

SELECT FIRSTNME, LASTNAME, WORKDEPT, HIREDATE, EDLEVEL, SALARY
FROM DSN.EMPTAB
WHERE EDLEVEL>10

```

INPUT RECORD DESCRIPTION:

COLUMN NAME	START POSITION	END POSITION	DB2 DATA TYPE	SYNCSORT DATA TYPE	NULL VALUE
FIRSTNME	1	14	VARCHAR(12)	CH	DISALLOWED
LASTNAME	15	31	VARCHAR(15)	CH	DISALLOWED
WORKDEPT	32	34	CHAR(3)	CH	ALLOWED
NULLINDICATOR	35	35		CH	
HIREDATE	36	45	DATE	CH	ALLOWED
NULLINDICATOR	46	46		CH	
EDLEVEL	47	48	SMALLINT	BI	ALLOWED
NULLINDICATOR	49	49		CH	
SALARY	50	54	DECIMAL(9,2)	PD	ALLOWED
NULLINDICATOR	55	55		CH	

WER110I SORTOUT : RECFM=FB ; LRECL= 55; BLKSIZE= 27995

WER110I SORTOF1 : RECFM=FBA ; LRECL= 75; BLKSIZE= 27975

WER124I ESTIMATED PREALLOCATED/USED SORTWORK SPACE USAGE FACTOR = 600.00

WER045C END SORT PHASE

WER405I SORTOF1 : DATA RECORDS OUT 10; TOTAL RECORDS OUT 16

WER211I SYNCSMF CALLED BY SYNCSORT; RC=0000

WER449I SYNCSORT GLOBAL DSM SUBSYSTEM ACTIVE

WER246I FILESIZE 550 BYTES

WER054I RCD IN 10, OUT 10

WER169I RELEASE 1.1A BATCH 0999 TPF LEVEL 0

WER052I END SYNCSORT - JOBNAME, SORTSQL, , DIAG=8CFF, 4AD7, A09F, 28FD, D572, 68EB, A6C8, 2462

Figure 215. Sample SYSOUT Report

The following shows the output from the application:

EMPLOYEE INFOMATION					
FIRSTNAME	LASTNAME	WORK DEPT	HIRE DATE	LEVEL	SALARY
CHRISTINE	HAAS	A00	01/01/1975	18	52,750.00
CHRISTINE	MIKE	A00	06/08/1978	12	53,330.00
DIANE	HARISON	A00	02/01/1978	13	12,500.00
DIANE	HEMMINGER	A00	01/01/1975	14	46,500.00
JOAN	PAN	A00	05/01/1973	15	12,110.00
KEVEN	MASK	B00	06/01/1988	14	34,780.00
MAGGIE	NEME	A00	06/01/1978	13	54,330.00
MIKE	BUSH	B00	12/08/1987	11	12,340.00
PETER	MAWAH	B00	02/01/1988	18	30,000.00
STEVE	ARNEY	B00	02/01/1990	18	34,560.00

Figure 216. Sample Application Output

Chapter 12. Tape Sort

When to Use Tape Sort

Traditionally, Tape Sort has been used where there is insufficient disk space for sort work. Currently, MAXSORT is the recommended sorting technique for large applications: (1) MAXSORT's disk SORTWK allocations can be defined independently of SORTIN size; (2) when a MAXSORT job is canceled for higher priority jobs, all previously produced data sets can be used without resorting; (3) all Disk Sort features and performance improvements are available with MAXSORT; (4) significantly less tape drive time is required; and (5) the job can be executed in segments to facilitate production scheduling.

Options Reserved to Disk Sort

These PARMs are reserved for sorts with SORTWK assigned to disk: BALANCE, BMSG, CMP=CPD/CLC, COMMAREA/NOCOMMAREA, CPU, DEBUG, DYNALLOC, ELAP, E15/E35=COB, FILSZ=n, HBSI, HBSO, INCOR=ON/OFF, IO, LIST/NOLIST, NOIOERR, L6/L7=n, PRINT121, NORC16, RELEASE=ON/OFF, RESET, RLSOUT, SDB=xxx, SECOND=ON/OFF, SKIPREC=n, STOPAFT=n, VLTEST=n/0, VLTESI=n/0, and all MAXSORT PARMs.

Tape Sort will *not* validate decimal control fields, permit exits written in COBOL, C, or REXX, check the size of the input file after input processing, run incore, abend without a dump, optimize with HISTOGRM, issue the STIMER macro, adjust the message data set's DCB, release unused SORTWK or SORTOUT space, accept VSAM input, or stop processing after a specified number of records. Control statements and header line(s) appear with SYSOUT if Tape Sort is initiated through job control language; invoked Tape Sorts do not list sort control statements.

Invoked Tape Sorts may only use the 24-bit addressing mode.

These control statements are reserved for sorts with SORTWK assigned to disk: MERGE, INCLUDE/OMIT, INREC, OPTION, OUTREC, UTFIL, SUM, and ALTSEQ. In addition, the SORT statement does not permit COPY, DYNALOC or FILSZ=n.

EXEC Statement

The Tape Sort EXEC statement differs from that for a Disk Sort in that the PGM=SYNCSORT coding cannot be used.

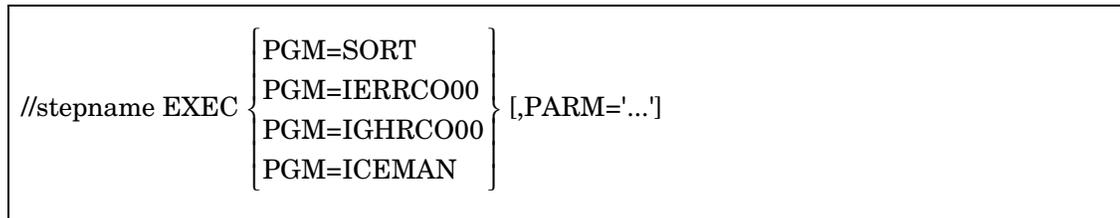


Figure 217. Tape Sort EXEC Statement Format

In addition, the set of PARMs is restricted to the following list.

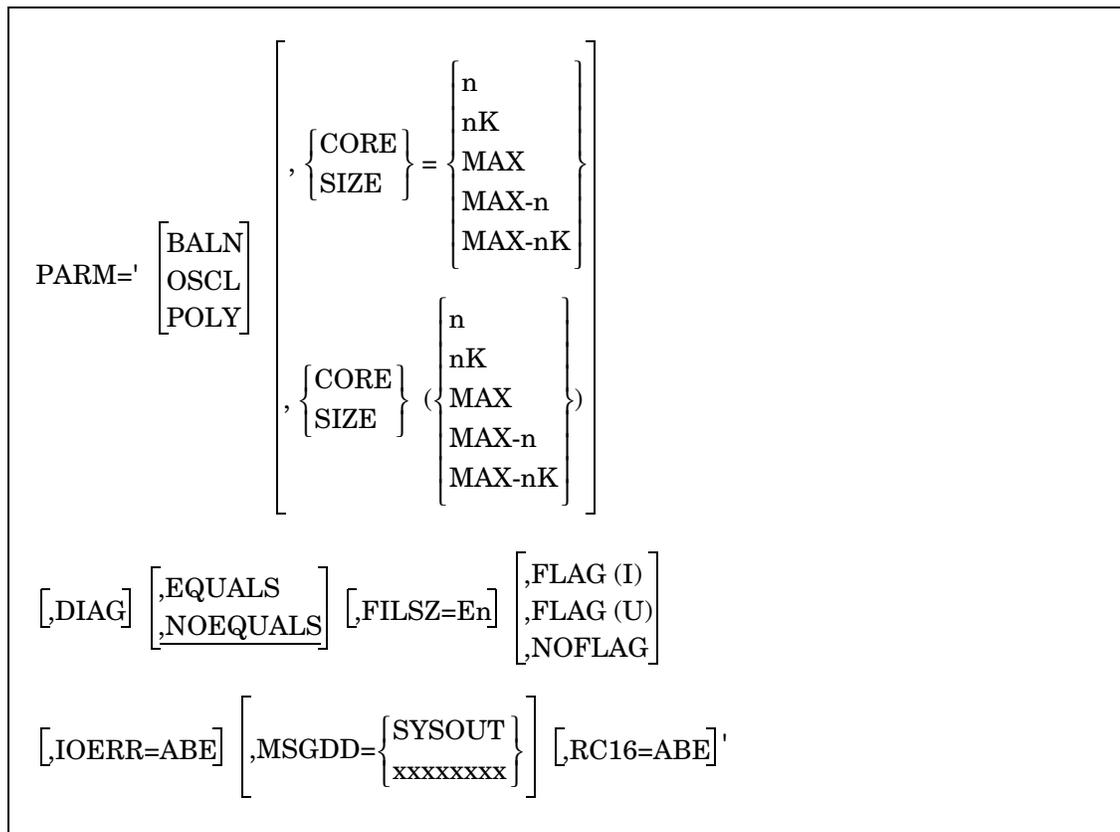


Figure 218. Tape Sort PARM Parameter Format

As usual, the PARM list is enclosed in single quotes.

The BALN/OSCL/POLY PARM option is effective only with Tape Sort, where it specifies the balanced (BALN), oscillating (OSCL) or polyphase (POLY) sorting technique. It is recommended that this PARM value be omitted, permitting the sort to choose the technique best suited to the application.

DD Statements

Tape Sort's DD statements are summarized in the table below. They differ from Disk Sort's only in the additional requirement of a SORTLIB statement. SORTWKxx DD statements are required--Tape Sort is initiated by assigning these to tape. \$ORTPARM is restricted to one 80-byte card image, which can include only PARMs.

Tape Sort DD Statements		
//\$ORTPARM	DD	Used to override PARM information. Restricted to one 80-byte card image.
//SYSIN	DD	Control statement input data set. Required unless the invoking program supplies the address of a 24-bit parameter list.
//SYSOUT	DD	Message data set.
//SORTWKxx	DD	Required. Assigned to tape.
//SORTLIB	DD	Required. Refers to tape sort library.
//SORTIN	DD	Sort input data set. Required unless there is an E15. Ignored if the invoking program supplies an inline E15 exit routine; optional if the MODS statement activates an E15 exit routine.
//SORTOUT	DD	Sort output data set. Required unless there is an E35. Ignored if the invoking program supplies an inline E35 exit routine; optional if the MODS statement activates an E35 exit routine.
//SORTCKPT	DD	Checkpoint data set. Required if Checkpoint-Restart is used.
//SORTMODS	DD	Required if user exits are in SYSIN and if user exits are to be linkage-edited at execution time.
//SYSLIN	DD	
//SYSLMOD	DD	
//SYSUT1	DD	
//SYSPRINT	DD	
//ddname	DD	Library definition of user exits. Required unless exits are in LINKLIB.

Table 42. Tape Sort DD Statements

SORTLIB DD Statement

The SORTLIB DD statement is required to reference the special tape sort program.

```
//SORTLIB DD DSN=SYNCTAPE,DISP=SHR
```

Figure 219. Sample SORTLIB DD Statement

In this example, the tape sort modules are in a partitioned data set called SYNCTAPE. The modules will be used as needed by the Tape Sort control program which was given control by the operating system.

SORTWKxx DD Statement

Tape Sort is initiated by the use of tape devices for intermediate storage. Tape Sort requires a minimum of three SORTWKxx DD statements, numbered consecutively from 01. There may be as many as thirty-two sort work files, making 32 the largest possible nn value.

Tape Sort uses 2400 and 3400 series tape units, including the 3480, 3490 and 3590 cartridge systems, with densities of 800 BPI, 1600 BPI, and 6250 BPI. Each reel of tape must be a full-size 2400-foot reel. It is possible to mix different densities or device types within the same sort, but SyncSort will use the lowest density to calculate the capacity of each SORTWK volume. If 3480 or 3490 IDRC tape drives are used, DCB=TRTCN=NOCOMP must be in effect.

In Figure 220, the xxxx in the UNIT parameter represents the installation--specific name chosen to define a tape device.

```
//SORTWK01 DD UNIT= { 2400  
                    { 3400  
                    { 3480  
                    { 3590  
                    { xxxx
```

Figure 220. SORTWKxx DD Statement Format for Tape Sorts

Calculating Tape SORTWK Requirements

The number of files needed for intermediate storage varies with the size of the input and the sorting technique used (BALN, OSCL or POLY). The table below indicates the minimum required for each of the three sorting techniques. This minimum figure is *not* the recommended number to use--allow more than the minimum number of drives to achieve sorting efficiency.

Sorting Technique	Maximum No. Input Tapes	Minimum No. Tape Drives For SORTWKs	Maximum No. Tapes For SORTWKs	Note that...
OSCL	15	(no. input volumes + two) or four, whichever is larger.	17	(1) An exact or a closely estimated SIZE on the SORT statement is necessary. (2) SORTIN and SORTWK tapes may not be mounted on the same drive.
BALN	15	Two times (no. of input volumes + one).	32	(1) An exact or a closely estimated SIZE is recommended. (2) Provide a generous amount of memory.
POLY	1	Three.	17	POLY is always used, whether specified or not, when only 3 SORTWK tape drives are available. This technique is not recommended unless necessary.

Table 43. Tape Sort Requirements

\$ORTPARM DD Statement

When used in conjunction with Tape Sort, \$ORTPARM can pass only PARM-coded information, and then only by way of a single 80-byte card image. \$ORTPARM *cannot* be used to override control statements for Tape Sort.

The following example changes three PARM options.

```
//$ORTPARM DD *
CORE=128K,EQUALS,FILSZ=E15000
```

Figure 221. Sample Tape Sort \$ORTPARM Data Set

Optimizing Tape Sort

Three factors are crucial to Tape Sort efficiency: a generous amount of intermediate storage, a closely estimated input size value on the SORT or EXEC statement, and the freedom to select the best sorting technique (BALN, OSCL or POLY) based on the nature of the application and conditions at execution time. Accordingly, the number of SORTWK data sets should be in excess of those suggested in the chart above, the BALN/OSCL/POLY PARM should be omitted and an accurate SIZE or FILSZ *estimate* should be provided.

Control Statements

Tape Sort supports only four control statements: SORT, RECORD, MODS and END. Of these four, only MODS and END are supported in their full Disk Sort version, and MODS is available only for JCL-initiated executions.

$$\text{SORT } \left\{ \begin{array}{l} \text{FIELDS} = (p_1, l_1, f_1, o_1, p_2, l_2, f_2, o_2, \dots, p_{64}, l_{64}, f_{64}, o_{64}) \\ \text{FIELDS} = (p_1, l_1, o_1, p_2, l_2, o_2, \dots, p_{64}, l_{64}, o_{64}), \text{FORMAT} = f \end{array} \right\} \\
 \left[\begin{array}{l} \text{,SIZE} = \left\{ \begin{array}{l} n \\ E_n \end{array} \right\} \\ \text{,FILSIZ} = E_n \quad \text{,SKIPREC} = n \end{array} \right] \\
 \left[\begin{array}{l} \text{,EQUALS} \\ \text{,NOEQUALS} \end{array} \right] \left[\begin{array}{l} \text{,CKPT} \\ \text{,CHKPT} \end{array} \right]$$

Figure 222. SORT Control Statement Format for Tape Sort

$$\text{RECORD TYPE} = \left\{ \begin{array}{l} F \\ V \end{array} \right\}, \text{LENGTH} = (l_1, l_2, l_3, l_4, l_5)$$

Figure 223. Tape Sort RECORD Control Statement Format

$$\text{MODS exit-name}_1 = (r_1 \ b_1 \ [d_1] \left[\begin{array}{l} \{,N\} \\ \{,S\} \end{array} \right]) \dots \text{MODS exit-name}_{16} = (r_{16} \ b_{16} \ [d_{16}] \left[\begin{array}{l} \{,N\} \\ \{,S\} \end{array} \right])$$

Figure 224. MODS Control Statement Format

$$\text{END}$$

Figure 225. END Control Statement Format

Exit Programs

With the exception of the merge input exit E32, all of the exits available for Disk Sort are supported for a JCL-initiated Tape Sort. When invoked, however, only E15 and E35 exits are available; these must be coded as subroutines of the calling program. The MODS statement is not supported for an invoked Tape Sort.

Initiating Tape Sort Through JCL/Control Streams

//TAPESORT	JOB		1
//STEPT	EXEC	PGM=SORT	2
//SORTLIB	DD	DSN=TAPESORT.RESI.DENCE, DISP=SHR	3
//MODLIB	DD	DSN=EXIT.E15, DISP=SHR	4
//SYSOUT	DD	SYSOUT=A	5
//SORTIN	DD	DSN=INTAPE, UNIT=3480,	6
//		VOL=SER=385678, DISP=(OLD, KEEP),	
//		DCB=(LRECL=100, RECFM=FB,	
//		BLKSIZE=900)	
//SORTOUT	DD	DSN=OUT.TAPE, VOL=SER=783456,	7
//		UNIT=3480, DISP=(NEW, KEEP)	
//SORTWK01	DD	UNIT=TAPE	8
//SORTWK02	DD	UNIT=TAPE	
//SORTWK03	DD	UNIT=TAPE	
//SORTWK04	DD	UNIT=TAPE	
//SORTWK05	DD	UNIT=TAPE	
//SORTWK06	DD	UNIT=TAPE	
//SORTWK07	DD	UNIT=TAPE	
//SORTWK08	DD	UNIT=TAPE	
//SYSIN	DD	*	9
	SORT	FIELDS=(1, 15, CH, A, 16, 8, BI, D),	10
		EQUALS, SIZE=E20000	
	RECORD	TYPE=F, LENGTH=100	11
	MODS	E15=(E15, 550, MODLIB, N)	12
	END	END TAPE SORT	13
*		MAY ACCOUNTS PROCESSED	14
/*			

Figure 226. Sample JCL/Control Stream

1. The JOB statement gives TAPESORT as the name of the job.
2. The EXEC statement identifies SORT as the program to be executed.
3. The required SORTLIB DD statement instructs the system to look for the Tape Sort secondary modules in the SORTLIB library under the data set name TAPESORT.RESI.DENCE. The DISP shows that this library may be shared.
4. The MODLIB DD statement defines the partitioned data set in which the exit routine resides. (Note that MODLIB is referenced in the MODS control statement.) The data set name of the exit library is EXIT.E15, and the DISP shows that the library may be shared.

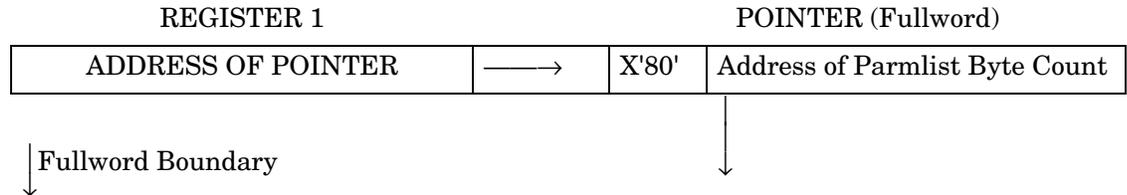
5. The SYSOUT DD statement assigns the SyncSort messages to the output device associated with SYSOUT class A.
6. The SORTIN DD statement gives INTAPE as the input data set name, and specifies a 3480 tape unit with the volume serial number 385678. The DISP shows that the data set is already in existence.

The DCB parameter shows in LRECL of 100 bytes, a fixed blocked RECFM, and a 900 byte BLKSIZE.

7. The SORTOUT DD statement gives OUT.TAPE as the output data set name, and specifies a 3480 tape unit with the volume serial number 783456. The DISP parameter shows that this data set is not in existence yet. The DCB parameter for the SORTOUT data set defaults to that of SORTIN.
8. The eight SORTWK DD statements indicate that 8 tapes are to be used for intermediate storage.
9. The SYSIN DD * statement marks the beginning of the system input stream that includes the program control statements.
10. The SORT control statement shows that two control fields are to be sorted on. The first begins on byte 1 of the record, is 15 bytes long, has character data, and is to be sorted in ascending order. The second begins on byte 16 of the record, is 8 bytes long, contains binary data, and is to be sorted in descending order. EQUALS requests that records with equal control fields leave the sort in the same order in which they came in. The SIZE parameter shows that the number of records in the input data set is *estimated* at 20,000.
11. The RECORD control statement shows that fixed-length records are being sorted. The LENGTH parameter shows 100 byte records at input, during the sort, and at output time.
12. The MODS control statement gives E15 as the exit-type. The name of the exit routine is also E15. It will take 550 bytes in main storage, and resides in a library defined on the MODLIB DD statement. (See the MODLIB DD statement.) The N indicates that link-editing has already been performed.
13. The END control statement marks the end of the control statements. A comment is given.
14. This is a comment statement.
15. The delimiter statement marks the end of the SYSIN input stream.

Invoking Tape Sort from a Program

When initiating Tape Sort from a program, only certain parameters from the parameter list are processed. The X'02' (MODS), X'04' (merge input files), X'05' (DEBUG), X'06' (ALTSEQ), X'07' (SUM), X'08' (INCLUDE/OMIT), X'09' (OUTREC), X'0A' (INREC), X'0B' (OUTFIL), X'F6' (ALTSEQ translation table), and X'F7' (User address constant) parameters are ignored. When invoked, Tape Sort supports only the E15 and E35 exits, which must be coded in line with the invoking program.



	Byte 1	Byte 2	Byte 3	Byte 4
			Number of bytes in following list	
Required in order shown	X'00'	Beginning address of SORT statement		
	X'00'	Ending address of SORT statement		
	X'00'	Beginning address of RECORD statement		
	X'00'	Ending address of RECORD statement		
	X'00'	Address of E15 exit routine (zeros if none)		
	X'00'	Address of E35 exit routine (zeros if none)		
Optional	X'00'	Main storage value		
	X'01'	Reserved main storage value		
	X'03'	Beginning address of message DD name to replace SYSOUT		
	X'05'	Beginning address of DEBUG statement (not processed)		
		Ending address of DEBUG statement		
	X'FD'	IMS flag		
	X'FE'	Pointer to STAE work area (may code zeros if none)		
	X'FF'	Message options (code in EBCDIC)		
		DIAG option (code in EBCDIC)		
		BALN OSCL or PLY (code in EBCDIC)		
		CRCX, PEER or LIST (not processed)		
		DD name prefix to replace SORT in JCL (code in EBCDIC)		

Table 44. Tape Sort Parameter List

	.		
	.		
	.		
	LA	1, PTRWORD	Load address of pointer to parameter list
	LINK	EP= SORT	Initiate tape sort
	LTR	15, 15	Test tape sort return code
	BNZ	SORTERR	Branch on error condition
	B	SORTOK	Branch to normal processing
	CNOP	0, 4	Fullword alignment for pointer
PTRWORD	DC	X'80'	Indicates pointer to parameter list
	DC	AL3(PARMS)	Address of parameter list
	DS	H	Unused first two bytes of first parameter
PARMS	DC	Y(28)	Byte count of remaining parameters
	DC	A(SORTBEG)	Beginning address of sort statement
	DC	A(SORTEND)	Ending address of sort statement
	DC	A(RECBEG)	Beginning address of record statement
	DC	A(RECEND)	Ending address of record statement
	DC	A(E15)	Address of E15 exit routine
	DC	A(E35)	Address of E35 exit routine
	DC	C'DIAG'	Turns on IOERR=ABE and RC16=ABE options
SORTBEG	DC	C'SORT FIELDS=(19,6,PD,A,5,10,CH,A),EQUALS'	Sort image begins
SORTEND	DC	C' '	Sort image ends
RECBEG	DC	C'RECORD TYPE=V,LENGTH=(104,,,64,84)'	Record image begins
RECEND	DC	C' '	Record image ends
	USING	*,15	Using 15 as base register
E15	DS	0H	Exit E15 has full responsibility for sort input
	.		
	.		
	.		
	USING	*,15	Using 15 as base register
E35	DS	0H	Exit E35 has full responsibility for sort output
	.		
	.		
	.		
SORTERR	DS	0H	Error routine for unsuccessful sort
	.		
	.		
	.		
	BR	14	Return to invoking program
SORTOK	DS	0H	Normal processing for successful sort
	.		
	.		
	.		
	BR	14	Return to invoking program

Figure 227. Sample Invoked Tape Sort

In this example, Tape Sort's input file is provided by the in-line E15 exit routine. Since this causes the sort to ignore a SORTIN DD statement, the required RECORD control statement must include the LENGTH parameter. The RECORD statement specifies that the

records to be sorted are variable in length, ranging from 64 to 104 bytes long at sort time, that the most frequent input record length is 84, and that the maximal length of 104 is not changed by the E35 exit routine; all length values include the Record Descriptor Word's 4 bytes. The records are sorted in ascending numeric order by the packed decimal data in the 15th - 20th data bytes in the record. Records with equal numeric values in this field are further sorted by the character data in their first 10 data bytes, in ascending order. Records with equal control keys are passed to the E35 exit routine in the same order as they were generated by the E15 exit routine (the EQUALS parameter). The DIAG option is also set.

Chapter 13. Performance Considerations

Disk Sort? MAXSORT? PARASORT? Tape Sort?

Disk Sort provides the current, established sorting technique, suitable for most sort/merge applications. Intermediate storage is allocated on disk devices and the sort size is limited by the allocated disk space plus secondary extents automatically obtained by the sort.

MAXSORT, SyncSort's maximum capacity sorting technique, is not limited by disk space availability. MAXSORT determines how much data can be sorted using the available disk work space and divides SORTIN into SORTWK-manageable segments; the sorted segments are stored on tape, disk or MSS for a later, automatic merge. MAXSORT makes all the Disk Sort operational optimizing features and modern programming options available to large sorts, and additionally provides an enhanced breakpoint/restart capability for greater scheduling flexibility--the user can stop MAXSORT processing at selected intervals without loss of sorted output.

PARASORT improves elapsed time performance for sorts whose input is read from a multi-volume tape data set and/or concatenated tape data sets. The performance improvement from PARASORT is a result of processing the SORTIN input volumes in a parallel fashion. PARASORT requires two to eight times the current number of tape units, depending upon resource availability and the degree of improvement desired. PARASORT automatically manages the tape units and minimizes the use of the tape drive resources by deallocating excess tape drives during initialization and releasing all the extra units at the end of the sort input phase.

By definition, Tape Sort uses tape for intermediate storage. This inhibits such state-of-the-art sorting techniques as sophisticated disk I/O methods, high order merges, and modern

parameter capabilities. For these reasons, MAXSORT performance is far superior to that of Tape Sort. MAXSORT is therefore the preferred method of handling all applications previously routed to Tape Sort.

To convert a Tape Sort execution to MAXSORT, these changes must be made:

- Use the \$ORTPARM DD statement to pass the MAXSORT PARM or (if the sort is initiated through JCL) add PARM='MAXSORT' to the EXEC statement.
- Allocate SORTWK files to disk (SORTWK requirements are independent of SORTIN size).
- Supply a SORTBKPT DD statement.
- Provide 2 or more SORTOU n DD statements for the intermediate output and, if these are allocated to tape (as is usual), a SORTOU00 DD statement.
- If exits are included or the sort is invoked, supply a SORTCKPT DD statement in case restart should be necessary.
- (Optional.) Remove the SORTLIB DD statement.

Note that it may be necessary to supply additional memory in order to execute MAXSORT.

JCL Sorts vs. Program-Invoked Sorts

When SyncSort is initiated from a COBOL program, the calling program handles I/O, remains in core, and generally retards sort execution. SyncSort will yield maximum performance through proper synchronization of all data whenever it has control of the sorting process, i.e., whenever // EXEC PGM= SYNC SORT is used.

From the point of view of performance, the JCL-initiated sort execution has the advantage. Whenever possible, tasks incidental to the sort/merge/copy process should be handled via SyncSort control statements. Where this is not possible, the JCL/control stream should be supplemented with user-written exit routines. Ideally, the exit routines exist as load modules, so that they do not require link-editing every time the job is run. SyncSort permits exit routines to be written in COBOL, C, FORTRAN, REXX, or Assembler language.

If you must invoke the sort from a COBOL program, you may improve sort performance by passing an accurate FILSZ= n / E n parameter via \$ORTPARM.

Control Statement Issues

SyncSort control statements can be used to eliminate records from the input file (INCLUDE/OMIT), summarize and/or eliminate equal-keyed records (SUM), reformat records (INREC/OUTREC), set up multiple output files (OUTFIL) or write formatted reports (OUTFIL statement with HEADER, TRAILER, SECTIONS and OUTFIL parame-

ters. These control statements provide a high performance alternative to the use of exits and invoking programs. The tasks they address are those which are most frequently executed and/or improve sort performance. Since sort throughput is in part a function of the number of bytes that are to be manipulated, considerable performance savings can result from using the INCLUDE/OMIT statement to eliminate irrelevant records; INCLUDE/OMIT affects the data set prior to sorting/merging/copying. The SKIPREC and STOPAFT parameters are recommended for test runs of sorting applications for the same reason. When the file bias is high enough for a significant number of records to be summarized early in the sort, SUM will also provide performance gains if the XSUM option has not also been selected. When reformatting records, it is desirable to minimize the amount of data that must pass through the sort process. Other things being equal, INREC should be used to shorten records, OUTREC to lengthen them.

The Efficient Use of PARMs

There are four programming PARMs that may have a significant effect on sort performance: CMP, EQUALS, STOPAFT and SKIPREC.

This CMP PARM specifies the kind of compare operation to be used for sort/merge control fields up to 16 bytes long, bearing the format code PD or ZD. When CMP=CPD, the default, is used, ZD fields are PACK'ed and then compared. Invalid PD data may cause a system 0C7 abend and program termination. The integrity of fields labelled "ZD" is only guaranteed when they contain valid ZD data. The delivered default of the VLTEST PARM supports CMP=CPD, as do certain other VLTEST PARM values. Whenever possible, set CMP=CPD for better sort performance.

The alternative, CMP=CLC, is a more costly option--it forces the sort to extract potentially invalid PD and ZD fields and do a certain amount of data manipulation to obtain valid sign comparisons.

The EQUALS PARM instructs the sort/merge to preserve the order of equal-keyed records. EQUALS will have a slight but generally significant impact on sort performance. By making EQUALS available on an individual sort basis, SyncSort makes this programming option available where it is needed, without imposing it on the installation's more routine jobs. For sort efficiency, use EQUALS only where the preservation of the input order of equal-keyed records is important.

The user interested in sort performance will specify the STOPAFT PARM in test runs of the sort. With STOPAFT=n, only the first n records of the input file will be sorted. By reducing the number of records to be processed, STOPAFT improves sort performance. If additional tests are necessary, the SKIPREC PARM can be used together with STOPAFT to select a different subset of the SORTIN data set.

Optimizing System Resources

The efficiency of sort processing is measured in terms of the performance measures of CPU time, elapsed time, and I/O activity. Ordinarily, when SyncSort performs a sort, it seeks to balance these performance measures in a way that yields the best *overall* sort performance. It is possible, however, to define a particular performance measure as more important than others for a particular job. This can be done through SyncSort's Dynamic Storage Management (DSM) facility, which makes available four optimization modes for sort processing. These are BALANCE, CPU, ELAP and IO. BALANCE is the default optimization mode which provides the best overall balance between CPU time, sort elapsed time and I/O activity to SORTIN, SORTOUT and SORTWK. If CPU time is given the highest priority, SyncSort will minimize this resource at the expense of elapsed time and I/O activity. Selecting ELAP as the optimization mode will cause SyncSort to minimize the elapsed (wall clock) time of each sort, usually at some expense of the sort's CPU time. Likewise, if IO is selected as the optimization mode, SyncSort will minimize the I/O activity (EXCPs) performed by the sorts.

Setting CORE

The following examples illustrate the most common types of alternative codings for the CORE PARM:

```
CORE=MAX-30K  
CORE=500K  
CORE=MAX
```

From the perspective of memory management, there are three types of sort executions, requiring three different approaches to CORE coding: invoked sorts, JCL sorts with exit routines, and JCL sorts without exit routines.

In the first case, where for example, a COBOL program calls SyncSort via the SORT verb, the sort and the invoking program (including its Input Procedure and Output Procedure) are all in memory at the same time. The only dynamic aspect to memory management in this case is the acquisition of memory for the buffers of any files opened by the Input and Output Procedures; it is only when a file is opened that the memory for the file's buffers is obtained. Therefore, all data sets required during the sort should, if possible, be opened before invoking the sort.

The coding of the CORE parameter must make allowances for the Input and Output Procedures' file buffers by reserving enough memory for the greater of the two procedures' requirements. If, for example, the Input Procedure's files require 50K and the Output Procedure's files require 100K, SyncSort should be instructed to set aside 100K for their use; code CORE=MAX-100K. If CORE=MAX is coded, it is likely that no memory will be available for buffers when the Input or Output Procedure attempts to open a file, resulting in an ABEND80A message. If CORE is coded with a constant value such as CORE=756K, there is still the possibility of an ABEND80A message since the constant value requested (in this

case, 756K) may account for all the memory available, again leaving no memory for the buffers.

With CORE=MAX-100K, the precise amount of memory used by the sort depends both on the amount of memory that is available and on the maximum value set at sort installation time (the site maximum). Since this form of the parameter ensures that 100K of the total memory available to this job will be set aside for the buffers, CORE=MAX-100K will not produce an ABEND80A message. Note that MAX-value must be greater than the minimum memory requirement for SyncSort execution.

The table below illustrates the relationship between the site maximum and the available memory for an invoked sort requiring 100K bytes worth of buffers for the Input and Output Procedures. In reviewing the table, note that the site maximum sets an absolute ceiling on the amount of memory that can be used by the sort; even if additional memory is available, it is not available to the sort. This additional memory would, however, be available to the Input or Output Procedure for file buffers, accounting for some of the normal sort terminations indicated. Since the programmer has no way of knowing whether these conditions will hold at execution time, CORE=MAX-100K remains the preferred method of setting memory for an invoked sort with 100K bytes worth of buffers.

The COBOL programmer has the option of setting CORE by means of the SORT-CORE-SIZE special register. In order to set memory aside for the buffers, the invoking program places a negative value into the special register prior to sort execution; CORE=MAX-100K is equivalent to *MOVE-102400 TO SORT-CORE-SIZE*. Under VS COBOL II or COBOL/370, CORE can be set by submitting the CORE=MAX-nK PARM via the \$ORTPARM data set.

Site Maximum	Available Memory	CORE=	Memory used by the Sort	Available for 100K Buffer Space	Probable Sort Return Code
1024K	1024K	MAX	1024K	0	S80A - No core for buffers
1024K	512K	MAX	512K	0	S80A - No core for buffers
1024K	2048K	MAX	1024K	100K	0
1024K	900K	756K	756K	100K	0
512K	2048K	756K	512K	100K	0
1024K	756K	756K	756K	0	S80A - No core for buffers
1024K	512K	MAX-100K	412K	100K	0
512K	512K	MAX-100K	412K	100K	0
512K	1024K	MAX-100K	512K	100K	0
1024K	360K	MAX-100K	260K	100K	0 - Inefficient sort

Table 45. Illustration: CORE Alternatives for Invoked Sort with 100K Buffer Space

When exits are included, the optimal coding of the CORE parameter depends on the memory value in the MODS control statement. As in the case of the COBOL Input/Output Procedure, coding CORE=MAX-100K will set aside 100K bytes for buffers. If the MODS

statement's memory value included sufficient buffer space, code `CORE=MAX`; *coding anything but `CORE=MAX` nullifies the `MODS` memory value(s)*. Again, the site maximum prevents the sort from appropriating too much memory. When the exit program is not referenced in a `MODS` statement (e.g., when an E15/E35 exit routine is coded in line with an invoking Assembler program) or the `MODS` memory value accounts only for the program's code, memory must be reserved for the buffers of any files to be opened. *An Assembler program's in-line E15/E35 exit routine is equivalent to a COBOL Input/Output Procedure.*

In JCL sorts without exit routines, it is not necessary to code any core parameter. SyncSort will use as much of the site maximum as is available at the time of execution. Thus, if the site maximum is set to 1024K and 2048K bytes are available, SyncSort will use 1024K.

The Incore Sort

An incore or turnaround sort is only possible when the `INCORE PARM` has the value `ON`. (This is the delivered default for this `PARM`.)

Whenever there is sufficient memory, `INCORE=ON` permits the standard Disk Sort to sort all the input data within its memory area, without writing to any of the work data sets that may have been provided. Sufficient memory, as discussed here, means that SyncSort's memory area/address space is large enough to hold the SyncSort program, all of the input data, `SORTIN` or `SORTOUT` buffers (whichever are larger) and, if work data sets are allocated, `SORTWKxx` buffers.

The incore sort is not available to Disk Sorts taking checkpoints, using `SUM`, `OUTREC`, `OUTFIL`, an E14 or E16 exit routine, or producing VSAM output. `INCORE=ON` is ignored by Tape Sort, `MAXSORT` and `PARASORT`.

When Can a Sort Run Entirely in Main Storage (No SORTWK Needed)?

An Incore Sort is possible when all of the data that is to be sorted can be contained in main storage. For most simple applications:

Number of records that will fit in main storage = $\frac{A-(B+C)}{D + 12}$

A = Core available to SyncSort.

B = 200K

C = The greatest of: 2 X `SORTIN` block size, 2 X `SORTOUT` block size, and 15% of A.

D = Average record length of data being sorted.

Note: `SORTWK` data sets are *required* in order to use `SUM`, `OUTREC`, `OUTFIL`, a VSAM `SORTOUT` data set, checkpoint/restart, an E14 or E16 exit routine, `MAXSORT` or `PARASORT`.

Disk Space Considerations

Tuning Disk Space Allocations

With the operational features SECOND and RELEASE turned ON (this is the delivered default), SyncSort automatically supplements and releases any disk space the user allocates for intermediate storage, making the allocation of the correct amount of SORTWK space an automatic, sort-controlled process. For general sorting purposes, the user need not be concerned with precise SORTWK space allocations. However, allocating SORTWK space in cylinders, rather than blocks or tracks, will usually yield optimal performance.

For best performance with filesizes greater than 30 megabytes, especially when DYNALLOC is not enabled, allocate the required space across 4 to 6 SORTWK devices.

Message WER124I is provided in some applications in order to permit the user who is interested in a finely tuned sort execution to improve intermediate storage allocation for future runs. Routinely overallocating SORTWK, relying on RELEASE=ON, will delay sort step execution until all the space requested (including the excess space) is available, and will waste this excess space until its released at the end of Phase 1. Routinely underallocating by a large amount assumes that the needed storage will always be physically available. If, for some reason, the required storage cannot be obtained on any volume assigned for sort work areas, SyncSort will terminate with a SORT CAPACITY EXCEEDED error.

A sort is considered to be finely tuned when WER124I reports an overallocation factor between 1.00 and 1.50.

The Impact of Disk Space on the Work Data Sets on SyncSort

SyncSort's work data set disk space management is automated to a very high degree. It can:

- Automatically correct the underallocation of disk space by obtaining secondary allocations of disk space, as needed. This prevents costly SORT CAPACITY EXCEEDED terminations.
- Automatically release excess disk space at the completion of Phase 1. The space immediately becomes available for allocations to other jobs.
- Dynamically allocate work data sets through the MVS DYNALLOC capability.

You can improve the efficiency of disk space usage by allocating optimally at the outset.

Disk Sort Intermediate Storage Calculation Formulas

Approximate Number of Tracks Required = $\frac{A \times B \times 1.3}{C}$

where:

A = Number of records to be sorted.

B = Average record length of data being sorted.

C = Track capacity of the work device:

DEVICE TYPE	TRACK CAPACITY IN BYTES
3350	19,069
3375	36,000
3380	47,476
3390	56,664
9345	46,456

Allocating disk storage space in cylinders rather than tracks will improve the performance of SyncSort. When converting tracks to cylinders, round the number of cylinders up to the higher number. For example, if 9.5 cylinders are needed, allocate 10 cylinders.

Tape Sort Intermediate Storage Calculation Formulas

BALN Number of tape units required = $2(A+1)$

Maximum number of input volumes: 15

Minimum number of areas = 4

Maximum number of areas = 32

POLY Number of tape units required = 3

Maximum number of input volumes: 1

Minimum number of areas = 3

Maximum number of areas = 17

OSCL Number of tape units required = The greater of 4 and $A + 2$

Maximum number of input volumes: 15

Minimum number of areas = 4

Maximum number of areas = 17

where

A = Number of input volumes

Note: These formulas are based on work units 2400 feet long, of the same density as the input volumes, or 3480 tape cartridges used as work units.

Special Considerations Concerning SyncSort's Disk Space Management on Work Data Sets

SyncSort implements the automatic space management facility by reading the JFCB and modifying the SPACE parameter to enter a secondary allocation quantity (or accept one that was coded) and the RLSE subparameter. The JFCB is the z/OS control block that represents the DD statement.

Several SyncSort options may be implemented which affect the disk space management of the sort. (See the *Default Options* chapter of the *Installation Guide*.) Specific functions of these features are as follows.

1. The *automatic* use of secondary allocation and/or release can be selectively suppressed.
2. The *amount* of secondary space per allocation can be modified.
3. The use of RLSE can be suppressed.
4. The use of space release is suppressed for small sorts, including the in-core sort, in order to minimize system overhead. For non-in-core sorts, if the file size is less than 4 megabytes, space release is normally suppressed. The 4 megabytes threshold may be altered.
5. The use of space release is normally suppressed for all invoked sorts to prevent SORT CAPACITY EXCEEDED termination when SyncSort is invoked more than once by a single program. If the first sort involves a modest volume of data, and causes space release to make the work data sets smaller, and the second sort is larger, the second sort might not find sufficient work space. Your installation can turn on space release for invoked sorts and thus save disk space. This very rarely causes problems because (1) few programs invoke the sort more than once and (2) SyncSort's automatic secondary allocation normally prevents a SORT CAPACITY EXCEEDED termination.
6. SyncSort can routinely DYNALLOC data sets for every run.

Other factors which may result in suppression of these features include:

1. Automatic release is suppressed for permanent data sets unless additional sort work space has been allocated.
2. Automatic release is normally suppressed by the installation for initiator-dedicated data sets.

SyncSort uses normal z/OS facilities to obtain secondary allocations on work data sets. Consequently, the sort, like any other program under z/OS, is restricted to sixteen extents per data set. SyncSort, however, will recover from system B37 ABENDS that other pro-

grams might encounter in attempting secondary allocations. If SyncSort determines that a particular sort work data set cannot sustain a secondary allocation because it already has sixteen extents or because there is not enough space left on the volume, it does not attempt secondary allocation on that data set. SyncSort further checks all other work data sets, and if none of them can sustain a secondary allocation, it must abort with SORT CAPACITY EXCEEDED.

SyncSort often avoids the use of one or more work data sets to minimize overall system conflict, as, for instance, between SORTIN and a work data set. It may obtain secondary allocation on some data sets while releasing on others.

The Coding and Use of Checkpoint-Restart

Occasionally, a hardware failure may prevent the successful completion of a sort or merge. Examples include a physically defective output volume or device, or a failure of the operating system for reasons unrelated to the sort. Since sorts tend to consume more system resources than any other type of application, it may be advantageous to be able to resume execution just before the failure occurred rather than restart the job at the beginning of the failed job step. SyncSort provides this restart capability through its support of the standard z/OS Checkpoint-Restart feature.

To instruct SyncSort to take checkpoints, code CKPT or CHKPT (either spelling) on the SORT/MERGE control statement and supply a SORTCKPT DD statement.

For a sort, checkpoints are taken at the beginning of Phase 3 before the output data sets (if any) are opened, and at every end-of-volume of a SORTOUT data set when OUTFIL is not in use. An operator may then restart the sort at Phase 3 or at any end-of-volume checkpoint. If necessary, a new output volume or device with identical characteristics to the defective volume or device may be substituted.

For a merge or copy, SyncSort takes a checkpoint at every end-of-volume of a SORTOUT data set when OUTFIL is not in use.

Checkpoints cannot be taken within a user exit routine.

The DISP Parameter for SORTCKPT, SORTWKxx and SORTOUT Data Sets

The coding of the DISP parameter for these data sets depends in part on the PARM-specified response to an unsuccessful sort. There are four cases:

- NOIOERR and NORC16 (These are the delivered defaults.)
- IOERR=ABE and RC16=ABE
- IOERR=ABE and NORC16
- NOIOERR and RC16=ABE

When return code 16 is issued by the unsuccessful sort (i.e., when an I/O error occurs and NOIOERR is set or, for other errors, when NORC16 is set), the *second* subparameter of the DISP parameter should be specified as KEEP or CATLG. When the unsuccessful sort causes a user abend (i.e., when IOERR=ABE for I/O errors, RC16=ABE for other errors), the *third* subparameter of the DISP parameter should be specified as KEEP or CATLG. Thus, with NOIOERR and RC16=ABE or with IOERR=ABE and NORC16, both the second and the third DISP subparameter should be specified as KEEP or CATLG. Unless the DISP parameter is coded in accordance with these two PARM values, restart will be impossible.

It is recommended that these data sets be deleted upon successful completion of the sort. This can be done by coding the COND parameter for an IEFBR14 step to follow the sort step in the jobstream. The COND parameter makes the IEFBR14 (data set deletion) execution depend upon the successful completion of the previous step (the sort).

The SORTCKPT Data Set

Assign a permanent DSN to the SORTCKPT DD statement and specify the UNIT, SPACE and VOL=SER parameters to make the operator's job easier should a deferred restart become necessary.

```
//SORTCKPT DD UNIT=3390,DSN=sort.ckpt,
//          SPACE=(CYL,(1,1)),
//          VOL=SER=WORK01,DISP=(MOD,KEEP,KEEP)
```

Figure 228. Sample SORTCKPT DD Statement

The SORTWKxx Data Set(s)

Assign a permanent DSN to every SORTWKxx DD statement and specify the UNIT, SPACE and VOL=SER parameters in case a deferred restart becomes necessary. Avoid using passed data sets, JCL refer-backs, and any other references which would make the JCL following the restart dependent on the JCL preceding the restart.

Note that the SORTCKPT data set and the SORTWKxx data set(s) may reside on the same direct access device without loss of efficiency.

```
//SORTWK01 DD UNIT=3390,DSN=sort.wk01,
//          SPACE=(CYL,(20,10)),
//          VOL=SER=WORK02,DISP=(,KEEP,KEEP)
```

Figure 229. Sample SORTCKPT DD Statement

Automatic Checkpoint-Restart

With automatic checkpoint-restart, the operating system will ask the operator whether an unsuccessful/abending step should be restarted. A "yes" reply instructs the system to restart the job at the last checkpoint taken. If the operator replies "no", the job will still be eligible for deferred checkpoint-restart, but its control statements will have to be modified before the job is resubmitted.

The requirements for automatic checkpoint-restart are:

- The sort step must have a unique name.
- The JOB statement must specify RD=R and MSGLEVEL=1.
- All system completion codes with which the sort may abend should be defined at system generation time as being eligible for restart. If the RC16=ABE and/or IOERR=ABE options are in effect, for example, then user abend codes 16 and/or 999 must be eligible for restart.
- User-written exit routines and calling programs may not issue the STIMER macro.

```
//AUTOCKPT JOB (1101,2333),P.ARBEAU, RD=R,
// MSGLEVEL=(1,1)
//XVISORT EXEC PGM=SORT,
// PARM='RC16=ABE,IOERR=ABE'
//SYSOUT DD SYSOUT=A
//SORTIN DD DSN=XVI.SORTIN,DISP=OLD
//SORTOUT DD UNIT=TAPE,DISP=(,CATLG,KEEP),
// DSN=XVI.SORTOUT
//SORTWK01 DD UNIT=3390,DISP=(,DELETE,KEEP),
// VOL=SER=WORK01,DSN=XVI.SORTWK01,
// SPACE=(CYL,40)
//SORTWK02 DD UNIT=3390,DISP=(,DELETE,KEEP),
// VOL=SER=WORK02,DSN=XIV.SORTWK02,
// SPACE=(CYL,40)
//SORTCKPT DD UNIT=3390,DISP=(,DELETE,KEEP),
// VOL=SER=WORK02,DSN=XVI.SORTCKPT,
// SPACE=(CYL,(1,1))
//SYSIN DD *
SORT FIELDS=(1,10,CH,A),CKPT
/*
```

Figure 230. Sample Automatic Checkpoint-Restart JCL Stream

Deferred Checkpoint-Restart

Unlike automatic checkpoint-restart, deferred checkpoint-restart requires that certain JCL changes be made before resubmitting the job.

The requirements for a deferred restart are:

- A SYSCHK DD statement must appear immediately before the first EXEC statement in the job. The SYSCHK DD must use the same DSN name as the SORTCKPT DD of the sort that failed. Specify UNIT, VOL=SER, and DISP=(OLD,KEEP).
- The RESTART parameter must be specified, and must provide the job stepname and the PROC stepname (if any) associated with the step containing the failed sort, as the first subparameter. (Separate the two stepnames by a period.) The second subparameter should contain the checkpoint ID of the last checkpoint taken before the sort failed. This can be determined from the console messages given for the job. For JCL sorts, the ID is usually "Cnnnnnnn," referring to the sequence number assigned by the operating system.
- SORTIN and SYSIN DD DUMMY statements are permissible if the program is being restarted at a point where they are no longer needed.

```
//DEFCKPT      JOB      (5433,2333),PAT.TAIG.NANT,          1
//
//              RD=R,MSGLEVEL=(1,1),
//              RESTART=(XVISORT,C0000001)                1
//SYSCHK        DD      UNIT=3390,DISP=(OLD,KEEP),         1
//              VOL=SER=WORK02,
//              DSN=XVI.SORTCKPT                          1
//XVISORT       EXEC    PGM=SORT,
//              PARM='RC16=ABE,IOERR=ABE'
//SYSOUT        DD      SYSOUT=A
//SORTIN        DD      DUMMY                              1
//SORTOUT       DD      UNIT=TAPE,DISP=(,CATLG,KEEP),
//              DSN=XVI.SORTOUT
//SORTWK01      DD      UNIT=3390,DISP=(OLD,DELETE,KEEP),
//              VOL=SER=WORK01,DSN=XVI.SORTWK01,
//              SPACE=(CYL,40)
//SORTWK02      DD      UNIT=3390,DISP=(OLD,DELETE,KEEP),
//              VOL=SER=WORK02,DSN=XVI.SORTWK02,
//              SPACE=(CYL,40)
//SORTCKPT      DD      UNIT=3390,DISP=(MOD,DELETE,KEEP),
//              VOL=SER=WORK02,DSN=XVI.SORTCKPT,
//              SPACE=(CYL,(1,1))
//SYSIN         DD      DUMMY                              1
```

Figure 231. Sample Deferred Checkpoint-Restart JCL Stream

1. This JCL differs from automatic checkpoint-restart JCL.

Optimizing Data Set Placement

The Impact of Work Devices on SyncSort

The performance of SyncSort is almost totally independent of the number of work data sets. The sort's performance may, however, be strongly influenced by the number of devices to which the work data sets are allocated. Generally, for any sort of significant size, the more work devices, the better the sort can perform. If the sort file size is small, however, performance improvements might be outweighed by increased overhead in managing the extra data sets. Increasing the number of work devices will:

1. Improve the overlap between CPU and I/O processing.
2. Improve the effectiveness of SyncSort's integrated activity monitoring.
3. Reduce the likelihood of SORT CAPACITY EXCEEDED.

Increasing the number of data sets without increasing the number of devices will only increase overhead. It is suggested that, as an initial standard, you implement:

1. Four work devices if file size > 100 MB (megabytes).
2. Three work devices if $100\text{MB} \geq \text{file size} > 10\text{MB}$.
3. Two work devices if $10\text{MB} \geq \text{file size} > 1\text{MB}$.
4. One work device if file size $\leq 1\text{MB}$.

In all cases, allocate one work data set per work device.

Obtaining Device Separation

The easiest way to increase the number of work devices is to increase the number of work data sets. This tends to increase the number of devices to which work data sets allocate, although the relationship between the two may be complex and unpredictable.

Try to ensure that every sort has at least one work data set on a pack that does not contain SORTIN or SORTOUT. SyncSort will avoid work data set contention with SORTIN and SORTOUT if it can.

Channel Separation

Try to obtain as many paths to the work devices as possible. It is particularly desirable to provide some path to the work data sets that will not be jammed with traffic from SORTIN or SORTOUT.

On the other hand, SORTIN and SORTOUT may be on the same channel, or even the same device, without any performance loss.

Device Type Considerations

Avoid using a mixture of device types with different track capacities for the work data sets, since SyncSort sacrifices some efficiency if this is the case.

If you must choose between two different disk device types for the work data sets, use the faster; if they are close in speed, use the one with the larger track size.

Avoid the use of VIO data sets for work data sets.

If you must use tape work data sets, allocate as many as possible.

Chapter 14. The HISTOGRM Utility Program

What Is HISTOGRM?

HISTOGRM is a separate program which is used to gain information about variable-length files. The program scans a variable-length file and provides information which can then be used to run more efficient sorts. HISTOGRM can report the:

- Block count for minimum and maximum block lengths
- Record count for minimum and maximum record lengths
- Average record length
- Total number of bytes in the file
- Total number of blocks in the file
- L6 value (average work space) for variable-length records
- L7 value (segment length) for variable-length records

HISTOGRM can be used to analyze variable-length records in a VSAM entry-sequenced or key-sequenced data set. When HISTOGRM processes a VSAM file only record information is gathered; block statistics are not produced.

Using HISTOGRM to Determine L6 and L7 Values for SyncSort

The L6 and L7 values HISTOGRM calculates are passed to SyncSort via the L6, L7 PARM options or the l_6 , l_7 values in the LENGTH parameter of the RECORD control statement. (When there is a conflict, the PARM specification takes precedence.) These values are ignored in a merge or copy application.

Control Parameters for HISTOGRM

The control parameters are outlined below; defaults are underlined>. To specify other values, include a control statement in the SYSIN DD portion of the job control stream. Parameters may appear anywhere through column 71, provided they are separated by commas with no intervening blanks.

NRECS

$\text{NRECS}=\left\{\begin{array}{c} \underline{\text{ALL}} \\ \text{nnn} \end{array}\right\}$

Tells how many records to scan in the variable-length file.

WIDTH

$\text{WIDTH}=\left\{\begin{array}{c} \underline{20} \\ \text{nnnn} \end{array}\right\}$
--

Indicates the range between minimum and maximum block lengths and the minimum and maximum record lengths in each group of the HISTOGRM output. The number specified for the WIDTH value must be a multiple of 4. (4, 8, 12, . . . See examples of block and record HISTOGRMs that follow.) Adjust this range based on the characteristics of the file (the lengths of the shortest and longest record) and the desired length of HISTOGRM.

DEVWK

$\text{DEVWK}=\left\{\begin{array}{c} 2311 \\ 3330 \\ 3340 \\ \underline{3350} \\ 3375 \\ 3380 \\ 3390 \end{array}\right\}$

Tells the type of disk device that will be used for intermediate storage when the sort is run. Specify the device number if HISTOGRM is to calculate L6 and L7.

KEYL

KEYL = $\left\{ \begin{array}{l} \underline{20} \\ \text{nnnn} \end{array} \right\}$

Gives the end location of the last control field in the record. Specify a value for KEYL if HISTOGRM is to calculate L6 and L7.

BIGREC

BIGREC = $\left\{ \begin{array}{l} \underline{20} \\ \text{nnnn} \\ \text{MAX} \end{array} \right\}$

Specifies the maximum number of HIS025I messages that will be issued in a HISTOGRM execution. When HISTOGRM processes a large file, this message may be generated as often as once for each record in the file. BIGREC limits the number of HIS025I messages that will be issued in each execution. HISTOGRM processing continues, but no further messages are issued once the BIGREC value is reached.

BLOCK

$\left\{ \begin{array}{l} \underline{\text{BLOCK}} \\ \text{NOBLOCK} \end{array} \right\}$

Tells whether or not to print the graphic portion of the HISTOGRM for block length.

REC

$\left\{ \begin{array}{l} \underline{\text{REC}} \\ \text{NOREC} \end{array} \right\}$

Tells whether or not to print the graphic portion of the HISTOGRM for record length.

BIGSTOP

$\left\{ \begin{array}{l} \underline{\text{BIGSTOP}} \\ \text{NOBIGSTP} \end{array} \right\}$

Tells whether or not to terminate the HISTOGRM run if an RDW value greater than the DCB LRECL is encountered in the input file.

Job Control Language

The following example shows a sample execution of HISTOGRM.

1. SYSUT1 is the variable-length file to be scanned. Specify the DCB parameter if

```
//L6L7          JOB
//STEP1        EXEC      PGM=HISTOGRM
//STEPLIB      DD        DSN=HISTOGRM,DISP=SHR
//SYSUT1       DD        UNIT=3490,VOL=SER=000001,          1
//              DD        DSN=VLRECS,LABEL=(1,SL),
//              DD        DISP=OLD
//SYSPRINT     DD        SYSOUT=A                          2
//SYSIN        DD        *                                  3
                KEYL=50,DEVWK=3390,NOBLOCK,NOBIGSTP
/*
```

Figure 232. Sample JCL/Control Stream for HISTOGRM

SYSUT1 is a non-standard label tape.

2. SYSPRINT is the data set on which printed output will appear. The DCB (not illustrated) is: DCB=(LRECL=121,BLKSIZE=121,RECFM=F).
3. You may use DD DUMMY instead of SYSIN DD *. Specify //SYSIN DD DUMMY,DCB=(LRECL=80,RECFM=FB,BLKSIZE=80).

Executing HISTOGRM through an E15 Exit

It is possible to execute HISTOGRM during a sort by specifying an E15 exit in the MODS control statement and coding HISTE15 as the r value. This produces a printout of the HISTOGRM for Records at the conclusion of the job. (It is, however, not possible to get a printout of the HISTOGRM for Blocks when initiating HISTOGRM in this way.)

The following example shows a sample execution of HISTOGRM by an E15 exit during a sort.

```

//HISTSORT      JOB
//STEP2        EXEC      PGM=SYNCSORT
//SORTIN       DD        UNIT=3490,VOL=SER=000001,          1
//             DSN=VARDATA,LABEL=(1,SL),
//             DISP=OLD
//SORTOUT      DD        UNIT=3490,VOL=SER=000002,          2
//             DSN=SORTED.DATA,LABEL=(1,SL),
//             DISP=(,KEEP)
//SORTWK01     DD        UNIT=SYSDA,SPACE=(CYL,(20,10))      3
//SORTWK02     DD        UNIT=SYSDA,SPACE=(CYL,(20,10))
//SORTWK03     DD        UNIT=SYSDA,SPACE=(CYL,(20,10))
//SYSOUT       DD        SYSOUT=A                            4
//MODLIB       DD        DSN=SYS1.SYNCLIB,DISP=SHR           5
//SYSIN        DD        *                                   6
      SORT          FIELDS=(4,10,CH,A)
      MODS          E15=(HISTE15,7400,MODLIB,N)              7
//SYSPRINT     DD        SYSOUT=A                            8
//HISTIN       DD        *                                   9
      WIDTH=40
/*

```

Figure 233. Sample JCL/Control Stream for HISTOGRM Initiated by an E15 Exit

1. SORTIN is a DD statement for SyncSort. It contains the data set that will be analyzed and then sorted. The data set name is VARDATA, and it is found on the standard labeled tape with the volume serial number 000001. The data set is already in existence. If SORTIN is not a standard label tape, DCB parameters must be specified. Note that RECFM must be either V, VB, or VBS.
2. SORTOUT is a DD statement for SyncSort. It assigns the data set name SORTED.DATA to the output file, and specifies a 3490 tape unit with the volume serial number 000002. It is not yet in existence. The DCB parameters default to those of SORTIN.
3. SORTWK01, SORTWK02, and SORTWK03 are DD statements for SyncSort. They reserve 20 cylinders of primary space, 10 cylinders of secondary space on direct access devices for intermediate storage.
4. SYSOUT is a DD statement for SyncSort. It assigns the SyncSort messages to the output device associated with class A.
5. The MODLIB DD statement is used to define the partitioned data set in which the HISTE15 program resides; MODLIB is referenced in the MODS control statement. The data set name is SYS1.SYNCLIB, and the DISP shows the library may be shared.

6. The `SYSIN DD *` statement marks the beginning of the input stream that includes the sort control statements. The `SORT` control statement shows that one control field will be sorted on. It consists of bytes 4-13 of the record, contains character data, and is to be sorted in ascending order.

The `MODS` control statement must specify an `E15` exit as an exit-type parameter and give `HISTE15` as the exit routine name. `HISTE15` takes 5000 bytes of core storage and resides in the main SyncSort library referenced here by a `DD` statement named `MODLIB`. The routine does *not* require link-editing during sort execution.

7. `SYSPRINT` is the data set on which the printout from `HISTE15` appears. Its DCB is: `DCB=(LRECL=121,BLKSIZE=121,RECFM=F)`.
8. The `HISTIN DD` statement is optional. It is used to override any default values. The following DCB parameter must be specified: `DCB=(LRECL=80,RECFM=FB,BLKSIZE=80)`. (With `HISTIN DD *`, the DCB is not necessary.)

Defaults for HISTE15

NRECS=	ALL
WIDTH=	20
DEVWK=	The same <code>SORTWK</code> devices used in executing this sort.
KEYL=	End of key field furthest into record for this sort.
BIGREC=	0 (This cannot be overridden.)
NOBLOCK	(This cannot be overridden.)
REC	
NOBIGSTP	(This cannot be overridden.)

1	HIS010I	NUMBER OF RECORDS.....	952				
	HIS011I	TOTAL LENGTH OF ALL RECORDS....	93412				
	HIS012I	AVERAGE RECORD LENGTH.....	98				
	HIS016I	KEY LENGTH.....	50				
	HIS015I	AVERAGE SPACE PER RECORD - L6..	129				
	HIS014I	RECOMMENDED SEG. SIZE - L7.....	72				
	HIS017I	LINE WIDTH.....	20				
	HIS018I	LONGEST RECORD.....	155				
	HIS019I	SHORTEST RECORD.....	40				
	HIS005I	RECORDS TOO LONG.....	48				
	HIS023I	RECORDS TOO SHORT.....	41				
	HIS020I	DEVICE TYPE.....	3390				
2							
		RECORD					
		COUNT					
3	104*				40	59
	181*				60	79
	199*				80	99
	196*				100	119
	211*				120	139
	61*				140	159

Sample Contents of HISTOGRAM Output

1. HISTOGRAM informational messages for records are printed at the top of the report. For explanations, see individual messages in the message section which follows these examples.
2. RECORD COUNT gives the number of records falling within the minimum and maximum numbers shown as RECORD LENGTH. The range is the WIDTH value that has been specified.
3. The asterisks are the graphic representation of the number of records within this range of record lengths.

HISTOGRM Messages

HISnnnA	messages indicate a critical error condition. HISTOGRM terminates to allow you to correct the error(s) so that a successful program may be run.
HISnnnI	messages are informational or indicate a non-critical error. They are printed on the HISTOGRM output for blocks and records and contain statistical information inserted by HISTOGRM.
HIS001A	INVALID CONTROL CARD EXPLANATION: A blank control statement or an incomplete control parameter was found.
HIS002A	INVALID DATA ON CONTROL CARD EXPLANATION: An invalid control parameter was found.
HIS003A	EXPECTED CONTIN NOT FOUND EXPLANATION: A control statement continuation was indicated either by a non-blank character in column 72 or by a comma immediately after the last control field, but no continuation card image was found.
HIS004A	INVALID DCB OR ACB DATA EXPLANATION: The SYSUT1 data set was opened and one of three errors was detected: (1) LRECL was not specified, (2) BLKSIZE was not specified, (3) RECFM was not V, VB, or VBS, or (4) the data set is a VSAM RRDS. ACTION: Check for a missing DCB parameter if SYSUT1 is a non-standard label tape. If the file is a standard label tape or a disk file, one of the DCB subparameters may be missing, or the file is not a variable-length file.
HIS005I	RECORDS TOO LONG nnnn EXPLANATION: Records with lengths exceeding the length specified in the DCB were found. The nnnn represents the number of long records found. Long records have no effect on other HISTOGRM statistics.
HIS006A	INVALID SPAN CONTROL FIELD BLOCK nnnn LOGICAL RECORD nnnn {DATA SET # nnnn} EXPLANATION: The third byte of the four byte record descriptor word preceding a variable-length record does not contain a valid code X'00', X'01', X'10', X'11', or the code is inconsistent with the code of the previous segment. The block and record number being processed are included in the message text. The first 100 bytes of both current and previous segment along with their RDWs, follows

this message. DATA SET # will be the concatenation number within SYSUT1 if the input data set is concatenated.

- HIS007I** **NUMBER OF BLOCKS nnnn**
EXPLANATION: The total number of blocks read from the SYSUT1 data set is given on the HISTOGRM for blocks.
- HIS008I** **TOTAL LENGTH OF ALL BLOCKS nnnn**
EXPLANATION: The total length in bytes of all blocks read is given on the HISTOGRM for blocks.
- HIS009I** **AVERAGE BLOCK LENGTH nnnn**
EXPLANATION: The average block length of all blocks read is given on the HISTOGRM for blocks.
- HIS010I** **NUMBER OF RECORDS nnnn**
EXPLANATION: The total number of records read from the SYSUT1 data set is given on the HISTOGRM for records. The total will exclude any records with lengths greater than the length specified in the DCB.
- HIS011I** **TOTAL LENGTH OF ALL RECORDS nnnn**
EXPLANATION: The total length in bytes of all records read is given on the HISTOGRM for records.
- HIS012I** **AVERAGE RECORD LENGTH nnnn**
EXPLANATION: The total length of all records is divided by the number of records and the quotient is given on the HISTOGRM for records.
- HIS013I** **NUMBER OF SPANNED RECORDS**
EXPLANATION: The number of records contained within two or more blocks is given on the HISTOGRM for records.
- HIS014I** **RECOMMENDED SEG. SIZE - L7**
EXPLANATION: The recommended segment size is given on the HISTOGRM for records. Supply SyncSort with this value either through L7 in the PARM field of the EXEC statement or through l₇ in the LENGTH parameter of the RECORD control statement.
- Note:** If the recommended number is 0, the range of record lengths in the file was too wide to compute an optimal value. In this case, do not supply an L7.
- HIS015I** **AVERAGE SPACE PER RECORD - L6**
EXPLANATION: The average work space necessary for each record is given on the HISTOGRM for records. Supply SyncSort with this value either through L6 in the PARM field of the EXEC statement

or through l_6 in the LENGTH parameter of the RECORD control statement.

Note: If the recommended number is 0, the range of record lengths in the file was too wide to compute an optimal value. In this case, do not supply an L6.

HIS016I	KEY LENGTH nnnn EXPLANATION: The end location of the last control field in the record is given on the HISTOGRM for records.
HIS017I	LINE WIDTH nnnn EXPLANATION: The numeric interval between the minimum and maximum block/record length is given on the HISTOGRM for records.
HIS018I	LONGEST RECORD nnnn EXPLANATION: The length of the longest record read; that is the record containing the largest value in the record descriptor word.
HIS019A	INVALID DEVICE TYPE EXPLANATION: An invalid device type was specified on the control statement in the SYSIN data set.
HIS019I	SHORTEST RECORD nnnn EXPLANATION: The length of the shortest record read is given on the HISTOGRM for records.
HIS020I	DEVICE TYPE nnnnnn EXPLANATION: The type of intermediate storage device to be used for the sort is given on the HISTOGRM for records.
HIS021I	BLOCK PARAMETER IGNORED EXPLANATION: Information about blocks cannot be collected when running HISTE15 during a sort.
HIS022A	INPUT FILE IS EMPTY EXPLANATION: There are no records in the input file which are not longer than the data set's LRECL.
HIS023I	RECORDS TOO SHORT nnnn EXPLANATION: Records with lengths less than the KEYL value specified for the HISTOGRM execution were found. The nnnn is the number of short variable-length records in the file.
HIS024I	LRECL nnnnn,BLKSIZE nnnnn,RECFM xxx EXPLANATION: The logical record length, block size, and record

format of the input data set obtained from the SYSUT1 DCB after OPEN.

- HIS025A** **INVALID RDW/RECORD LENGTH BLOCK nnnn LOGICAL RECORD nnnn {DATA SET # nnnn}**
EXPLANATION: The RDW value of the current record is greater than the DCB LRECL and HISTOGRM has been requested to terminate (thru the BIGSTOP parameter). The block and record number are supplied in the message and the first 100 bytes of the record and the RDW follow the message. DATA SET # will be the concatenation number within SYSUT1, if the input data set is concatenated.
- HIS025I** **INVALID RDW/RECORD LENGTH BLOCK nnnn LOGICAL RECORD nnnn {DATA SET # nnnn}**
EXPLANATION: The RDW value of a record is greater than the DCB LRECL. Block number and record number are supplied in the message text, along with the concatenation number if SYSUT1 is concatenated.
- HIS026I** **INPUT DATA SET IS VSAM ... NO BLOCK STATISTICS GATHERED**
EXPLANATION: The input to HISTOGRM is a VSAM data set; therefore block statistics are not produced for this HISTOGRM execution.
- HIS027A** **SYSUT1 DD STATEMENT MISSING**
EXPLANATION: The input data set is absent; the HISTOGRM run has terminated.
- HIS028A** **VSAM LOGICAL ERROR nn**
EXPLANATION: An error occurred while reading a VSAM data set. For the definition of the error number, nn, consult one of the following IBM publications:
- *DFSMS/MVS Macro Instructions for Data Sets, SC26-4913*
- HIS029A** **VSAM OPEN ERROR nn**
EXPLANATION: An error occurred during an attempt to OPEN a VSAM file. For the definition of the error number, nn, consult one of the following IBM publications:
- *DFSMS/MVS Macro Instructions for Data Sets, SC26-4913*
- HIS030A** **message text**
EXPLANATION: An I/O error has occurred. The message text gives a detailed description of the error.

HIS031A

INVALID BDW ENCOUNTERED BLOCK nnnn

EXPLANATION: The block descriptor word for block number nnnn, was either zero or greater than the DCB blocksize.

Chapter 15. Value-Added Products

This chapter describes SyncSort's value-added products:

- Visual SyncSort for z/OS
- SyncSort/COBOL Advantage
- PROC SYNCSORT - An Accelerator for SAS™ Sorting
- PipeSort

These products significantly improve sorting efficiency and enhance programmer productivity.

Visual SyncSort

SyncSort for z/OS incorporates functionality to integrate Visual SyncSort with SyncSort for z/OS mainframe processing. Visual SyncSort is a separately available PC product that is designed to allow programmers and non-programmers alike to easily create and manage SyncSort applications for the mainframe environment. With Visual SyncSort, you can create new sort, merge, and copy applications, or you can import and modify existing ones. Visual SyncSort saves programmer time while taking full advantage of the processing power of SyncSort for z/OS.

Visual SyncSort does not require knowledge of sort syntax, so training time for new programmers is reduced dramatically. Buttons, pull-down menus, and other aids make navigation easy, and comprehensive context-sensitive Help is always available. Instant error

checking provides immediate feedback. Visual SyncSort generates applications that run correctly the first time, because they are always free of syntax errors.

Visual SyncSort's data dictionary and interactive design work together in a visual environment that greatly simplifies application development. Once you identify a field in your record layout by its position, length, and format and define a name for it (or import a COBOL file definition with the same information), you simply specify fields by name in your application. Visual SyncSort tracks the position and length of all fields, automatically handling changes in field specifications that occur as a result of output reformatting, data conversion, summarization, arithmetic manipulation, and report writing.

When you develop an application, Visual SyncSort leads you through a series of dialogs that make SyncSort's powerful features easily available. Based on your responses, Visual SyncSort builds the SyncSort for z/OS control data set for you, so you can concentrate on what you want to do, not how to do it. And Visual SyncSort checks the information you enter as you are building the application, eliminating the need for debugging runs.

Visual SyncSort makes it easy to develop reports because you type features like headers and trailers on your computer screen exactly as you want them in your report. You can import existing mainframe applications into Visual SyncSort to enjoy the benefits of modifying them through the Visual SyncSort interface. Visual SyncSort will automatically generate field names and lay out the entire application so you can modify and re-use it quickly and easily. Visual SyncSort automatically analyzes input and output specifications, and creates an optimized application that runs fast and minimizes SyncSort's use of system resources. For every Visual SyncSort application, you get a clearly laid out, consistently formatted application description, rather than cryptic control statements.

SyncSort/COBOL Advantage

The SyncSort/COBOL Advantage is a fully automatic facility for improving the performance of all COBOL programs that invoke SyncSort for z/OS. By enhancing and replacing COBOL sort interfaces, it will improve elapsed time by 25 to 40%. It will significantly reduce CPU time and EXCPs in addition to elapsed time in COBOL sorts with USING and/or GIVING clauses.

The SyncSort/COBOL Advantage is transparent to the user. It is invoked dynamically whenever SyncSort is invoked, and it automatically decides how to accelerate COBOL processing.

The SyncSort/COBOL Advantage is easy to install and requires no changes to existing programs or procedures. Programs do not have to be recompiled to benefit from the SyncSort/COBOL Advantage.

For more information regarding the benefits and installation of this product, refer to the *SyncSort/COBOL Advantage Installation and Use Guide*.

PROC SYNCSORT - An Accelerator for SAS™ Sorting

PROC SYNCSORT - An Accelerator for SAS™ Sorting is a high performance replacement for the SAS-provided procedure PROC SORT. Compared to PROC SORT, PROC SYNCSORT reduces the resources required for sorting within SAS applications and significantly cuts sort elapsed time.

Sort processing within SAS often consumes as much as 30 percent of CPU time and EXCPs. Because sorting is such a large part of system activity, PROC SYNCSORT's efficiency results in noticeable improvements in overall system throughput. This reduced elapsed time from PROC SYNCSORT makes it possible for SAS applications to complete much faster.

PROC SYNCSORT improves performance by providing a direct interface between SyncSort and SAS. This frees SyncSort to use its high performance techniques - sophisticated access methods, path length minimization algorithms and I/O optimization.

No modifications to SyncSort are required to install and use PROC SYNCSORT.

For more detailed information regarding the use and installation of PROC SYNCSORT, refer to the booklet titled *PROC SYNCSORT - An Accelerator for SAS™ Sorting: Installation and Use Guide*.

PipeSort

PipeSort works with SyncSort to run multiple sorts simultaneously on the same input data. For large input files, PipeSort significantly reduces total elapsed time compared to running separate sort jobs.

PipeSort reads SORTIN once and distributes the input records to up to eight simultaneous SyncSort executions. The complete range of SyncSort control statements and PARMs is available for the individual sort operations.

The output files are differently sequenced according to user-specified sort keys and are written to different SRTnOUT DD data sets.

Optionally, you can use an inline E15 exit, with or without one or more E35 exits. An inline E15 input exit can supply the input data to PipeSort, and E35 output exits can accept the different output record sets.

For detailed information regarding installation and implementation through z/OS JCL, refer to the *PipeSort User's Guide*.

Chapter 16. Messages

All messages issued by SyncSort have the form:

WERnnnx Message text

where *nnn* is the message number and *x* may be any of the letters A through I. The interpretation of the suffix letter *x* is given below.

A (action) message indicates a critical error condition: SyncSort terminates in order to allow the user to correct the error(s) so that a successful sort/merge may be run.

Example **WER002A EXCESS CARDS**

B (tuning) messages provide information that may be useful in adjusting the job/control stream to the actual demands of the job. These messages only print if a critical error forces sort termination or if B messages were requested at execution or installation time.

Example **WER151B SECONDARY EXTENTS OBTAINED xxx**

C-I (informational) messages document decisions internal to the sort as well as SyncSort's response to error conditions which are not severe enough to warrant sort/merge termination.

Example **WER177I TURNAROUND SORT PERFORMED**
WER185I SORTIN DCBBLK GT ACTUAL, I/O INEFF

SyncSort for z/OS provides an interactive message explanation facility, SS11MSG, that gives online access to all SyncSort message texts and their explanations by message number. If SS11MSG is included as an option in a PDF menu, you may invoke from that menu. Otherwise, you may invoke SS11MSG from the command line of any ISPF panel by entering the following command:

```
TSO %SS11MSG
```

The installation of the SS11MSG facility is optional. Therefore, if you are unable to invoke the facility as described, you should contact your system administrator for more information.

Note: All messages that refer to SORTIN, SORTWK, SORTOF and SORTOUT provide the actual DD name, which reflects any changes made via a prefix override.

WER001A COL 1 OR 1-15 NOT BLANK

EXPLANATION: This message is triggered by a character in column 1 of the END control statement or in columns 1-15 of a continuation statement following a statement with a character in column 72, or by a non-blank character in columns 1-15 of a sort control statement in the \$ORTPARM data set. These columns must be left blank.

WER002A EXCESS CARDS

EXPLANATION: The static internal storage area is inadequate for the quantity and/or complexity of the control statements in this application. Either the minimum storage value set at installation time is too low, or insufficient storage is available in your region.

ACTION: Ask the systems programmer in charge of SyncSort installation to increase the minimum storage (MINCORE) value unless the storage available in the region is less than the minimum storage value. In that case, increase the storage available in the region or partition so that it at least equals the minimum storage value.

WER003A NO CONTIN CARD

EXPLANATION: A continuation statement was indicated by a non-blank character in column 72 of a control statement or by a control statement ending in an operand-comma combination; no continuation statement followed.

WER004A	INVALID OP DELIMITER
	EXPLANATION: A control statement ends with an incorrect delimiter, such as a comma.
WER005A	STMT DEFINER ERR
	EXPLANATION: A control statement contains an invalid operation name. (Tape Sort accepts fewer operation names than Disk Sort, MAX-SORT and PARASORT.)
WER006A	OP DEFINER ERR
	EXPLANATION: An operation name must be followed by an operand on the same control statement.
WER007A	SYNTAX ERROR - {S/M,REC,MOD}
	EXPLANATION: The control statement shown in the message contains a syntax error.
WER008A	FLD OR VALUE GT 8 CHAR - {S/M,REC,MOD}
	EXPLANATION: A parameter exceeds eight characters on the statement shown in the message.
WER009I	EXCESS INFO ON CARD - {S/M, REC,MOD}
	EXPLANATION: The control statement shown in the message contains more information than necessary. This excess information will be treated as a comment.
WER010A	NO S/M CARD
	EXPLANATION: The SORT/MERGE control statement is required for all SyncSort applications including a straight copy (coded SORT FIELDS=COPY or MERGE FIELDS=COPY).
WER011A	TOO MANY S/M KEYWORDS
	EXPLANATION: The maximum of eight keyword operands for tape sort was exceeded on a control statement.
WER012A	NO FLD DEFINER
	EXPLANATION: The FIELDS operand was not specified on the SORT/MERGE control statement.

WER013A	INVALID S/M KEYWORD
	EXPLANATION: An invalid keyword operand was found on the SORT/MERGE control statement.
WER014A	DUPLICATE S/M KEYWORD
	EXPLANATION: A keyword operand was defined more than once on the SORT/MERGE control statement.
WER015A	TOO MANY PARAMETERS
	EXPLANATION: A keyword operand on the SORT/MERGE control statement contains too many parameters.
WER016A	INVALID VALUES IN FLD
	EXPLANATION: An invalid value was specified in the FIELDS operand on the SORT/MERGE control statement.
WER017A	ERR IN DISP LENGTH VALUE
	EXPLANATION: The length and displacement value of a control field is greater than 4092 (4084 for variable-length records), or less than one, or the sum of the lengths of all control fields exceeds 4092 (4084 for variable-length records).
WER018A	CTL FLD ERR
	EXPLANATION: An error was detected in the SORT/MERGE control statement for the data format of a control field. The format was specified for one field but not for another, or bit comparisons were specified and FORMAT=BI was not specified.
WER019A	SIZE/SKIPREC ERR
	EXPLANATION: The SIZE or SKIPREC parameter was incorrectly specified.
WER020A	INVALID REC KEYWORD
	EXPLANATION: An invalid keyword was detected in a RECORD control statement.

WER021A	NO TYPE DEFINER
	EXPLANATION: The TYPE operand must be specified in the RECORD control statement.
WER022A	RCD FORMAT NOT F/V
	EXPLANATION: An invalid RECFM was detected for SORTIN or SORTOUT.
WER023A	NO LENGTH DEFINER
	EXPLANATION: The LENGTH operand must be specified on the RECORD control statement.
WER024A	ERR IN LENGTH VALUE
	EXPLANATION: An invalid LENGTH subparameter was found in a RECORD control statement. For example, l5 was greater than l1 or l4 was greater than l2.
WER025A	RCD SIZE GT MAX
	EXPLANATION: The LENGTH operand in a RECORD control statement specified a value which is greater than the maximum value allowed (32,767).
WER026A	L1 NOT GIVEN
	EXPLANATION: The LENGTH operand on a RECORD control statement does not contain an l1 value.
WER027A	CONTROL FIELD BEYOND RECORD
	EXPLANATION: The last byte of a SORT/MERGE control field is located beyond the maximum record length specified, or a variable-length record is shorter than the ending location of a specified control field in an execution for which this is defined as cause for SyncSort termination (see VLTEST). Program HISTOGRM may be used to determine the length of the shortest record in the input file.
WER028A	TOO MANY EXITS
	EXPLANATION: More than the maximum number of exits allowed (16) were specified.

WER029A

IMPROPER EXIT

EXPLANATION: The set of legal exits depends on the sorting technique chosen. Tape Sort does not support E14 or E32; a merge or copy may not specify any Phase 1 or Phase 2 exits; a copy may not specify exit E32 or E61; and a sort or merge with data fields of Y2x or PD0 formats may not specify exit E61.

WER030A

MULTIPLY DEFINED EXIT

EXPLANATION: The same program exit was specified twice on a MODS control statement.

WER031A

INVALID MODS OP CHAR

EXPLANATION: An invalid character was detected in a parameter on a MODS control statement.

WER032A

EXIT E61 REQUIRED

EXPLANATION: A SORT control statement specified "E" in the FIELDS parameter but program exit E61 was not specified on a MODS control statement.

WER033A

CONTROL FIELD COLLATING ORDER E REQUIRED

EXPLANATION: Program exit E61 was specified on the MODS control statement but "E" was not specified in the FIELDS parameter of the SORT control statement.

WER034A

PARAM ERR FOR MODS

EXPLANATION: An incorrect number of parameters was given for an operand on a MODS control statement, or SYSIN was specified on a MODS control statement as the source for user exit routines but a // SORTMODS or //SYSIN statement is missing or dummy.

WER035A

DUPLICATE MOD RTN IN PHASE

EXPLANATION: The same exit routine was used for more than one program exit in the same phase of a tape sort, or two or more exit routines in a tape sort have the same name.

WER036B

G=ggg, B=bbb, SEGLN=sss, BIAS=zz

EXPLANATION: The tuning information displayed is as follows:

G=ggg *ggg* is the number of records that can be contained in SyncSort's working virtual storage area. For variable-length records, this number is the number of segments.

B=bbb *bbb* indicates the physical blocking used for intermediate storage. For fixed-length records, this number represents the blocking factor. For variable-length records, it represents the blocksize. The B value will not appear in the message for incore or turnaround sorts.

SEGLN=sss This value appears in the message for variable-length records, when the execution is not an incore or turnaround sort. It reflects the segment length used in SyncSort's working storage during Phase 1.

BIAS=zz *zz* reflects the degree of prior sequencing in the input data. The number displayed ranges from 00 to 99 indicating random to highly sequenced input. The BIAS value is not included in the message for an incore or turnaround sort, where it is 100 by definition.

WER037A

REXX ENVIRONMENT UNAVAILABLE

EXPLANATION: One or more REXX exits were specified in a MODS control statement, but the required operating system and/or TSO environment is not available.

WER039A

INSUFFICIENT VIRTUAL STORAGE

EXPLANATION: The amount of virtual storage available to SyncSort is not large enough to permit execution. Refer to the "Setting CORE" section of the Performance Considerations chapter for further information.

ACTION: Verify that virtual storage is specified properly. Check that the region size is sufficient for execution.

WER040A

INSUFFICIENT WORK UNITS

EXPLANATION: Tape Sort requires at least three work devices, numbered in order.

WER041A

N GT MAX

EXPLANATION: The number of records specified in the SIZE parameter on a SORT control statement is greater than the NMAX value calculated by SyncSort.

ACTION: Check SIZE parameter for error. If there is no error, increase intermediate storage.

WER042A UNITS ASSGN ERROR

EXPLANATION: The device type of a SORTWK data set is not valid.

WER043A DATA SET ATTRIBUTES NOT SPECIFIED

EXPLANATION: The DCB parameter was not specified for a SORTIN or SORTOUT data set.

ACTION: Specify the DCB parameters. When including an E18 exit routine, this is necessary even when SORTIN is a standard labeled tape.

WER044A EXIT E_{xx} INVALID OPTION

EXPLANATION: The exit routine shown in the message specified an invalid option for the modification of a DCB parameter of a sort/merge data set.

WER045C END SORT PH

EXPLANATION: SyncSort has completed its sort phase.

WER046A SORT CAPACITY EXCEEDED

EXPLANATION: All available intermediate storage is exhausted, including any secondary allocation allowed in this job set. Sort processing cannot continue.

ACTION: Supply more intermediate storage (see the SORTWK calculation formula) or use the MAXSORT technique.

WER047A RCD CNT OFF, IN x, OUT y

EXPLANATION: The actual number of records specified in the SIZE parameter on the SORT control statement (the IN value) was not equal to the number of records read from the SORTIN data set (the OUT value). (This comparison is made only when the SIZE parameter specifies an *actual* number of records.)

For Disk Sort, MAXSORT and PARASORT: the actual number of records (the IN value) for FILSZ=n specified either on the SORT control statement or as a PARM option was not equal to the total number of records (the OUT value) from the SORTIN data set after any changes

due to the INCLUDE/OMIT control statement, and an E14 or E15 exit routine, and SKIPREC and/or STOPAFT processing.

For Tape Sort only: there is a discrepancy between the number of records entering and leaving a phase which is not accounted for by user exits (this is usually caused by an I/O error - try running the sort again).

WER048I E16 EXIT CALLED

EXPLANATION: Program exit E16 was entered after all available SORTWORK space was exhausted.

WER049A SUM FIELD OVERFLOW

EXPLANATION: Summarization of two equally keyed records could not be done due to a numeric overflow or underflow in a defined SUM field.

ACTION: Investigate the use of the INREC control statement if possible to pad the fields with leading zeros of the proper numeric format. Adjust the SUM field and other control statement fields accordingly.

Visual SyncSort users may create an Advanced Field that could be substituted for the SUM field by using the Summation Field Expansion option.

WER049I SUM FIELD OVERFLOW

EXPLANATION: Summarization of two equally keyed records could not be done due to a numeric overflow or underflow in a defined SUM field.

ACTION: If complete summarization is desired, use the INREC control statement if possible to pad the fields with leading zeros of the proper numeric format. Adjust the SUM field and other control statement fields accordingly.

Visual SyncSort users may create an Advanced Field that could be substituted for the SUM field by using the Summation Field Expansion option.

WER050I SUM CONTROL STATEMENT IGNORED

EXPLANATION: A SUM control statement was specified in a SORT FIELDS=COPY application. Since a COPY operation does not use

SORT/MERGE key fields, the specification of SUM, which operates on equally keyed records, is illogical.

ACTION: The SUM statement will be ignored, but the application should be checked for correct specification of control statements.

WER051A UNENDING MERGE

EXPLANATION: The intermediate storage provided is insufficient to complete the intermediate merge phase.

ACTION: Assign more tape work units to the sort.

**WER052I END SYNC SORT - jobname, stepname, procstepname,
DIAG=hhhh,hhhh,...**

EXPLANATION: SyncSort has successfully completed execution. The hexadecimal information following the DIAG keyword is likely to change from execution to execution. It is internal diagnostic information intended for use by SyncSort personnel in Product Support.

WER053A OUT OF SEQ

EXPLANATION: The current record leaving the final merge phase is not in collating sequence with the last record blocked for output.

WER054I RCD IN x, OUT y

EXPLANATION: The *x* represents the number of records read from the input data set(s). If OUTFIL statements are not present, the *y* represents the number of records in the output file. If OUTFIL statements are present, the *y* represents the number of records available for OUTFIL processing.

WER055I INSERT x, DELETE y

EXPLANATION: The *x* represents the number of records inserted by user exit routines. The *y* represents the number of records deleted by user exit routines, SUM, and INCLUDE/OMIT control statements.

Note: For MAXSORT, these counts are cumulative for the entire MAXSORT application.

WER056A SORTIN/SORTOUT NOT DEFINED

EXPLANATION: A required SORTIN or SORTOUT DD statement could not be found.

WER057A

SORTIN NOT SORTWK01

EXPLANATION: A tape work data set other than SORTWK01 is assigned to the same tape unit assigned to SORTIN.

WER058A

SORTOUT A WORK UNIT

EXPLANATION: A tape work data set is assigned to the same tape unit assigned to SORTOUT.

WER059A

RCD LNG INVALID FOR DEVICE

EXPLANATION: The logical record length specified for a fixed-length input data set plus overhead, if any, is too large to fit on one disk track of the intermediate storage device, or (Tape Sort only) it is less than 18 bytes long.

WER060A

DSCB NOT DEFINED

EXPLANATION: There are no DD names in the TIOT for any of the sort work data sets.

WER061A

I/O ERR jobname, stepname, unit address, device type, DDname, operation attempted, error description, last seek address or block count, access method.

EXPLANATION: An I/O error has occurred on the device whose address is given. I/O errors are often transient - resubmitting the job may result in a successful run. However, if the I/O error is on SORTIN, the following should be checked first:

1. When SORTIN consists of concatenated data sets, check that the largest blocksize is available at sort initialization. See "Concatenating Input Data Sets".
2. If the data set is on disk and has just been created by another program, check that this program opened the data set *even if no data was written to the file*. The data set must be opened in order for an end-of-file mark to be written. (In the absence of an end-of-file mark, SyncSort will tend to read whatever was on the disk as part of SORTIN, causing an I/O error.)

WER062A

L E ERR

EXPLANATION: The linkage editor detects an error so serious that execution cannot continue.

- WER063A xxxxxx OPEN ERR**
- EXPLANATION: The data set shown cannot be successfully opened.
- ACTION: Check for missing DD statements.
- WER064A DELETE ERR**
- EXPLANATION: A SyncSort program module is unable to delete itself or a user exit routine.
- WER065A DECK STRUCTURE ERROR**
- EXPLANATION: The end of the SYSIN data set was reached before all user exit routines were read or an object deck was missing its first statement.
- WER066A APPROX RCD CNT x**
- EXPLANATION: Sort capacity was exceeded, so the sort terminated. The approximate number of records processed by SyncSort up to this point is given.
- WER067I INVALID EXEC OR ATTACH PARAMETER**
- EXPLANATION: An invalid parameter was detected in the PARM field of the EXEC statement or in the parameter list if SyncSort was initiated through ATTACH, LINK, or XCTL. Invalid parameters are ignored. (If a parameter is entered more than once, the last entry, if valid, is used.)
- WER068A OUT OF SEQ SORTINxx[, BLOCK y]**
- EXPLANATION: A record in the SORTIN data set indicated by xx is out of sequence according to the FIELDS specification on the MERGE statement. The number y of the block containing the out-of-sequence record is given if an E32 exit was not used.
- WER069A E39/OUTFIL INCOMPATIBLE**
- EXPLANATION: The E39 exit facility may not be utilized in sorts/merges which also specify OUTFIL control statements.
- WER070A ddname {TOTAL,SUBTOTAL,AVG,SUBAVG} FIELD OVERFLOW**
- EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter.

An overflow condition was generated during the TOTAL, SUBTOTAL, AVG, or SUBAVG OUTFIL function for the specified output file. All fields are totaled internally as 8-byte PD values, allowing for 15 decimal digits.

ACTION: This error is usually due to invalid data in the specified fields. Check the fields specified in the indicated parameter and check the actual data in those fields to ensure that no total will exceed 15 decimal digits. In some cases, within TRAILER2 or TRAILER3, you could change SUBTOTAL to TOTAL or SUBAVG to AVG to reduce the possibility of overflow.

WER071A MAXIMUM NUMBER OF RECORDS EXCEEDED

EXPLANATION: SyncSort's default internal limit on the maximum number of records that can be sorted has been exceeded. By default, the internal limit on the number of records that can be processed for variable-length data or for a sort application that uses the EQUALS option is 4,294,967,295 records. Specify the EXTCOUNT PARM to increase the internal limit to 140,737,488,355,327 records. Fixed-length sorts without EQUALS have automatic support for the maximum number of records allowed by the EXTCOUNT PARM. For additional information, see the EXTCOUNT option in the "PARM Options" chapter.

WER100A DUPLICATE RECORD KEYWORD

EXPLANATION: A keyword operand was defined twice on a RECORD control statement.

WER101D INVALID TAPE TYPE IN PARM FIELD

EXPLANATION: An invalid tape type was specified for DEVIN/DEVOUT in the PARM field of the EXEC statement. SyncSort ignored the invalid parameter.

WER102A COBEXIT=COB2 AND COBOL E15 AND E35 EXITS FOUND IN COPY APPLICATION

EXPLANATION: A COBOL E15 and COBOL E35 may not both be specified in a copy application if COBEXIT=COB2 is in effect. Only one of the exits is permitted.

WER103D INVALID MESSAGE TYPE IN PARM FIELD

EXPLANATION: An invalid message code was specified in the PARM field of the EXEC statement or in the invoking program parameter list. SyncSort ignored the invalid parameter.

WER104A REXX E15 AND REXX E35 EXITS FOUND IN A COPY APPLICATION

EXPLANATION: A REXX E15 and a REXX E35 may not both be specified in a copy application. Only one of the exits is permitted.

WER105A INCOMPATIBLE LEVELS BETWEEN THE STATIC AND DYNAMIC LIBRARIES OF A COBOL OR C EXIT

EXPLANATION: When using either a C or COBOL exit, insure that the run-time dynamic Language Environment libraries are at the same or higher level than the libraries used for the compile or link-edit of the exit.

WER106A ddname INVALID DEVICE TYPE

EXPLANATION: The ddname is either SORTIN, SORTINnn, SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. This file resides on an invalid device type. Valid device types include the IBM 2305, 2314, 3330, 3330-11, 3340, 3344, 3350, 3375, 3380, 3390, and 9345 direct access devices and 3850 mass storage systems, and their equivalent as well as the IBM 2400, 3400, 3480, 3490 and 3590 series tape devices and their equivalents.

WER107A ddname RECFM INCOMPATIBLE WITH REPORT WRITING

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The RECFM specified for the file did not include the 'A' (ASA control character) specification that is required when report writing is requested.

WER108I SORTIN: RECFM= ;LRECL= {;BLKSIZE=,CISIZE=} [;CINV ACCESS]

EXPLANATION: This informational message lists the DCB characteristics used by SyncSort to process the SORTIN file. For a non-VSAM data set that is concatenated, the DCB characteristics are for the first of the concatenated data sets, except for BLKSIZE, which is the largest of all data sets in the concatenation examined at sort initialization time. For a VSAM data set, the CISIZE is provided; if control interval access was used, the CINV ACCESS portion of the message will be displayed.

WER109I

MERGE INPUT: TYPE={F,V};LRECL=

EXPLANATION: This informational message lists the DCB characteristics used by SyncSort to process the input files for a merge.

WER110I

ddname RECFM= ;LRECL= {;BLKSIZE=,CISIZE=} [;CINV ACCESS]

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. This informational message lists the DCB characteristics used by SyncSort to process the indicated output file. This message will be provided for each output file specified. For a VSAM data set, the CISIZE is provided; if control interval access was used, the CINV ACCESS portion of the message will be displayed.

WER111A

[ddname] {INREC,OUTREC,TOTAL/SUBTOTAL,MIN/SUBMIN, MAX/SUBMAX,AVG/SUBAVG} INVALID DATA CONVERSION REQUESTED

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. Data conversion has been requested for INREC, OUTREC, OUTFIL OUTREC, TOTAL/SUBTOTAL, etc., as indicated, and one of the following error conditions has occurred:

1. The length of the field to be converted is too large.
2. Data conversion has been requested for a field that is not specified as BI, CSF/FS, FI, PD, Y2ID, Y2IP or ZD.
3. Illegal or conflicting EDIT/SIGNS parameters were specified.

WER112A

INVALID VALUES IN FIELD PARAMETER

EXPLANATION: An invalid value was specified in the FIELDS operand of the SORT/MERGE control statement.

WER113A

TOO MANY SORT FIELDS

EXPLANATION: The number of sort control fields specified exceeds the internal limits of the product. The absolute upper limit on the number of sort control fields is 128; however, depending on the complexity of an application, the limit may be reduced. When locale processing is used, the number of allowable CH control fields is also limited by the length of those fields.

- WER115A** **ILLEGAL MOD NAME**
- EXPLANATION: An invalid name for a program exit was entered on a MODS control statement.
- WER116A** **THE FOLLOWING H/T IS GT LRECL:**
- EXPLANATION: This message flags any HEADERS or TRAILERS that exceed the LRECL specification. The HEADER or TRAILER in error will be printed on the next line.
- WER117A** **INVALID ANSI CONTROL CHARACTER FOUND**
- EXPLANATION: An invalid ANSI control character appears in a HEADER or TRAILER. The ANSI Control Character Table lists the valid characters accepted by SyncSort.
- WER117I** **INVALID ANSI CONTROL CHARACTER FOUND**
- EXPLANATION: An invalid ANSI control character appears in an output data record. The sort will process the record as if a blank control character had been found. This message will be issued only once regardless of how many data records have invalid ANSI characters. The ANSI Control Character Table lists the valid characters accepted by SyncSort.
- WER118A** **ddname ILLEGAL OVERLAPPING FIELDS**
- EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an UTFIL FNAMES parameter. An UTFIL control statement contains a HEADER or TRAILER parameter which contains overlapping fields. This may be caused, for example, by a positional sub-parameter specification which overlaps a previously defined field.
- WER119A** **NO DD NAME IN MODS FIELD**
- EXPLANATION: A DD name is missing on the MODS control statement.
- WER120A** **SEP. LKED NOT ALLOWED**
- EXPLANATION: A module for which separate link-editing was specified on a MODS control statement is not allowed to be link-edited separately.

WER121A

TASK CALL PARAM ERROR

EXPLANATION: If a 24-bit list is being used, either a control statement address is zero, or the length of a control statement is not positive, or the parameter list ends with the first word of a two-word parameter. If a 31-bit list is being used, the last parameter word in the list is not followed by the four byte field X'FFFFFFFF'.

WER122A

INVALID INTERMEDIATE STORAGE DEVICE

EXPLANATION: An invalid device was assigned as intermediate storage. Valid devices include IBM's 2305, 2314, 3330, 3330-11, 3340, 3344, 3350, 3375, 3380, 3390, and 9345 mass storage system, and equivalent units. A tape device may not be used unless Tape Sort is installed.

WER123A

IMPROPER RETURN CODE FROM E_{xx}

EXPLANATION: An invalid return code was passed by the exit that appears in the message. Valid return codes are 0, 4, 8, 12, 16 (and 20, if the exit is a COBOL or C E15 or E35).

WER124I

**[ESTIMATED] PREALLOCATED/USED SORTWORK SPACE
USAGE FACTOR {=,<,>}nn.nn**

EXPLANATION: *nn.nn* represents the quotient obtained by dividing the number of tracks assigned within preallocated sortworks (sortworks allocated in the JCL or dynamically allocated by an invoking program) by the number of tracks actually used by SyncSort. The word ESTIMATED is included in when SyncSort's derivation of this factor is inexact, for example, when all sortwork data sets are not opened, or when data space or hiperspace are used to contain part or all of the sortwork data.

Note that for MAXSORTs, the factor displayed is at or near "1.00" for all but the last sort. For the last sort, the factor may be anywhere between "0.01" and "1.00" depending on the amount of data sorted.

WER125A

NO DATA ON MODS CARD

EXPLANATION: The MODS control statement contains no parameters.

WER128A

INVALID CARD BEFORE END CARD

EXPLANATION: A control statement containing an error was found.

- WER130A** **I/O ERROR ON SYSIN**
- EXPLANATION: An I/O error occurred on SYSIN.
- WER131I** **PARM FIELD ERROR - xxxxxxxx**
- EXPLANATION: An invalid character was found in the PARM shown in the message. SyncSort ignored the remainder of the PARM and continued processing.
- WER133A** **Exx USER EXIT RETURNED CODE TERMINATE**
- EXPLANATION: Return code 16 was passed by the exit routine shown in the message. SyncSort terminated.
- WER135A** **TASK CALL/E35 TERMINATED PREMATURELY**
- EXPLANATION: An E35 exit routine (COBOL Output Procedure) passed a return code of 8, terminating the sort before the sort was able to pass all of the records. A SORTOUT data set was not present.
- WER135I** **TASK CALL/E35 TERMINATED PREMATURELY**
- EXPLANATION: An E35 exit routine (COBOL Output Procedure) passed a return code of 8, terminating the sort before the sort was able to pass all of the records. A SORTOUT data set was not present.
- This message may not indicate an error condition - it depends on what the programmer intended. For example, this message will be generated if a COBOL program using the SORT verb RELEASEs 100 records in the Input Procedure without RETURNing all 100 records in the Output Procedure because the logic dropped to the bottom of the Output Procedure "prematurely". If this is what the programmer intended, then no data has been lost. If, however, the programmer intended the Output Procedure to write all the records read in the Input Procedure, then this message indicates a logic bug in the COBOL program.
- WER136A** **{INREC,OUTREC,ddname OUTREC} HAS OVERLAPPING
FIELDS SPECIFIED**
- EXPLANATION: The column specification of a c: sub-parameter in the indicated control statement overlaps a field previously defined in the same control statement. Note that the sub-parameters used to define each field must be coded in the order in which the fields will appear in the reformatted record. The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter.

- WER138A** **ddname BLKSIZE NOT EVENLY DIVISIBLE BY LRECL**
- EXPLANATION: The ddname is either SORTIN, SORTINnn, SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. A block was read from the indicated file whose length was not a multiple of the LRECL value, or the JCL or data set attributes are incorrect.
- WER141A** **ddname RECFM IS U**
- EXPLANATION: The ddname is either SORTIN, SORTINnn, SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. SyncSort does not support undefined record format for any of these files.
- WER142A** **MIXED SORTIN TYPES F/V NOT SUPPORTED**
- EXPLANATION: SyncSort permits only one record format type (fixed or variable) for input files per sort/merge.
- WER143A** **SORTIN LRECLS ARE MIXED**
- EXPLANATION: The LRECL must be the same for all fixed-length files supplied to a merge.
- WER144B** **UNEXPECTED VIRTUAL STORAGE FRAGMENTATION**
- EXPLANATION: The amount of virtual storage calculated by SyncSort for Phases 2 or 3 was not available in a contiguous block. Additional virtual storage was obtained to satisfy the sort requirement. This condition was probably caused by virtual storage not released by the user program in the job step (for example, user exit buffer space was not released).
- WER146B** **nnn BYTES OF EMERGENCY SPACE**
- EXPLANATION: The indicated amount of virtual storage has been set aside by SyncSort for use by other programs (e.g., program invoking the sort, system SVCs, tape management system).
- WER147I** **CONTROL FIELD GT REC LEN, POSSIBLE OUT OF SEQ REC**
- EXPLANATION: The sort encountered a variable-length record that was too short to contain all of the control field(s) specified in the SORT statement. VLTEST instructed SyncSort to pad the record with binary zeros to the length of the sort key and continue processing. The added binary zeros account for the position of this record in the sorted file,

which may appear to be out of sequence for this reason. The binary zeros are removed when the record is processed for output. Program HISTOGRM may be used to determine the length of the shortest record in the input file.

WER148A OPEN ERR SYSIN

EXPLANATION: SYSIN is either not present or cannot be opened.

WER149B FRAGMENTED VIRTUAL STORAGE IN SORT PHASE

EXPLANATION: The virtual storage specified for SyncSort's use was not available in a contiguous block for Phase 1. This condition was probably caused by a calling program or user exit routine. SyncSort obtained its virtual storage in fragments and continued execution. Note that the calling program or user exit routine used virtual storage in such a way as to cause fragmentation, which might another time result in ABEND 80A or S804.

WER151B SECONDARY EXTENTS OBTAINED xxx

EXPLANATION: This gives the number of secondary extents obtained for SORTWKxx data sets.

**WER152B REQUESTED VIRTUAL STORAGE NOT AVAILABLE, nnn
BYTES USED**

EXPLANATION: The CORE parameter specified a value which was not available when SyncSort received control. The number of available bytes used by SyncSort is given.

**WER153A INSUFFICIENT VIRTUAL STORAGE IN {INT.,FINAL} MERGE
PHASE**

EXPLANATION: The amount of virtual storage available for the indicated merge phase (the intermediate or final merge phase) was not sufficient to allow execution. Refer to the "Setting CORE" section of the Performance Considerations chapter for further information.

WER154A NO MODS DD CARD

EXPLANATION: The DD statement whose name was specified on the MODS control statement was not provided, so the user exit routine cannot be found.

WER157A**SPANNED REC. LEN LARGER THAN LRECL/L2**

EXPLANATION: A record from a VBS SORTIN data set contains a record longer than the maximum record length specified by LRECL in the DCB.

ACTION: Execute program HISTOGRM to get the length of the longest record in the data set. Use this length for the LRECL value in the DCB parameter of the SORTIN data set.

WER158I**REC. LEN GT L2, CUT TO L2**

EXPLANATION: A variable-length record read from the SORTIN data set is longer than the maximum record length specified by either LRECL in the DCB or the l_2 value in the RECORD control statement. (If l_2 was not specified, the variable-length record is longer than the l_1 value.) SyncSort has truncated the record.

ACTION: If truncation is not desired, execute program HISTOGRM to get the length of the longest record in the data set. Use this length for the LRECL value in the DCB parameter of the SORTIN data set.

WER159A**REC LEN 0, {SORTIN REC x, INSERTED REC x}**

EXPLANATION: A bad variable-length record (length code <4 in its Record Descriptor Word) has been found. If the record was found in the SORTIN file, the number of the bad record is given. If the record was inserted from a user input exit routine, the number of the inserted record is given. (E.g., 45 indicates the forty-fifth record read from or inserted into the SORTIN file.)

WER160A**REC. LEN GT LRECL/L2, USER REQ ABORT**

EXPLANATION: VLTEST has requested the sort to abort because of the following condition. A variable-length record read from the SORTIN file is longer than the maximum record length specified by LRECL in the DCB or (after E15 processing) is longer than the l_2 value in the RECORD control statement. (If l_2 was not specified, the l_1 value was used as its default.)

ACTION: Change the LRECL or l_2 value to reflect the record length, or specify another value for VLTEST. Program HISTOGRM may be used to determine the length of the longest record in the input file.

WER161B ALTERNATE PARM USED

EXPLANATION: The alternate PARM option (\$ORTPARM DD, PARMEXIT or PARMTABLE) was used and SyncSort received the parameters specified.

**WER162B ppp PREALLOCATED SORTWORK TRACKS, ddd DYNAMI-
CALLY ALLOCATED sss ACQUIRED IN xxx SECONDARY
EXTENTS, rrr RELEASED, TOTAL OF uuu TRACKS USED**

EXPLANATION: *ppp* is the number of tracks found available in sortwork data sets which were allocated prior to SyncSort's gaining control. (These may have been allocated in the JCL or dynamically allocated by an invoking program.) *ddd* is the number of tracks dynamically allocated as primary space by SyncSort. *sss* is the number of tracks acquired as secondary space, on both preallocated data sets and data sets dynamically allocated by SyncSort. *xxx* is the total number of secondary extents acquired. *rrr* is the total number of unneeded tracks released from both preallocated data sets and data sets dynamically allocated by SyncSort. *uuu* is the total number of tracks actually used in sorting.

The following notes apply to the information in this message:

- *ppp* may not represent all of the preallocated tracks available, since not all preallocated sortwork data sets may be opened by SyncSort.
- *uuu* may be less than the sum of *ppp*, *ddd* and *sss* since it represents the space actually used and not the space available.
- For MAXSORTs, all dynamic allocation and secondary space acquisition is done during the first sort. For this reason, the WER162B message for the first sort will indicate the number of tracks dynamically allocated, the number acquired via secondary extents, etc. However, the WER162B message in all subsequent MAXSORT sorts will report these tracks as "preallocated."

**WER164B www BYTES OF VIRTUAL STORAGE AVAILABLE, xxx BYTES
REQUESTED, yyy BYTES RESERVE REQUESTED, zzz BYTES
USED**

EXPLANATION: The amount of virtual storage available (free) when SyncSort received control is represented by w's. The amount of virtual storage requested for SyncSort's use is represented by x's. The amount of virtual storage that the user requested SyncSort to reserve below the 16-megabyte line is represented by y's. The amount of virtual storage used by SyncSort is represented by z's. This message reflects the total

amount of virtual storage below and above the 16-megabyte line that was available to SyncSort and used by SyncSort.

WER165I STAT DATA REC NOT WRITTEN

EXPLANATION: The installation default for the SyncSort SMF record feature is applied, but the sort did not invoke the module that creates the sort statistical record. A possible reason for the sort's not invoking the module may be that FREE=CLOSE was coded on the SORTOUT (SYSUT2) or SORTWKxx DD statement.

ACTION: Remove the FREE=CLOSE parameter if full SMF statistics are desired.

WER166I REC LEN GT L3, CUT TO L3

EXPLANATION: SyncSort has truncated a variable-length record prior to output processing. If an E35 exit was in use, the truncated record was longer than the LRECL of the output file's DCB or greater than the l_3 value on the RECORD control statement. If an E35 exit was not in use, the truncated record was longer than the LRECL in the output file's DCB. If OUTFIL processing is requested additional truncation may occur as a result of the OUTFIL processing regardless of the action requested by the VLTEST PARM.

WER167A REC LEN GT L3, USER REQ ABORT

EXPLANATION: Prior to output processing SyncSort has encountered a variable-length record longer than the l_3 value on the RECORD control statement (if an E35 exit was in use) or longer than the LRECL in the output file's DCB. The VLTEST PARM requested SyncSort to terminate when this condition occurs.

ACTION: Change the LRECL in the output file's DCB or the l_3 value on the RECORD control statement (if an E35 exit is used) to reflect the correct record length, or specify another value for the VLTEST PARM.

WER168A CONTROL FIELD WITHIN RDW

EXPLANATION: A SORT/MERGE control field for a variable-length file fell within the Record Descriptor Word of each record. This is a critical error whenever the control field is specified with a ZD or PD format code.

- WER168I CONTROL FIELD WITHIN RDW**
- EXPLANATION: A SORT/MERGE control field for a variable-length file falls within the Record Descriptor Word of each record. (The first byte of the data portion of a variable-length record is at byte position 5.)
- WER169I RELEASE r.ll BATCH nnnn TPF LEVEL tt**
- EXPLANATION: Details on the release level, the batch number and the last TPF applied to SyncSort are given.
- WER170A CONCAT DS, BLKSIZE NOT DIVIS BY LRECL**
- EXPLANATION: One of the files concatenated to a fixed-length SORTIN data set has a BLKSIZE that is not evenly divisible by the original LRECL.
- ACTION: Check the BLKSIZE specified on each of the DD statements concatenated to SORTIN.
- WER171A CONCAT DS, LRECLS NE OR RECFMS DIFF**
- EXPLANATION: One of the files concatenated to a fixed-length SORTIN data set has an LRECL not equal to the original LRECL; or one of the files concatenated to a variable-length SORTIN data set has an LRECL greater than the original LRECL; or one of the files concatenated to a fixed or variable-length SORTIN data set has a RECFM not equal to the original RECFM.
- WER172A CONCAT DS, BLKSIZE GT ORIG BLKSIZE**
- EXPLANATION: One of the files concatenated to a SORTIN data set has a BLKSIZE greater than the original BLKSIZE.
- WER173A BDW INVALID**
- EXPLANATION: The Block Descriptor Word of a block in the SORTIN data set contains a value less than 8; or the Block Descriptor Word contains a value greater than the number of bytes actually read.
- ACTION: Check the data set for the invalid block.
- WER174A RDW INVALID, OVERFLOWS BUFFER**
- EXPLANATION: The Record Descriptor Word of a record in the SORTIN data set is too large. (According to the RDW, the record extends beyond the buffer.)

ACTION: Execute HISTOGRM to check the data set for an invalid record.

WER175A INCORE SORT CAPACITY EXCEEDED

EXPLANATION: There are too many input records to fit in virtual storage.

ACTION: Either increase the amount of virtual storage the sort is able to use or supply SORTWKxx DD statements. (The DYNALLOC option may be used instead of SORTWKxx DD statements.)

WER176A USER EXIT LKED FAILED

EXPLANATION: Exit routine(s) needing to be link-edited were present, but the linkage editor passed a return code greater than 0.

ACTION: Check that the DD statement specified on the MODS control statement is present in the JCL and contains the modules specified on the MODS statement. Check that all exits (except E11, E21, and E31) to be link-edited together have an external name identical to the exit name. Examine the linkage editor output for other errors.

WER177I TURNAROUND SORT PERFORMED

EXPLANATION: SyncSort was able to sort the input file without using intermediate storage (SORTWKxx's). All input data was contained in virtual storage.

WER178A SORTIN [nnnnn] MEMBER NOT FOUND

EXPLANATION: A SORTIN DD statement specified a member of a partitioned data set that could not be found. If a value nnnnn is provided, it represents the concatenation number of the SORTIN data set that has the member-not-found condition.

ACTION: Check the SORTIN DD statement for an error or list the members of the partitioned data set.

WER179A ddname INVALID DCB PARAMETERS

EXPLANATION: The ddname is either SORTIN, SORTINnn, SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. SyncSort is unable to derive RECFM, LRECL, and BLKSIZE parameters from the JCL, the DSCB on the disk or the tape label.

ACTION: Check the JCL and the disk or tape labels for the error.

WER180A ddname MEMBER NOT SPECIFIED

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The indicated DD statement defines a partitioned data set, but a member name has not been specified.

ACTION: Specify a member name on the indicated DD statement or change the partitioned data set to a sequential data set.

WER182A INVALID RDW {SORTIN,SORTINXX} BLOCK x

EXPLANATION: An invalid spanned record indicator was detected in a SORTIN file whose RECFM=VBS, or an invalid record length was detected in a copy operation. The block number of the file is given and SORTINXX indicates a merge.

ACTION: Execute HISTOGRM to check the data set for a record containing invalid span bits.

WER183A SORTWORK DATASET REQUIRED

EXPLANATION: SORTWKxx data set(s) are required for one of the following conditions in this execution of SyncSort: (1) INCORE=OFF is specified as a PARM, (2) exit E14 or E16 is activated, (3) the SUM control statement is used, (4) the OUTREC control statement is used, (5) the checkpoint-restart facility is used, (6) SORTOUT is a VSAM data set, (7) the OUTFIL control statement is used. (All conditions only apply to sort applications.)

WER184A INVALID RETURN CODE FROM E32

EXPLANATION: The return code from merge exit E32 must be 8, 12, or 16.

WER185I SORTIN DCBBLK GT ACTUAL, I/O INEFF

EXPLANATION: The I/O rate is reduced to an inefficient level because the blocksize specified for the SORTIN data set is larger than the actual blocksize, causing excessive error correction.

ACTION: Correct the blocksize specification for future jobs.

- WER186I** **SVC nnn IS INCORRECT VERSION OR NON-SYNC SORT - SVC NOT USED - INEFFICIENT SORT**
- EXPLANATION: The SVC did not return a code indicating it was at the correct version level, therefore it was not used. The SVC is either at the wrong SyncSort release/maintenance level or is not a SyncSort SVC. The problem could cause less efficient I/O and/or loss of SMF records.
- ACTION: Notify your system programmer, who should check that the SVC has been installed in the system libraries, has been IPLed into the system, was specified via SYNCMAC, and was not incorrectly overridden via \$ORTPARM or the PARM field.
- WER187A** **ddname CINV SIZE LT RECORD LENGTH BUT SPANNING NOT SPECIFIED**
- EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The record length is greater than the control interval size specified in the definition of the indicated VSAM data set, but the data set definition did not also include a specification for spanned records.
- WER188A** **ddname IS D.A./DSCB NOT FOUND/OBTAIN FAILED**
- EXPLANATION: The ddname is either SORTIN or SORTINnn. SyncSort was unable to successfully issue an OBTAIN for the specified direct access data set and was therefore unable to determine the DCB characteristics for the file. The OBTAIN failed either because the volume parameter was incorrectly specified for the output file indicated or because the data set was deleted from the volume. (NOTE: the data set may still be in the master catalog even though the data set is no longer on the volume.)
- WER189A** **ddname DCB RECFM REQUIRED**
- EXPLANATION: The ddname is either SORTIN or SORTINnn. The RECFM was not specified on the indicated DD statement, nor was it available in the DSCB on disk nor the tape label, and the TYPE operand was not specified on the RECORD control statement.
- WER190A** **ddname DUPLICATE OUTFIL SPECIFICATION**
- EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The indicated output file is referred to more than once in FILES parameters on the OUTFIL control statement.

- WER191A** **ddname BLKSIZE/LRECL INVALID**
- EXPLANATION: The ddname is either SORTIN or SORTINnn. This message is displayed in conjunction with either WER108I or WER109I which will indicate the invalid DCB characteristic specification. BLKSIZE and LRECL must be equal if RECFM=F. BLKSIZE must be evenly divisible by LRECL if RECFM=FB. BLKSIZE must be greater than or equal to LRECL + 4 if RECFM=V.
- WER192A** **ddname DCB LRECL MISSING**
- EXPLANATION: The ddname is either SORTIN or SORTINnn. The LRECL was not specified on the indicated DD statement, in the DSCB on the disk, in the tape label, or on the RECORD control statement.
- WER193A** **ddname DCB LRECL AND BLKSIZE MISSING**
- EXPLANATION: The ddname is either SORTIN or SORTINnn. The BLKSIZE or LRECL must be specified either on the indicated DD statement, in the DSCB on the disk, or in the tape label. Alternatively, an l_1 specification may be included on the RECORD control statement. None of these specifications were made.
- WER194A** **SORTOUT DCB REQRD/TAPE NOT SL**
- EXPLANATION: DISP=OLD was specified on the SORTOUT DD statement, the tape label was not specified as SL in the LABEL parameter, and required DCB information (LRECL, RECFM, BLKSIZE) was not specified.
- WER195A** **ddname DCB REQUIRED/VSAM SORTIN**
- EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The indicated output file requires additional DCB information (RECFM, LRECL or BLKSIZE) on its DD statement.
- WER196A** **ddname RECFM=VB, LRECL GT BLKSIZE**
- EXPLANATION: The ddname is either SORTIN or SORTINnn. RECFM=VB requires the BLKSIZE to be greater than or equal to LRECL + 4.
- WER197A** **ddname RECFM=F/FB, LRECL/BLKSIZE INVALID**
- EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter.

BLKSIZE and LRECL were not equal on the indicated DD statement for RECFM=F, or BLKSIZE was not a multiple of LRECL for RECFM=FB.

WER198A ddname VARIABLE LRECL LE 4

EXPLANATION: The ddname is either SORTIN, SORTINnn, SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The LRECL specification on the indicated DD statement did not allow 4 bytes for the RDW plus 1 byte for data.

WER199A ddname RECORD TYPE=V, BLKSIZE LE 8

EXPLANATION: The ddname is either SORTIN, SORTINnn, SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The BLKSIZE specified for the indicated DD statement did not allow 4 bytes for the BDW, 4 bytes for the RDW plus 1 byte of data.

WER200A ddname RECFM=V/VB LRECL/BLKSIZE INVALID

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. RECFM=V or VB requires the BLKSIZE to be greater than or equal to LRECL + 4.

WER201A ddname is D.A./DSCB NOT FOUND/OBTAIN FAILED

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. SyncSort was unable to successfully issue an OBTAIN for the specified direct access data set, and was therefore unable to determine the DCB characteristics of the indicated file. The OBTAIN failed either because the volume parameter was incorrectly specified for the indicated output file, or because the data set was deleted from the volume. (NOTE: the data set name may still be in the master catalog even though the data set is no longer on the volume.)

WER202A ddname RECFM INCOMPATIBLE WITH SORTIN

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The record formats for the input and output files are not the same. (Both formats must be either fixed-length or variable-length.) If you want to convert a variable-length input file into a fixed-length output file, use the CONVERT parameter of the OUTFIL or OUTREC control statements.

- WER206A** **INVALID PAGEFIX SVC NUMBER**
- EXPLANATION: SyncSort's EXCPVR facility is in use but no page-fix SVC number was specified at installation time.
- ACTION: Inform your systems programmer of this error condition.
- WER207I** **SORTCKPT DD STATEMENT MISSING OR INVALID**
- EXPLANATION: SyncSort could not take checkpoints because a SORTCKPT DD statement was not supplied or the statement specified an invalid device for a checkpoint data set. Invalid devices include DUMMY data sets or devices other than disk or tape. Processing continued but checkpoints were not taken.
- WER208I** **MIXTURE OF SORTWK DEVICES**
- EXPLANATION: SORTWKxx data sets were assigned to different device types.
- WER209B** **xxx PRIMARY AND yyy SECONDARY SORTOUT TRACKS
ALLOCATED, zzz USED**
- EXPLANATION: It was necessary for SyncSort to request one or more secondary allocations for SORTOUT. xxx is the number of tracks that were initially allocated, yyy is the total number of tracks acquired via secondary allocation, and zzz is the total number of tracks actually required to contain the SORTOUT data set.
- WER210I** **E15 RC INVALID, IGNORED**
- EXPLANATION: A return code of 0 or 4 was passed by an E15 exit routine at a time when these return codes are invalid because SyncSort has not passed the E15 a record address. The invalid return code was ignored by SyncSort, and a return code of 8 was presumed.
- WER211B/I** **[] CALLED BY SYNC SORT; RC=xxxx**
- EXPLANATION: The sort statistics routine (the name inserted in the message) is called by SyncSort. RC gives the code returned to SyncSort by the statistics routine. If RC does not equal zero, see "What to Do Before Calling z/OS Product Services" in this chapter.
- WER213A** **ILLEGAL SUM DATA FIELD**
- EXPLANATION: A field with an illegal data length was specified on the SUM statement.

ACTION: Correct the field length.

WER215A [SORTOFnn] {INREC,OUTREC} ARITHMETIC OVERFLOW

EXPLANATION: When using either INREC, OUTREC or OUTFIL OUTREC, an arithmetic calculation or a data format conversion had an overflow. An arithmetic calculation overflow will occur if any intermediate result exceeds 15 decimal digits or if division by zero is attempted. Overflow may also occur when converting a number with a value of 4G or more to BI format or a number with an absolute value of 2G or more to FI format.

ACTION: Review the arithmetic calculations specified in the indicated statement for errors. If they appear to be correct, consider whether the data could possibly cause an overflow or division by zero. If possible, eliminate any data with questionable values via INCLUDE/OMIT. Consider changing the order of the calculations to prevent intermediate calculation overflow.

WER216A SUM FIELD OUTSIDE RANGE

EXPLANATION: A sum field on the SUM control statement is located beyond the record length.

WER217A DYNALLOC {UNIT,STORCLASS} ASSIGNMENT ERROR

EXPLANATION: Either the unit name or storage class name (DFSMS STORCLAS) is missing or specified incorrectly.

WER218A DYNALLOC WORKFILE ASSIGNMENT ERROR

EXPLANATION: More than 32 work files were requested for dynamic allocation.

WER219A DYNALLOC FAILED RC=(nnnn) - uuuuuuuu [-SMS RC=ssss]

EXPLANATION: The execution of the DYNALLOC macro instruction failed. *nnnn* represents the error reason code, *uuuuuuuu* represents either the unit name or storage class name, and *ssss* represents the SMS return code (only present for certain failures detected by SMS). Two possible reason codes are:

021C - Undefined unit name.

0214 - Unit not available. If all specified units are unavailable when DYNALLOC is issued, the DYNALLOC request fails.

For other reason codes, see either IBM publication *z/OS MVS Programming: Authorized Assembler Services Guide SA22-7608* or *OS/390 V2R10.0 MVS Authorized Assembler Service Guide GC28-1763*.

**WER219I DYNALLOC FAILED RC=(nnnn) - uuuuuuuu [-SMS RC=ssss]
SORT PROCESSING CONTINUES**

EXPLANATION: Dynamic allocation was unsuccessful. *nnnn* represents the error reason code, *uuuuuuuu* represents either the unit name or storage class name, and *ssss* represents the SMS return code (only present for certain failures detected by SMS). Sort processing continues with previously allocated SORTWKS and JCL-allocated SORTWKS. For an explanation of the error reason code, see either IBM publication *z/OS MVS Programming: Authorized Assembler Services Guide SA22-7608* or *OS/390 V2R10.0 MVS Authorized Assembler Service Guide GC28-1763*.

WER220A ILLEGAL OVERLAPPING OF SUM FIELDS

EXPLANATION: A SUM field overlaps another SUM field, a SORT/MERGE control field or the Record Descriptor Word of a variable-length record. All of these are illegal.

**WER223A ddname ASCII XLATION, BUT VOLUME IS NOT ASCII TAPE
OR RECFM IS V**

EXPLANATION: The *ddname* is either SORTIN, SORTINnn, SORTOUT, SORTOFxx, SORTOFx or the *ddname* provided by an OUT-FIL FNames parameter. RECFM=D was specified for the indicated file which is not a tape data set. (RECFM=D is valid for tape data sets only.) Or, RECFM=D was specified for the input data set and no DCB was specified for the output data set.

ACTION: In the first case, code correct RECFM for the data set specified; in the latter case, code DCB characteristics for the output data set, and rerun the job.

WER224A ddname NOT DEFINED

EXPLANATION: The *ddname* is either SORTIN, SORTINnn, SORTOUT, SORTOFxx, SORTOFx or the *ddname* provided by an OUT-FIL FNames parameter. A required input or output DD statement could not be found.

WER225I

E35 RC INVALID, IGNORED

EXPLANATION: An invalid return code was received from an E35 exit routine. If an output data set was not present, the invalid code was other than 4 or 8, and SyncSort assumed return code 4. If end of file was reached, the invalid code was other than 8 or 12, and SyncSort assumed return code 8.

WER227A

ddname BLKSIZE GT ASCII LIMIT

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The DD statement for an output data set targeted to an ASCII-labeled tape requested a blocksize greater than 2048 bytes; that violates the standard and cannot be done.

WER228A

ddname DCB BLKSIZE GT TRACK CAPACITY

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The BLKSIZE for the indicated output file was greater than the track capacity of the output device.

ACTION: Specifying the track-overflow RECFM in the DCB may possibly correct the error condition, or the BLKSIZE should be reduced.

WER229A

ddname DSORG NOT PS/PO

EXPLANATION: The ddname is either SORTIN, SORTINnn, SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. This ddname must be a sequential data set (PS) or a partitioned data set (PO) member.

WER230A

[ddname] xxxxxxxx FIELD OUTSIDE RANGE

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. A field specified in the SORT/MERGE statement is not located within the first 4088 bytes of the variable-length record. (This limit is lower if AC, AQ, E, PD0, Y2x or LOCALE CH fields are used.) Alternatively, a field specified for INREC, OUTREC, OUTFIL OUTREC, SECTION control, (SUB)TOTAL, (SUB)MIN, (SUB)MAX, (SUB)AVG or HEADER/TRAILER data field is located beyond the maximum record length. Or INREC, OUTREC, OUTFIL OUTREC or HEADER/TRAILER n/col/date/page attempted to build a record larger than the allowable maximum.

- WER231A** **[ddname] {INREC,OUTREC} - ILLEGAL DATA FIELD**
- EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNames parameter. An error was found in the INREC, OUTREC or OUTFIL OUTREC specification.
- ACTION: Check the statement for alphabetic data in a numeric field, for a parameter value of 0, for an omitted value, for a space value greater than 256X, for incorrect boundary alignment, and for inclusion of the "variable portion" of *fixed-length* input records in the output records. Also, LINES=ANSI or LINES=(ANSI,n) may not be used on the OUTFIL statement when using multi-line OUTREC.
- WER232A** **ddname RECFM=VBS, LRECL MISSING**
- EXPLANATION: The ddname is either SORTIN or SORTINnn. A RECFM of VBS was specified without an accompanying LRECL specification.
- WER233A** **VIO INVALID FOR DYNALLOC**
- EXPLANATION: VIO is not permitted as a unit device for dynamic allocation. This is due to a possible performance degradation if VIO data sets are used as SORTWK.
- WER234I** **DYNALLOC REQUEST FOR GT 32 SORTWKS**
- EXPLANATION: A total of more than 32 work files were specified in both the JCL and the DYNALLOC parameter combined. The number was reduced to 32.
- WER235A** **[ddname] {INREC,OUTREC} RDW NOT INCLUDED**
- EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNames parameter. Four bytes must be provided for the RDW of the variable-length output record in the FIELDS parameter of the INREC, OUTREC or OUTFIL OUTREC specification. These bytes must appear at the beginning of the record and must not be edited.
- WER236A** **[ddname] {INREC,OUTREC} NULL RECORD**
- EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNames parameter. A variable-length INREC, OUTREC or OUTFIL OUTREC output record must contain at least one other data field in addition to the RDW.

Or if multi-line OUTFIL OUTREC is being used, at least one non-blank line must be defined.

WER237I OUTREC RECORD LENGTH=xxxx

EXPLANATION: The *xxxx* represents the length of the record after OUTREC processing. OUTREC occurs prior to E35 and/or SORTOUT/OUTFIL processing. If the data consists of variable-length records, *xxxx* represents the maximum record length.

WER238I POTENTIALLY INEFFICIENT USE OF INREC

EXPLANATION: The INREC control statement has been used to increase the input record length. This can reduce SyncSort's performance because a larger volume of data is being processed than if the OUTREC control statement were used to perform the same function. Typically, increasing the record length with INREC is only useful when expanding SUM fields with leading zeros to prevent an overflow condition during SUM.

ACTION: Revise the application so that addition of data is performed in an OUTREC statement. Be sure to adjust the FIELDS of the SORT, MERGE or SUM control statements if necessary.

WER239A TYPE PARAMETER REQUIRED

EXPLANATION: There was a VSAM input or output file but the TYPE parameter was not specified. Or, an E15 or E32 exit routine is passing all of the records to the sort/merge (no SORTIN/SORTINnn), but the TYPE parameter was not specified on the RECORD control statement.

WER240A ddname UNSUPPORTED DCB FUNCTION

EXPLANATION: The DD statement specified or implied an attribute which is not supported, e.g., hardware keys for a disk output data set or a block prefix length other than 0, 4 or L for an ASCII tape output data set.

WER243I SHORT RECORD FOR SUM

EXPLANATION: One or more variable-length records were too short to contain all the sum fields specified on the SUM control statement. These records were therefore not summarized. Program HISTOGRM may be used to determine the length of the shortest record in the input file.

WER244A **[ddname] {INREC,OUTREC} SHORT RECORD**

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. A variable-length record was too short to contain all the fields specified on the control statement. Program HISTOGRM may be used to determine the length of the shortest record in the input file.

WER246I **FILESIZE x**

EXPLANATION: The number of bytes of input data sorted or copied by SyncSort is given for FILESIZE. This number reflects SORTIN, E15, INCLUDE/OMIT and INREC processing. Note the following:

1. For MAXSORT, the FILESIZE is given in kilobytes for each individual sort in a WER351I message; the FILESIZE in the WER246I for the final merge is the sum of the individual sorts' sizes and, because of truncation in each intermediate sort, may not be exact.
2. When WER246I is issued instead of WER054I in a variable-length record copy operation, the number of bytes processed (copied) includes multiple segment descriptor words for a single record if the record is comprised of multiple segments on SORTIN, since all segments were copied; for a variable-length record sort or merge operation, the number of bytes processed (sorted or merged) includes a single record descriptor word for each record even if the record is comprised of multiple segments on SORTIN, since it is records, not record segments, that are being operated on.

WER247A **ddname HAS INCOMPATIBLE LRECL**

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. There is a conflict between the LRECL specification for the indicated output file and either the post-OUTFIL or post-OUTREC record length. Padding of records is not permitted after OUTFIL processing, so the LRECL may not be greater than the post-OUTFIL record length. Alternately, truncation of records is not permitted after the OUTREC statement or the OUTFIL OUTREC processing, so the LRECL may not be less than the post-OUTREC record length.

WER250A **[ddname] INCLUDE/OMIT FIELD BEYOND RECORD**

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter.

A compare field specified for INCLUDE, OMIT or OUTFIL INCLUDE/OMIT extended beyond the end of the record.

WER251A **xxxx INVALID yyyyyyyyyy**

EXPLANATION: The *xxxx* represents either INCL (INCLUDE) or OMIT. The invalid relational condition represented by *yyyyyyyyyy* was found in the INCLUDE or OMIT specification.

WER253A **INCLUDE/OMIT FORMATS INCOMPATIBLE**

EXPLANATION: A relational condition specified on an INCLUDE or OMIT control statement or OUTFIL parameter contains an invalid field-to-field, field-to-constant or field-to-mask comparison. Note that if LOCALE processing has been specified, a CH to BI comparison is not supported.

WER254A **ddname VSAM {OPEN,CLOSE} ERROR - xx**

EXPLANATION: The *ddname* is either SORTIN, SORTOUT, SORTOFnn, SORTOFn, or the *ddname* specified on an OUTFIL FNames parameter. An error occurred during an attempt to OPEN or CLOSE the indicated VSAM file. For the definition of the error number, *xx*, consult the following IBM publication:

- *DFSMS Macro Instructions for Data Sets*

WER255A **VSAM LOGICAL ERROR xx ON {INPUT,OUTPUT}**

EXPLANATION: An error occurred while processing a VSAM data set. For the definition of the hexadecimal error number represented by *xx*, see one of the following IBM publications:

- *DFSMS Macro Instructions for Data Sets*

WER256I **ddname VSAM file, RECORDS PADDED ON OUTPUT**

EXPLANATION: The *ddname* will be SORTOUT, SORTOFxx, SORTOFx or the *ddname* provided by an OUTFIL FNames parameter. The fixed-length VSAM LRECL for the indicated output file is greater than the length of the records at the end of SyncSort processing. SyncSort padded the output records with filler characters on the right.

- WER257I** **INREC RECORD LENGTH=xxxxxx**
- EXPLANATION: *xxxxxx* represents the length of the record immediately after INREC processing. If you have variable-length records, *xxxxxx* represents the maximum record length.
- WER258A** **DUPLICATE DDNAME: SORTINxx**
- EXPLANATION: Two input files for a merge have the same number. The file number is given.
- WER259A** **DUPLICATE ALTSEQ STATEMENT**
- EXPLANATION: Two ALTSEQ control statements were found.
- WER260I** **RECOVERY FROM B37 SUCCESSFUL. SORT PROCESSING CONTINUES**
- EXPLANATION: SyncSort recovered from a B37 abend and continued processing.
- WER262I** **REENTRANT SORT NOT RESIDENT - INEFFICIENT SORT**
- EXPLANATION: The resident SyncSort load module(s) were loaded into the private area instead of being executed from the Link Pack Area/Extended Link Pack Area. This situation may have occurred because the module(s) were found in a STEPLIB/JOBLIB DD data set. Loading the resident modules into the private area limits the amount of virtual storage available to the sort and may reduce the efficiency of the sort.
- ACTION: Contact the systems programmer in charge of SyncSort installation.
- WER263A** **ILLEGAL USE OF MULTI-VOLUME SORTWK**
- EXPLANATION: SyncSort does not support the use of multi-volume disk SORTWK data sets. (However, if SyncSort only requires the use of the space on the first volume of a multi-volume SORTWK file, this error message will not be issued.)
- ACTION: Remove the volume count subparameters of the UNIT parameter on all SORTWK DD statements that specify more than one volume.

- WER264A** **UNEQUAL REC LENS - VSAM SORTIN - TYPE=F**
- EXPLANATION: A record in a fixed-length VSAM input data set was encountered whose length was not equal to the length specified in the RECORD statement or VSAM cluster definition.
- ACTION: Use the IDCAMS utility to identify and correct the records in error.
- WER265A** **ddname VSAM CONCATENATED SORTIN NOT ALLOWED**
- EXPLANATION: The ddname indicated represents either a SORTIN or SORTINnn input file which consists of concatenated VSAM data sets. SyncSort does not support concatenated VSAM input files.
- WER266A** **ALTPARM - PARM LENGTH GT MAX SUPPORTED**
- EXPLANATION: The length of the parameter list passed through the alternate parameter data set exceeded the 256 byte limitation.
- WER267A** **statement STATEMENT: STATEMENT NOT FOUND**
- EXPLANATION: A required SORT/MERGE or RECORD statement (as indicated in the message text) is missing.
- WER268A** **statement STATEMENT: SYNTAX ERROR**
- EXPLANATION: A SyncSort control statement, as indicated in the message text, contains a syntax error. The next line will contain an '*' indicating the approximate location of the syntax error.
- WER269A** **statement STATEMENT: DUPLICATE STATEMENT FOUND**
- EXPLANATION: More than one SORT/MERGE, INCLUDE/OMIT, INREC, OUTREC, RECORD, MODS, SUM, ALTSEQ or END statement was found, as indicated.
- WER270A** **statement STATEMENT: DUPLICATE PARM FOUND**
- EXPLANATION: A single parameter was multiply specified on the indicated SyncSort control statement; or a single parameter was specified both in the invoking parameter list and in the control statements.
- WER271A** **statement STATEMENT: NUMERIC FIELD ERROR**
- EXPLANATION: A numeric field has been improperly specified on the indicated SyncSort control statement.

- WER272A** **statement STATEMENT: PARS NOT FOUND**
- EXPLANATION: Required parameters have not been included on the indicated SyncSort control statement.
- WER273A** **BLANK STATEMENT FOUND**
- EXPLANATION: A blank statement has been encountered.
- WER274A** **CONTINUATION STATEMENT ERROR FOUND**
- EXPLANATION: SyncSort has encountered a statement containing a continuation indicator, but cannot locate a continuation statement which should follow.
- WER275A** **NO KEYWORDS FOUND ON CONTROL STATEMENT**
- EXPLANATION: A required keyword has not been specified on a SyncSort control statement.
- WER300A** **SORTBKPT DD STATEMENT REQUIRED**
- EXPLANATION: The SORTBKPT DD statement was not included in the job stream. This is a required data set for all MAXSORTs.
- WER301A** **SORTBKPT DATA MUST RESIDE ON DISK**
- EXPLANATION: The SORTBKPT data set must be allocated to a disk device.
- WER302A** **SORTBKPT TRACK CAPACITY TOO SMALL**
- EXPLANATION: Direct access devices with a track capacity smaller than 3600 bytes cannot be used for the SORTBKPT data set.
- WER303A** **SORTBKPT SYSTEM OPEN FAILURE**
- EXPLANATION: The operating system could not open the SORTBKPT data set.
- ACTION: Check to see that the DD statement is correct. Determine if operating system is at proper maintenance level.
- WER304A** **SORTBKPT RECORD FORMAT ERROR**
- EXPLANATION: There is a record format error in the SORTBKPT data set.

ACTION: Check that the SORTBKPT DD statement points to the correct DSNNAME. Check that the data set has not been inadvertently written into and modified. Use the HEX function on the OUTREC statement or OUTREC parameter on the OUTFIL statement to get a hex format listing of the data.

WER305A SORTBKPT RECORD EXCEEDS BLKSIZE

EXPLANATION: The use of an excessive number of parameters in a control statement has caused the SORTBKPT data set to overflow the maximum blocksize limit of 32760.

ACTION: Reduce the size of the control statement specification if possible, or convert the application from a MAXSORT to a conventional sort.

WER306A RESTART FROM BREAKPOINT PROHIBITED

EXPLANATION: The SORTBKPT data set indicates that a program-initiated sort or a sort with exit programs tried to restart from a breakpoint.

ACTION: Use z/OS checkpoint facilities since only these will save your work areas and the program memory for restart.

WER307A SORTBKPT RECORD SEQUENCE ERROR

EXPLANATION: An out-of-sequence record was read from the SORTBKPT data set.

ACTION: Use the HEX function on the OUTREC statement or OUTREC parameter on the OUTFIL statement to get a hexadecimal listing of the data set for analysis. See if the data set was damaged by another program. Check system for hardware error.

WER308A BREAKPOINT ID NOT FOUND ON SORTBKPT

EXPLANATION: The parameter RESTART=id was specified but id could not be found.

ACTION: Check spelling, correct, and return.

WER309A SORTOUXX DATA MUST BE ON DISK OR TAPE

EXPLANATION: Intermediate sort output data was allocated to an unsupported device. Only disk or tape is allowed.

ACTION: Allocate SORTOUxx data to either disk or tape.

- WER310A SORTOUXX DEVICE MIXING PROHIBITED**
- EXPLANATION: Intermediate sort output was allocated to both tape and disk in the same job or to a mixture of disk device types.
- ACTION: Allocate all intermediate sort data to the same device type.
- WER311A DISK SORTOUXX REQUIRES SORTOUXX DD**
- EXPLANATION: No SORTOUxx DD statements were found so there was no place to store intermediate sort output.
- ACTION: Supply one or more SORTOUxx DD statements with xx represented by 01 to 99.
- WER312A TAPE SORTOUXX REQUIRES SORTOU00 DD**
- EXPLANATION: One or more SORTOUxx DD statements were allocated to tape but the SORTOU00 statement was not present.
- ACTION: Allocate a tape unit using the SORTOU00 DD statement.
- WER313A SORTOUXX DEVICE NOT SUPPORTED**
- EXPLANATION: The SORTOUxx DD statements specify an unsupported device type.
- ACTION: Change the device allocation of the SORTOUxx data set.
- WER314A INSUFFICIENT VIRTUAL STORAGE FOR MAXSORT**
- EXPLANATION: MAXSORT cannot run efficiently in the amount of virtual storage provided.
- ACTION: Increase virtual storage or decrease the number of tape units requested by MINMERGE.
- WER315A SORTOUXX BLKSIZE GT TRACK CAPACITY**
- EXPLANATION: Intermediate sort output is on disk, but the SORTIN data set requires too large a blocksize for a disk device.
- ACTION: Allocate intermediate sort output to tape and rerun.

WER316A

INSUFFICIENT SORTOUXX DD STATEMENTS

EXPLANATION: The data to be sorted requires one or more additional data sets.

ACTION: Recalculate and restart the job including additional SORTOUxx DD statements. (Make sure each statement's number is greater than the last one you put in.)

WER317I

MAXSORT OPTION SELECTED

EXPLANATION: A MAXSORT was requested.

WER318I

INPUT CARDS IGNORED - SORTBKPT USED

EXPLANATION: The control statement just listed on SYSOUT for a breakpoint restart were not used to control sorting. Whatever control statements were specified when the job was started were used. (They may be the same as the statements just listed, however.)

WER319I

SORT RESTARTED AT BKPT xxxxxxxxxxxxxx

EXPLANATION: This message identifies the breakpoint id from which MAXSORT resumes execution on a breakpoint restart.

WER320I

INEFFICIENT SORTOUXX BLKSIZE FORCED

EXPLANATION: Due to the constraints between the amount of memory and the value specified for MAXMERGE, MAXSORT was forced to compromise and choose a smaller blocksize than would permit efficient buffering in sorts and merges.

ACTION: If you wish a more efficient MAXSORT, either increase the amount of memory or reduce the number specified for MAXMERGE. This will permit a larger blocksize to be chosen which will allow multiple buffering of all the intermediate sort output data.

WER321B

SORTOUXX BLKSIZE=xxxxx

EXPLANATION: This gives the blocksize that MAXSORT has chosen for intermediate sort output.

WER322A

TAPE DYNALLOC FAILURE - CODE=xxxx

EXPLANATION: Attempts to dynamically allocate tape units for a merge phase met with unexpected failure. Code xxxx gives the hexadecimal return code from the dynamic allocation request. For an explana-

tion of this code, see either IBM publication *z/OS MVS Programming: Authorized Assembler Services Guide SA22-7608* or *OS/390 V2R10.0 MVS Authorized Assembler Service Guide GC28-1763*.

WER323A BKPT DATA AT DIFFERENT RELEASE LEVEL

EXPLANATION: The SORTBKPT data was created by a different SyncSort release than the SyncSort program reading it. Because of this, the breakpoint data cannot be processed.

ACTION: Restart this job and run under the same SyncSort release that you started with.

WER324A TAPENAME CLASS NOT FOUND ON SYSTEM

EXPLANATION: The tapes could not be dynamically allocated because a TAPENAME was specified that was not generated into the operating system.

ACTION: Check with the systems programmer for acceptable unit names.

WER325A MAXSORT STOPPED BY OPERATOR

EXPLANATION: The operator responded to a message by stopping the sort. The sort may be restarted from the last breakpoint or checkpoint.

WER326A DYNALLOC UNALLOC FAILURE - CODE=xxxx

EXPLANATION: Attempts to dynamically deallocate tape units met with unexpected failure. Code *xxxx* gives the hexadecimal return code from the dynamic deallocation request. For an explanation of this code, see either IBM publication *z/OS MVS Programming: Authorized Assembler Services Guide SA22-7608* or *OS/390 V2R10.0 MVS Authorized Assembler Service Guide GC28-1763*.

WER327A INSUFFICIENT UNITS FOR MINIMAL MERGE

EXPLANATION: Too few tape units were allocated to meet the number specified in MINMERGE. Either too few SORTOUxx DD were supplied or the z/OS system was unable to dynamically allocate enough units.

ACTION: Restart the job with additional SORTOUxx DD statements.

- WER328A SORTOUXX SYSTEM OPEN FAILURE**
- EXPLANATION: The operating system could not open the SORTOUxx data sets.
- ACTION: Check to see that SORTOUxx DD statements are correct. Determine if operating system is at a proper maintenance level.
- WER329A SORTOU00 SYSTEM RDJFCB FAILURE**
- EXPLANATION: The operating system could not read the Job File Control Block for SyncSort analysis.
- ACTION: Determine if operating system is at a proper maintenance level.
- WER330A SPECIFIED SORTING TIME HAS EXPIRED**
- EXPLANATION: The time limit specified in the SORTTIME parameter has expired. The job may be restarted from the last breakpoint or checkpoint.
- WER331A SYSTEM CHECKPOINT FAILURE**
- EXPLANATION: Request for z/OS checkpoint facilities failed.
- ACTION: Ascertain that the SORTCKPT DD statement was correctly specified. Check that rules for the use of checkpoint were not violated.
- WER350I {SORT/MERGE} # XX COMPLETE {AT BREAKPOINT/AT CHECKPOINT} bbbbbbbbbbbb, DIAG=hhh,hhh...**
- EXPLANATION: This message tells which individual sort or merge has completed. Restart can be performed from the breakpoint or checkpoint id given in *bbbbbbbbbbbb*. If restart is not possible the above message will read:
- SORT/MERGE # XX COMPLETE.**
- The hexadecimal information following the DIAG keyword is likely to change from execution to execution. It is internal diagnostic information intended for use by SyncSort personnel in Product Support.
- WER351I DATA SIZE xxxx KB [FROM yy WAY MERGE]**
- EXPLANATION: The amount of data that was processed for the current SyncSort individual sort/merge is given in kilobytes. When a merge is processed yy gives the number of tape units used.

WER352I DYNAMICALLY ALLOCATED TAPE UNITS - XX

EXPLANATION: The number of tapes drives that were dynamically allocated for the current merge pass is given.

WER353I STARTING TIME hh.mm.ss - ENDING TIME hh.mm.ss

EXPLANATION: The starting and ending times in hours, minutes, and seconds of the individual sort or merge just completed are given.

WER354I -----DATA SET STATUS-----

EXPLANATION: This is a header. Messages relating to data sets will follow.

WER355I {DSN=dsname/VOL SERS = vvvvvv...}

EXPLANATION: The data set names of the tapes for intermediate sort output are given. The tape volumes are listed for tape intermediate sort output. Retain these reels for input to a later merge.

WER356I SORTOUXX DD STATEMENT IS ACTIVE

EXPLANATION: The disk data set allocated to the SORTOUxx DD statement is needed as input to a subsequent merge. Be sure to keep it in case restart is necessary.

WER375D jobname.stepname - MAXSORT BKPT id
TIME ESTIMATE: XXX MINUTES UNTIL NEXT
NOTIFICATION.
REPLY 'GO' TO CONTINUE, 'STOP' TO TERMINATE

EXPLANATION: A long-running MAXSORT has exhausted its assigned block of computer time.

ACTION: The operator's decision should be based on scheduling priorities and the estimated time of the sort. A 'GO' reply will permit sort execution to proceed in stages. This message is generated at discrete intervals so that the operator can again opt to continue or terminate its execution.

WER376D jobname.stepname - MAXSORT BKPT id
aaa TAPE UNITS ALLOCATED TO jobname
bbb TAPE UNITS NEEDED FOR BEST PERFORMANCE
TIME ESTIMATE USING aaa TAPE UNITS -
xxxxx MINUTES TO {NEXT BREAKPOINT | END OF JOB}

REPLY 'GO' TO CONTINUE, 'STOP' TO TERMINATE, 'NN' # UNITS

EXPLANATION: The first time this message is generated, it indicates that MAXSORT has dynamically allocated the optimum number *aaa* of tape drives up to MAXMERGE. Reissued, this message documents MAXSORT's response to the operator's previous reply of 'NN' tape units. 'NN' represents the total number of tapes that will be allocated.

ACTION: Given a reply of 'NN' tape drives, MAXSORT will attempt to satisfy the operator's request. For 'NN' larger than *aaa*, MAXSORT will try to raise its allocation to 'NN'. (The operator can delay the request for more tape units in order to give other jobs time to free any tape drives they are using.) The above message is reissued and the operator can see how the decision will affect sort execution.

As soon as allocations and time estimates are satisfactory, the reply 'GO' will cause continued execution using the allocated tape units. If allocation or time estimates are not satisfactory, the job may be terminated (reply 'STOP') or a new number 'NN' of units may be requested.

WER377D

**jobname.stepname - MAXSORT BKPT id
INSUFFICIENT TAPE UNITS AVAILABLE
aaa TAPE UNITS ALLOCATED TO jobname
bbb TAPE UNITS NEEDED TO CONTINUE EXECUTION
REPLY 'RETRY' TO GET UNITS, 'STOP' TO TERMINATE**

EXPLANATION: MAXSORT cannot immediately acquire enough tape drives to make continued processing worthwhile.

ACTION: The operator can wait until other tape drives have been released, then reply 'RETRY'. If enough drives are now available, execution continues. Otherwise the above message is repeated. Eventually enough tape drives become available or the operator terminates the job with a 'STOP' response.

WER378I

**NO ADDITIONAL TAPE UNITS EXIST FOR GENERIC CLASS
tapename**

EXPLANATION: All tape units on the system within the TAPENAME class have been allocated. Further DYNALLOC attempts will fail to acquire more tape units. Message WER376D or WER377D will follow.

WER390A

MINIMUM SORTWK SPACE NOT AVAILABLE

EXPLANATION: MAXSORT could not obtain enough SORTWK disk space to run. When MAXSORT is executing with larger storage values,

SyncSort may need to automatically raise MINWKSP, overriding the specified MINWKSP value. Therefore, it may erroneously appear that JCL SORTWKS provided enough space to satisfy MINWKSP when this message was posted.

ACTION: Correct SORTWK volume, primary, and secondary allocations. Restart the job.

WER391A INSUFFICIENT VIRTUAL STORAGE FOR SORTBKPT BUFFER

EXPLANATION: MAXSORT was unable to obtain the necessary 3600-byte buffer space from the operating system.

ACTION: Check to see that sufficient virtual storage was allocated to the sort.

WER392A SORTBKPT FORMAT ERR - VBS PROCESSING

EXPLANATION: MAXSORT attempted to read back control information associated with VBS SORTIN data and found a format error in the SORTBKPT data set.

ACTION: In the U.S. and Canada, call SyncSort for z/OS Product Services directly at (201) 930-8260. Elsewhere, call your SyncSort support representative.

WER393I TURNAROUND MAXSORT SORT PERFORMED

EXPLANATION: The amount of SORTIN data was small enough to fit entirely on SORTWK disk space, so sorted data was produced in one SyncSort pass.

WER394A SORTOUXX DD STMT REQUIRED FOR MERGE

EXPLANATION: The above DD statement was required for disk intermediate sort output as input to a merge but could not be found.

ACTION: Supply the missing DD statement.

WER395A INVALID SORTOU00 OR SORTOUxx DSN PREFIX

EXPLANATION: The BKPTDSN parameter was used, but the required trailing period was not specified as part of the DSN prefix.

ACTION: Add a trailing period to the parameter specification.

WER396A

LKED DD STATEMENT MISSING OR INVALID

EXPLANATION: A MODS statement specified at least one exit to be link-edited by SyncSort, but a SYSPRINT and/or SYSLIN and/or SYSLMOD DD statement is missing. All of these statements are required for link-editing. Or, the SYSLMOD DD statement does not refer to a data set on a direct access device.

ACTION: Supply the missing DD statement(s) or adjust the SYSLMOD DD statement as appropriate.

WER400A

SORTIN(nn) IS AN UNINITIALIZED SEQUENTIAL DISK DATA SET

EXPLANATION: The data set was allocated but never opened for output. Therefore, there is no valid data or end-of-file mark in the data set. This condition usually occurs when a program abends and the steps to create the data are bypassed.

ACTION: Write the appropriate data or end-of-file mark in the data set.

WER401A

CSECT NAME DIFFERENT THAN MEMBER NAME

EXPLANATION: The MODS statement specified an exit routine module in SYSIN that was not found.

ACTION: Either change the member name in the MODS statement to match the module name or reassemble the exit module with a name to match the member name on the MODS statement.

WER402A

SORTMODS STOW FAILURE

EXPLANATION: While copying an exit routine from SYSIN to SORTMODS, SyncSort attempted unsuccessfully to store (STOW) the exit routine in the SORTMODS directory. This condition is caused either by specifying insufficient directory blocks when creating the SORTMODS data set or by the presence of a member or alias with the same name as the exit routine in the SORTMODS data set, or by a hardware failure.

ACTION: Check the SORTMODS directory names for a member-name conflict and rerun the job step.

WER403A

OUTFIL NOT VALID FOR MAXSORT

EXPLANATION: Using an OUTFIL control statement in a MAXSORT application is not supported at this time.

ACTION: Either remove the OUTFIL statement(s) or convert the application not to invoke MAXSORT.

WER404I SORTXSUM: RECFM= ;LRECL= ;{BLKSIZE=,CISIZE=} [;CINV ACCESS]; RCD OUT n

EXPLANATION: This informational message lists the DCB characteristics used by SyncSort to process the SORTXSUM file, as well as the number of records (n) that were written to the data set. For a VSAM data set, the CISIZE is provided; if control interval access was used, the CINV ACCESS portion of the message will be displayed.

WER405I ddname DATA RECORDS OUT n, TOTAL RECORDS OUT y

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The n represents the number of data records (exclusive of HEADERS/TRAILERS and multi-record OUTREC) in each output data set. The y represents the total number of records in each output data set (data records, HEADERS/TRAILERS and multi-record OUTREC records). Note that the total number of lines written to the line printer may be greater than the actual record count since multiple lines can be generated from one data record using ANSI control characters.

WER406A ddname HEADER/TRAILER/DATA LINES EXCEED PAGE SIZE

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. The number of lines generated by some HEADER and/or TRAILER and/or multi-line OUTREC parameters is greater than or equal to the number of lines to be written per logical page as specified by the LINES parameter. If LINES has not been coded, this number defaults to 60.

ACTION: Reduce the number of HEADER/TRAILER lines generated or increase the number of lines in the LINES parameter so that a minimum of all output lines from 1 data record can be written per logical page.

WER407I UNUSABLE SORTWK DEVICE ALLOCATED {,NON RPS,UNIT=VIO}

EXPLANATION: An unusable device was allocated during dynamic allocation. The device was held for the duration of the sort; however, the device was not used for SORTWK storage.

ACTION: For future executions, ensure that the DYNALLOC runtime parameter specifies a correct disk device. If the message cites "NON

RPS," specify RPS disk devices; if the message cites "UNIT=VIO," specify a true disk device.

WER409A MOD ON SYSIN NOT FLAGGED AS SYSIN MODULE

EXPLANATION: An object deck was found in the SYSIN data set that, according to the MODS statement, was not specified as belonging in SYSIN.

WER410B xxx BYTES OF VIRTUAL STORAGE AVAILABLE ABOVE THE 16MEG LINE, yyy BYTES RESERVE REQUESTED, zzz BYTES USED

EXPLANATION: The amount of virtual storage above the 16-megabyte line available (free) when SyncSort received control is represented by x's. The amount of virtual storage that the user requested SyncSort to reserve above the 16-megabyte line is represented by y's. The amount of virtual storage used by SyncSort above the 16-megabyte line is represented by z's.

WER411B nnn BYTES OF EMERGENCY SPACE ALLOCATED ABOVE THE 16MEG LINE

EXPLANATION: The indicated amount of virtual storage above the 16-megabyte line has been set aside by SyncSort for use by other programs (e.g., program invoking the sort, system SVCs, tape management system.)

WER412I ERROR TAKING SYSTEM CHECKPOINT. PROCESSING CONTINUES

EXPLANATION: An error occurred when SyncSort attempted to take a user-requested checkpoint. Sort/merge processing continued; however, a usable checkpoint may not exist. Refer to the IHJxxxx message in the job log to determine the cause of the error.

WER414A SORTIN(nn) OPEN ERROR ON AN UNINITIALIZED SEQUENTIAL DISK DATA SET

EXPLANATION: An error occurred during an OPEN of a multi-volume uninitialized sequential disk data set being used for SORTIN(nn). When the UNINTDS=YES option has been selected, either by default or parameter override, SyncSort will need to open for output a multi-volume uninitialized disk data set in order to set the DS1IND80 flag in the format-1 DSCB of the first volume. Typically this error will occur if the SyncSort step does not have the authority to open the SORTIN(nn) for output processing.

ACTION: In a separate step prior to the SyncSort invocation, write the appropriate end-of-file mark in the first volume of the multi-volume data set.

WER415B DSM FACILITY DISABLED

EXPLANATION: SyncSort's dynamic storage management feature was not active for this sort execution.

WER416B

```

{access-method WAS USED FOR ddname
{ddname: EXCP'S=eee [,UNIT=uuuu] [,DEV=dddd] [,CHP=cccccccc,n][,VOL=vvvvvv]
{TOTAL OF xxx EXCP'S ISSUED FOR totalid
  
```

EXPLANATION: This message provides summary I/O tuning information for files processed by SyncSort. The first form is used when an access method other than EXCP is used for a file. It uses a generic term for the access method (BSAM, HIPERBATCH, etc.) and the file for which it was used. When EXCP is used, the message takes on the second form which has the component parts listed below. Some of these components may or may not be included in the message depending on the level of the operating system and the availability of the information within SyncSort.

- EXCP'S=eee "eee" identifies the number of EXCPs issued for the file. For input files such as SORTIN, this is the total EXCPs issued for all concatenated input sets.
- UNIT=vuuuu "uuuu" is the unit type on which the data set resides. For files that can consist of concatenations or multi-volume data sets, the unit type displayed is for the first volume of the first data set.
- DEV=dddd "dddd" is the device name for the first or only device for the file.
- CHP=cccccccc,n This field identifies the channel paths available to the first or only device. "n" is the number of PAV aliases available.
- VOL=vvvvvv This field is displayed for only DASD devices and identifies the volume serial number of the first or only volume for the file.

For certain types of sorts, SyncSort may dynamically allocate data sets other than SORTWKxx data sets for use in the sorting process, and this can occur whether or not normal dynamic allocation of sortwork data sets is enabled. When used, such data sets are collectively represented in a single WER416B message using a ddname of "SORTWK&&" for the purpose of reporting EXCPs issued against them.

In the third form of the message, *xxx* provides a total of the EXCPs issued for SORTWORKS, SORTING, COPYING, or MERGING, as identified by "totalid."

WER417A UNEQUAL MAINTENANCE LEVELS: xxxxxxxx,yy,zz

EXPLANATION: The load module *xxxxxxx* and SyncSort root module maintenance levels do not correspond. *yy* represents the maintenance level of the *xxxxxxx* module; *zz* represents the maintenance level of the root module.

ACTION: Contact the systems programmer in charge of SyncSort maintenance.

WER418I DATASPACE(S) AND/OR xxxxxxxx USED

EXPLANATION: *xxxxxxx* can be either ZSPACE or HIPERSPACE(S). SyncSort has dynamically chosen to use data space, ZSPACE, or hiperspace during the execution of the sort. ZSPACE is a technique within SyncSort created as a replacement for hiperspace. It allows native use of the central storage resources which are available. This technique eliminates the additional overhead produced when hiperspace is simulated by the operating system in a z/Architecture environment. It provides superior CPU performance and reduced system overhead compared to a conventional hiperspace application.

WER420I COBOL ACCELERATOR ACTIVE

EXPLANATION: SyncSort's high performance access method was used for accessing a COBOL file.

WER422A SORTOUT STOW FAILURE

EXPLANATION: When writing to SORTOUT, SyncSort attempted unsuccessfully to store (STOW) the SORTOUT PDS member in the SORTOUT directory. This condition is caused by specifying insufficient directory blocks when creating the SORTOUT data set.

ACTION: Recreate the SORTOUT data set with more directory blocks and rerun the job step.

WER423I

DYNAMIC ALLOCATION RETRY - WAITING FOR SPACE

EXPLANATION: The DYNALLOC facility is being used to acquire sortwork space, but there is currently insufficient disk space on the system to satisfy the request. SyncSort will wait the prescribed number of minutes as specified by the DYNALLOC option and then retry the request.

WER424I

DYNAMIC ALLOCATION RETRY SUCCESSFUL

EXPLANATION: The dynamic allocation of sortwork space after a DYNALLOC RETRY attempt was successful. Sort processing continues.

WER425A

CONVERT FEATURE CANNOT BE USED FOR FIXED-LENGTH RECORDS

EXPLANATION: The CONVERT parameter of the OUTREC or OUTFIL statements has been used incorrectly. CONVERT can be used to convert variable-length input to fixed-length output records only. Also, the record format of an output data set must be fixed-length after CONVERT processing.

WER426I

SORT INTERNAL ERROR - RECOVERY ATTEMPT IN PROGRESS

EXPLANATION: The presence of this message indicates that an automatic retry of the SyncSort execution has been initiated. If the error recovery is successful, the SyncSort SYSOUT listing will contain a subsequent set of messages representing the complete information about the execution. The subsequent set of messages may be separated from the initial set of listings by a diagnostic output of significant size. The new listing will contain the message WER427I.

WER427I

RECOVERY ATTEMPT IN PROGRESS

EXPLANATION: The set of SYSOUT messages containing the WER427I will be from the automatic retry execution. Examine these messages to insure that it also contains a WER052I message indicating a successful completion of the SyncSort execution. In addition, a successful SyncSort recovery will complete with a return code of zero. Even if the WER426I and WER427I messages are present, this in itself does not constitute a successful recovery unless zero is returned for the step completion code.

If an execution of SyncSort does utilize the recovery facility, whether successfully or not, the SyncSort for z/OS Product Services Group should be contacted so that the underlying error can be investigated and resolved.

WER428I CALLER-PROVIDED IDENTIFIER IS "xxxx"

EXPLANATION: SyncSort was invoked by another program, and that program used a 31-bit parameter list where the "call identifier" parameter was specified. *xxxx* is the identifier specified by the calling program.

**WER429I SORT INTERNAL ERROR ON SORTWK_{nn} - RECOVERY
ATTEMPT IN PROGRESS**

EXPLANATION: An internal error occurred while processing the SORTWK data set indicated by *nn*. The presence of this message indicates that SyncSort has initiated automatic error retry logic to correct this error. If the recovery is successful, processing will resume and message WER052I will be issued when the sort has completed successfully. Absence of the WER052I message indicates that SyncSort was unable to recover.

ACTION: If an execution of SyncSort does use this recovery facility, whether successfully or not, SyncSort for z/OS Product Services should be contacted so that the underlying condition can be investigated and resolved.

**WER430I SORT INTERNAL ERROR ON SORTOUT - RECOVERY
ATTEMPT IN PROGRESS**

EXPLANATION: An internal error occurred while creating the SORTOUT data set. The presence of this message indicates that SyncSort has initiated automatic error retry logic to correct this error. If the recovery is successful, processing will resume and message WER052I will be issued when the sort has completed successfully. Absence of the WER052I message indicates that SyncSort was unable to recover.

ACTION: If an execution of SyncSort does use this recovery facility, whether successfully or not, SyncSort for z/OS Product Services should be contacted so that the underlying condition can be investigated and resolved.

WER431I COPY SUBSTITUTED FOR MULTIPLE OUTFILS

EXPLANATION: The SORT or COPY multiple output application (multiple OUTFILs) has been automatically converted by SyncSort to a single SORT or COPY operation followed by one or more COPY operations.

If system resources are available and the output files of a multiple output application have identical specifications, SyncSort will make this

type of change to take advantage of system resources to improve the application's performance.

WER432I {SORT,MERGE} FORMAT OPERAND IGNORED

EXPLANATION: On either a SORT or MERGE control statement, the format of the keys was specified in both the FIELDS and FORMAT parameters. SyncSort ignores the FORMAT parameter and uses the individual format specifications within the FIELD parameter.

WER433I SUM FORMAT OPERAND IGNORED

EXPLANATION: On a SUM control statement, the sum field format was specified in both the FIELDS and FORMAT parameters. SyncSort ignores the FORMAT parameter and uses the individual format specifications within the FIELD parameter.

WER434I INCLUDE/OMIT FORMAT OPERAND IGNORED

EXPLANATION: On an INCLUDE or OMIT control statement, the field format was specified in both the COND and FORMAT parameters. SyncSort ignores the FORMAT parameter and uses the individual format specifications within the FIELD parameter.

**WER435A SORTIN(nn) ALLOCATION ERROR ON AN UNINITIALIZED
SEQUENTIAL DISK DATA SET**

EXPLANATION: An error occurred during the dynamic allocation of a multi-volume uninitialized sequential disk data set being used for SORTIN(nn). When the UNINTDS=YES option has been selected, either by default or parameter override, SyncSort will need to dynamically allocate and open for output a multi-volume uninitialized disk data set in order to set the DS1IND80 flag in the format-1 DSCB of the first volume.

ACTION: In a separate step prior to the SyncSort invocation, write the appropriate end-of-file mark in the first volume of the multi-volume data set.

**WER436I UNEQUAL MAINTENANCE APPLIED TO GLOBAL DSM AND
SYNCSORT LIBRARIES**

EXPLANATION: The maintenance level of the SyncSort for z/OS product is in conflict with the maintenance level of the global DSM (GDSM) subcomponent due to the incomplete application of one or more maintenance levels.

WER437A

[ddname] SPLIT INCOMPATIBLE WITH REPORT WRITING

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNames parameter. The SPLIT parameter and one or more report writing parameters have been specified for an OUTFIL group. The specified ddname is the first ddname of the OUTFIL group. SPLIT and report writing parameters are incompatible on the same OUTFIL control statement. Specifically, SPLIT cannot be specified on the same OUTFIL statement with HEADERn, TRAILERn, LINES, NODETAIL, and SECTIONS.

WER438A

[ddname] {INREC,OUTREC} - NONE OF THE FIND-CONSTANTS WAS MATCHED WITH THE CHANGE FIELD (p,l), CONTENTS OF INPUT FIELD IN HEX: xxxxxxxx

EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNames parameter. A CHANGE subparameter on an INREC, OUTREC or OUTFIL OUTREC control statement was specified without a NOMATCH option and the input field did not match any of the specified find-constants. p,l represents the position and length of the input field. xxxxxxxx is the hexadecimal representation of the input field.

WER440A

UNSUPPORTED OPERATING SYSTEM

EXPLANATION: Only OS/390 and later operating environments are supported by SyncSort for z/OS.

WER441A

ERROR IN CALLING LANGUAGE ENVIRONMENT SERVICE, RC = nnnn

EXPLANATION: A Language Environment service used to support LOCALE processing indicated a critical error in its feedback code. nnnn is the error message number representing the feedback code. For an explanation of this code, see the IBM publication *Debugging Guide and Run-Time Messages, SC26-4829*.

WER442A

INVALID CHARACTER IN COMPARE FIELD FOR ACTIVE LOCALE

EXPLANATION: INCLUDE/OMIT processing with the LOCALE function active detected a character that is not defined in the current locale. The invalid character could be in a CH field or in a character or hexadecimal constant compared to a CH field.

- WER443A** **INVALID CHARACTER IN CONTROL FIELD FOR ACTIVE LOCALE**
- EXPLANATION: Sort or merge processing with the LOCALE function active detected a character that is not defined in the current locale. The invalid character is in a CH sort or merge field.
- WER444I** **LOCALE PROCESSING USED FOR LOCALE nnnnnn**
- EXPLANATION: Indicates that LOCALE processing was in effect. nnnnnn (up to 32 characters) represents the name of the locale used.
- WER445A** **LOCALE PROCESSING CONFLICT**
- EXPLANATION: LOCALE processing has been used illegally. LOCALE processing cannot be used with an E61 exit. The LOCALE specification cannot be changed on a MAXSORT breakpoint restart.
- WER446A** **[ddname] INCLUDE/OMIT FORMATS INCOMPATIBLE FOR LOCALE PROCESSING**
- EXPLANATION: The ddname will be SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. LOCALE processing has been requested and a character (CH) to binary (BI) comparison was specified on an INCLUDE/OMIT statement or OUTFIL INCLUDE/OMIT parameter. CH to BI comparisons are not supported when using LOCALE processing.
- WER447B** **PHASE 3 VIRTUAL STORAGE REDUCED TO nnn BYTES FOR OPTIMAL PERFORMANCE**
- EXPLANATION: Phase 3 optimization has determined that a reduction in virtual storage is appropriate for an efficient execution. nnn is the amount of virtual storage used during phase 3. The total bytes used value in message WER164B indicates the virtual storage used during earlier phases of the sort execution.
- WER448I** **Y2 FORMAT CENTURY WINDOW IS FROM xxxx TO yyyy**
- EXPLANATION: One of the Y2x data formats has been used for a SORT/MERGE field, an INCLUDE/OMIT field or an INREC/OUTREC edit field. The starting year is xxxx and the ending year is yyyy for the century window used to process the fields.

WER449I

SYNCSORT GLOBAL DSM SUBSYSTEM ACTIVE

EXPLANATION: The SyncSort Global DSM (GDSM) subsystem was active during the execution of this SyncSort application.

WER450I

PARASORT USED

EXPLANATION: The PARASORT technique has been used for this execution.

WER451A

PARASORT TAPE LABEL ERROR VOL(vvvvvv) [CONCATENATION+0nnn]

EXPLANATION: The tape label on volume *vvvvvv* does not match the DCB characteristics of the SORTIN data set. This could happen because of changed record length, BLKSIZE or recording format. This situation is normally caused by overwriting some of the data in a multi-volume data set. The concatenation number indicates where in the SORTIN concatenation the volume in error may be found.

WER452I

PARASORT NOT USED: reason

EXPLANATION: The PARASORT feature has been disabled and the sort was performed using conventional SORTIN processing. The message indicates the reason for this action, which may be any of the following:

- **AUTOMATIC RETRY DISABLED** Automatic sort retry must be enabled for PARASORT to be used. It is required in the event that the condition identified in WER454A is encountered.
- **CONCATENATED SORTIN DEVICES DIFFER** Concatenated SORTIN devices must be the same device; that is, unit affinity must be specified.
- **INCOMPATIBLE CONDITIONS** The application may specify elements that cannot be used together. This problem can be caused by unusual sort key types, some feature combinations, or very long sort keys.
- **INPUT IS NOT TAPE** PARASORT requires input from tape devices. Input from any other source is not permitted.
- **INSUFFICIENT TAPE CHANNELS** At least two channel paths must be available to the tape drives being used to read the SORTPARn DDs. For a description of a technique to help insure that this requirement is satisfied, see the description of esoteric unit names in the PARASORT chapters of this manual and the *SyncSort for z/OS Installation Guide*.

- **NO SORTWORKS AVAILABLE** PARASORT requires sortwork space, which must be specified in the JCL or provided dynamically by DYNALLOC.
- **RETRY IN PROGRESS** PARASORT failed, but a retry is being attempted.
- **SORTIN IS A NULLFILE**
- **SORTIN IS ONLY A SINGLE VOLUME DATA SET** The SORTIN DD statement for PARASORT must define either a single multi-volume SORTIN data set or several concatenated tape data sets, which can be single or multi-volume. One single-volume data set is not permitted.
- **V(B)S DATA SETS NOT ALLOWED** VS and VBS data sets are not compatible with PARASORT.

WER453A

FOR PARASORT text

EXPLANATION: PARASORT failed and the sort application will not execute. The message text indicates the condition that caused the failure or the PARASORT requirement that was violated:

- **A SORTPAR2 DD STATEMENT IS REQUIRED**
- **DUPLICATE VOLUMES ON SORTIN DD NOT ALLOWED**
- **EQUALS MAY NOT BE SPECIFIED** If EQUALS is not specified on the SORT control statement or as a PARM, ensure it is not enabled by default. Pass NOEQUALS to disable EQUALS.
- **E15 EXITS MAY NOT BE SPECIFIED**
- **MAXSORT MAY NOT BE SPECIFIED**
- **PASSED SORTIN IS INVALID**
- **SEQNUM MAY NOT BE SPECIFIED ON INREC**
- **SKIPREC MAY NOT BE SPECIFIED**
- **SORTIN AND SORTOUT MUST BE DIFFERENT DATA SETS**
- **SORTIN GDG NOT ALLOWED**
- **SORTIN VOLUME SEQUENCE MAY NOT BE SPECIFIED**
The volume sequence number must be 1, the first volume. The number cannot be greater than 1.
- **SORTPAR DD STATEMENTS ARE REQUIRED**
- **SORTPAR(N)S MUST BE SEQUENTIALLY NUMBERED**
- **SORTPAR1 AND SORTIN DATA SET NAMES MUST BE THE SAME**
- **SORTPAR1 DISPOSITION MUST BE OLD**
- **SORTPAR1 UNIT MUST BE THE SAME AS THE SORTIN UNIT**
- **SORTPAR1-4 DEVICE TYPES MUST BE THE SAME AS THE SORTIN DEVICE TYPE**
- **SORTPAR2-4 CANNOT BE THE SAME AS THE SORTIN UNIT**
- **SORTPAR2-4 and SORTIN DATA SET NAMES MUST BE THE SAME**

- **SORTPAR2-4 DISPOSITION MUST BE (NEW,KEEP,KEEP)**
- **SORTPAR2-4 MUST SPECIFY DEFER ON THE UNIT PARAMETER**
- **SORTPAR2-4 MUST SPECIFY VOL=PRIVATE**
- **STOPAFT MAY NOT BE SPECIFIED**
- **THE DISPOSITION OF SORTIN IS INVALID** SORTIN data sets may not be temporary data sets. They also may not be NEW, passed or have PASS on their JCL definition.
- **DB2 MAY NOT BE SPECIFIED** The DB2 query function is not supported with a PARASORT.

WER454A

PARASORT SORTIN END OF FILE ENCOUNTERED BEFORE THE VOLUME LIST EXHAUSTED

EXPLANATION: The SORTIN volume list is supplied from either the catalog or specific list of volume serial numbers. The volume serial list must accurately reflect the volumes in the data set. If extra volumes are specified (as may happen if an old data set is rewritten with less data) this error message will be generated. A volume sequence number may not be specified.

WER455I

PARASORT CHANNEL CONTENTION - SORTPARn NOT USED

EXPLANATION: SORTPAR2-4 has no available channel path to send data other than a path that would conflict with a previously defined SORTPARn definition. This SORTPARn will not be used during the PARASORT execution. This message may occur more than once if there are multiple conflicting SORTPARn DD's.

WER456I

VISUAL SYNC SORT APPLICATION SUCCESSFULLY EXPORTED

EXPLANATION: A file that describes your application has been created and written to the VISUALEX DD statement for export to the PC component of Visual SyncSort. The operations defined by the control statements have **not** been performed.

WER457A

VISUALEX NOT SPECIFIED OR INVALID

EXPLANATION: The VISUALEX DD statement for export to the PC component of Visual SyncSort is either missing or its data set has been incorrectly defined. The file must have physical sequential or extended sequential organization or be a member of a partitioned data set or PDSE. The record format must be undefined (RECFM=U) or unspecified.

**WER458A MAINTENANCE LEVEL INSUFFICIENT TO PROCESS VISUAL
SYNCSORT SYSIN DATA SET**

EXPLANATION: The SYSIN data set created by the PC component of Visual SyncSort cannot be processed by SyncSort. This is due to an insufficient level of maintenance on the SyncSort library. A newer level of SyncSort may be required to process the SYSIN data set.

WER459A A VISUAL SYNCSORT APPLICATION MAY NOT text

EXPLANATION: Only qualified SyncSort applications may be exported to Visual SyncSort. The reason this application is ineligible is supplied in the message text.

**WER460I SORTIN DATA TRUNCATED DUE TO DCB BLKSIZE OVER-
RIDE**

EXPLANATION: An extended sequential data set used as input to a sort, merge or copy has had its DCB BLKSIZE overridden to a smaller value via a JCL specification. A physical block exceeding this overridden BLKSIZE specification was truncated to the smaller size during input processing.

ACTION: Confirm that this truncation is desired. If not, remove the BLKSIZE specification from the JCL.

WER461A SORTOUT/OUTFIL DATA SET CONTAINS NO DATA RECORDS

EXPLANATION: If the NULLOUT=RC16 parameter is in effect and the SORTOUT data set had no records written to it during processing, WER461A will be posted. If one or more non-SORTOUT OUTFIL specifications had the NULLOFL=RC16 parameter in effect and they had no records written to them, the WER461A will be posted. The WER405I message, which details the records written to each OUTFIL, will provide information on the OUTFIL(s) that caused the message to be generated. Note that an OUTFIL, FILES=OUT, or FNAME SORTOUT is controlled by NULLOUT only, and not by NULLOFL.

WER461I SORTOUT/OUTFIL DATA SET CONTAINS NO DATA RECORDS

EXPLANATION: If the NULLOUT=RC4 parameter is in effect and the SORTOUT data set had no records written to it during processing, WER461I will be posted. If one or more non-SORTOUT OUTFIL specifications had the NULLOFL=RC4 parameter in effect and they had no records written to them, WER461I will be posted. The WER405I message, which details the records written to each OUTFIL, will provide

information on the OUTFIL(s) that caused the message to be generated.

WER462A

OUTPUT LRECL DIFFERS FROM SORTOUT LRECL

EXPLANATION: The LRECL defined in the JCL for a non-OUTFIL SORTOUT differs from the SORTIN/SORTINnn LRECL or the internally processed record length when the SORTIN/SORTINnn LRECL is modified by features and the PAD and/or TRUNC parameters have been set to RC16 to disallow this.

ACTION: Remove the SORTOUT LRECL specification, allowing SyncSort to calculate the appropriate SORTOUT LRECL or modify the SyncSort control statements to build a record of the desired length as specified by the SORTOUT LRECL.

WER462I

OUTPUT LRECL DIFFERS FROM SORTOUT LRECL

EXPLANATION: If the application is a sort, merge, or copy, the LRECL defined in the JCL for a non-OUTFIL SORTOUT differs from the SORTIN/SORTINnn LRECL or the internally processed record length when the SORTIN/SORTINnn LRECL is modified by features and the SOPADGN and/or SOTRNGN installation options have been set to RC0 or RC4. In a BetterGener application, the LRECL defined in the JCL for SYSUT2 differs from the SYSUT1 LRECL or the internally modified record length when the SYSUT1 LRECL is modified by features and the SOPADGN and/or SOTRNGN installation options have been set to RC=0 or RC=4.

Fixed-length records will be padded to the SORTOUT LRECL (SYSUT2 LRECL in a SYNCGENR application) when the SORTOUT LRECL is greater than the SORTIN or internally processed record length.

Records will be truncated to the SORTOUT LRECL (SYSUT2 LRECL in a SYNCGENR application) when the SORTOUT LRECL is less than the SORTIN or internally processed record length.

ACTION: Verify that the padding or truncation that will be performed is desired for this application. Refer to the provided WER108I and WER110I messages that detail the input and output record lengths.

WER463A

ddname IS A LINEAR VSAM DATA SET

EXPLANATION: The ddname will be SORTIN, SORTOUT, SORTOFxx, SORTOFx or the ddname provided by an OUTFIL FNAMES parameter. SyncSort does not support an input or output file that is a linear VSAM data set.

WER464I**INCOMPLETE SPANNED RECORD FOUND**

EXPLANATION: An invalid spanned record segment has been found while processing the input records in a sort or merge application, and VLTEST=(,OFF4) has been specified to produce a warning. A return code of 4 will be issued if not overridden by a higher return code issued for another reason.

WER467I**DB2 QUERY TRIAL MODE SUCCESSFULLY EXECUTED**

EXPLANATION: A report of the record layout produced by the DB2 query contained in the SORTDBIN data set has been successfully produced. No other processing has occurred.

WER468A**DB2 QUERY SUPPORT ERROR: text**

EXPLANATION: The DB2 query operation failed and the sort or copy application will not execute. The message text indicates the condition that caused the failure or the DB2 query requirement that was violated.

- **MAXSORT MAY NOT BE SPECIFIED**
- **AN E15 EXIT MAY NOT BE SPECIFIED**
- **MERGE OPERATION MAY NOT BE SPECIFIED**
- **SKIPREC MAY NOT BE SPECIFIED**
- **SORTDBIN OPEN ERROR**
- **SORTDBIN CANNOT BE FOUND** The DB2 parameter has been specified, but the required SORTDBIN DD has not been provided.
- **NO SQL SELECT STATEMENT FOUND IN SORTDBIN**
- **INVALID COMMAND, ONLY SQL SELECT STATEMENT SUPPORTED** Only a SELECT or \$ELECT statement is valid in SORTDBIN. No other SQL operations are supported.
- **QUERY STATEMENT TOO LONG (MAX 32765 BYTES)**
- **CANNOT CONNECT TO DB2** DB2 is not started or the subsystem name specified on the DB2 EXEC parameter is incorrect.
- **CANNOT BIND PLAN** The user ID from which the job was submitted has insufficient authority to bind the plan with the SyncSort module. Submit the application from an ID that is allowed the BIND privilege.
- **CANNOT OPEN PLAN** Insufficient resources were available for DB2 to process the open request.
- **UNSUPPORTED DATA TYPE FOUND** Long fields (LONG VARCHAR and LONG VARGRAPHIC) and large object fields (BLOB, CLOB, and DBCLOB) are not supported.
- **UNKNOWN DATA TYPE FOUND**
- **SQL ERROR: SQLCODE=xxxx,SQLSTATE=yyyy** Where xxxx is the SQLCODE and yyyy is the SQLSTATE returned. Refer to

IBM publication DB2 Universal Database for OS/390 Messages and Codes (GC26-9011) for details on these return codes.

- **DB2 MODULES ARE NOT LINKED** The DB2 query facility of SyncSort for z/OS has not been installed during SyncSort installation. Contact your systems programmer for assistance.

SyncSort Statistical Record Facility Messages

The following messages are not controlled by the MSG or FLAG PARM and will appear only on the console.

WER500I SYNCSORT STATISTICS DATA SET NOW OVER xx PERCENT FULL

EXPLANATION: xx percent of space currently allocated on the SyncSort Statistics data set has been used.

WER501A SYNCSORT STATISTICS DATA SET NOW FULL - NO RECORD WRITTEN

EXPLANATION: The SyncSort Statistics data set did not have enough space for the SYNC SMF record.

PROC SYNC SORT Messages

**WER700A PROC SYNC SORT UNSUPPORTED FUNCTION.
{RETRY,NORETRY} IN EFFECT**

EXPLANATION: SyncSort's high performance technique could not be used during this invocation by PROC SYNC SORT - An Accelerator for SAS Sorting. This may be due to a small region size or the generation of an unsupported SyncSort statement syntax. If the RETRY option of PROC SYNC SORT is in effect, SyncSort for z/OS will be reinvoked using a less efficient E15-E35 interface. If the RETRY option is not in effect, the PROC SYNC SORT execution will be terminated.

**WER744A CONFLICT BETWEEN SYNC SORT AND PROC SYNC SORT
MAINTENANCE LEVELS, VERIFY LIBRARIES**

EXPLANATION: Maintenance has been applied to either PROC SYNC SORT or SyncSort for z/OS, but not to both when maintenance to both is required.

ACTION: Check the libraries containing PROC SYNC SORT and SyncSort for z/OS and apply the required level of maintenance to each.

- WER775A SAS I/O ERROR OCCURRED. CHECK SAS MESSAGE LOG DATASET**
- EXPLANATION: An I/O error occurred when a SAS routine attempted to access or update a SAS data set. A message indicating the actual nature of the problem should appear on the SAS message LOG data set.
- WER776A BLDL FAILURE FOR DDNAME SASLIB. RAISE REGION OR CHECK SASLIB ACCESS**
- EXPLANATION: When attempting to perform a BLDL for the library identified by the SASLIB DD statement, an error occurred. The error is due either to insufficient virtual storage or a permanent I/O error on the library.
- WER777A ERROR LOADING PROC SYNC SORT MODULE. CHECK PROC SYNC SORT INSTALL**
- EXPLANATION: The PROC SYNC SORT module could not be found in any of the libraries on the normal z/OS search chain or an error occurred while loading the module.
- WER778A UNEQUAL MAINTENANCE APPLIED TO PROC SYNC SORT AND SYNC SORT LIBRARIES. DATA=hexdata**
- EXPLANATION: The maintenance level of the PROC SYNC SORT product is in conflict with that of the SyncSort for z/OS product due to the incomplete application of one or more maintenance levels. The hexadecimal data, if printed, indicates which maintenance fixes were incompletely applied.
- WER779I THE PERFORMANCE OF THIS SORT COULD BE SIGNIFICANTLY IMPROVED THROUGH THE USE OF THE PROC SYNC SORT PRODUCT**
- EXPLANATION: PROC SYNC SORT - An Accelerator for SAS Sorting is a high performance replacement for the SAS-provided procedure PROC SORT. When SyncSort is invoked with the PROC SYNC SORT product instead of through the interface supplied by SAS, significant performance improvements result. For more information, call SyncSort for z/OS Product Services.

License Key Messages

The following are the messages directly related to the use of a SyncSort for z/OS license key.

WER900A

**SYNCSORT 1.1 TPF_{xx} IS NOT LICENSED FOR SERIAL sssss,
TYPE mmmm mmm, MSU CAPACITY ccccc.**

or

**SYNCSORT 1.1 TPF_{xx} IS NOT LICENSED FOR SERIAL sssss,
TYPE mmmm, VERSION CODE vv.**

EXPLANATION: No valid license key for use on the specified machine was found, and the grace period for this error, noted by the WER903I warning message, has expired. A key must contain the correct information for both the serial number and the full model number. License keys are specified either in the KEY parameter of the SYNCMAC installation options macro, or included in a data set whose name is specified in the KEYDSN parameter of SYNCMAC.

ACTION: Execute the SYNCLIST program on the system where this message is occurring. Ensure that either the SYNCMAC KEY parameter or the data set named in the KEYDSN parameter has provided a valid key for this machine. If you require further assistance, contact SyncSort for z/OS Product Services with the SYNCLIST output available for reference.

WER901I

****WARNING** SYNCSORT 1.1 TPF_{xx} WILL EXPIRE IN nnn
DAYS**

EXPLANATION: The provided license key for this machine is only valid for the next nnn days. After that time, WER902A will be issued, and SyncSort will not execute.

ACTION: Contact the systems programmer in charge of SyncSort maintenance, or execute the SYNCLIST program on the system where this message is occurring and contact SyncSort for z/OS Product Services.

WER902A

SYNCSORT 1.1 TPF_{xx} HAS EXPIRED

EXPLANATION: The provided license key for this machine is no longer valid because the expiration date has passed. SyncSort will no longer execute.

ACTION: Contact the systems programmer in charge of SyncSort maintenance, or execute the SYNCLIST program on the system where this message is occurring and contact SyncSort for z/OS Product Services.

WER903I

**SYNCSORT 1.1 TPF_{xx} IS NOT LICENSED FOR SERIAL sssss,
TYPE mmmm mmm, MSU CAPACITY ccccc.**

or

**SYNCSORT 1.1 TPF_{xx} IS NOT LICENSED FOR SERIAL sssss,
TYPE mmmm, VERSION CODE vv.**

**SYNCSORT WILL STOP WORKING IN nnn DAYS UNLESS A
VALID KEY IS INSTALLED.**

EXPLANATION: No valid license key for use on the specified machine was found. License keys are specified in the KEY parameter of the SYNCMAC installation options macro, or included in a data set whose name is specified in the KEYDSN parameter of SYNCMAC.

SyncSort will allow sort processing to continue by issuing WER903I during a grace period after this error is first encountered. This will provide sufficient time to correct the problem by installing a valid key for this machine. If the grace period ends before a valid key is made available, WER900A will be issued and sort processing will terminate.

ACTION: Execute the SYNCLIST program on the system where this message is occurring. Ensure that either the SYNCMAC KEY parameter or the data set named in the KEYDSN parameter has provided a valid key for this machine. If you require further assistance, contact SyncSort for z/OS Product Services with the SYNCLIST output available for reference.

WER904I

**SYNCSORT 1.1 TPF_{xx} KEYUPDATE SUCCESSFUL;
xxxxxxxxxxxxxxxxxxxxx SELECTED**

EXPLANATION: The KEYUPDATE parameter was specified, and SyncSort has successfully obtained a valid license key denoted by xxxxxxxxxxxxxxxxxxxx from SyncSort's key data set. The name of the data set was specified in the KEYDSN parameter of the SYNCMAC installation options macro.

WER905A

SYNCSORT 1.1 TPF_{xx} KEYUPDATE FAILURE: reason

EXPLANATION: The KEYUPDATE parameter was specified, but SyncSort was unable to obtain a valid license key from SyncSort's key data set due to the specified reason. Possible reasons for this failure are:

1. The KEYDSN parameter of SYNCMAC was not specified when SyncSort was installed. KEYDSN, and not the KEY parameter, must be specified with the name of SyncSort's key data set when using the KEYUPDATE facility.

2. SyncSort was unable to dynamically allocate and/or read SyncSort's key data set. This can happen if you were editing the data set at the time of the KEYUPDATE run, or if the data set was not allocated as a fixed length 80-byte file.
3. No valid license key was found in SyncSort's key data set.
4. The SyncSort SVC was not available. SyncSort requires use of its SVC to perform the update.

ACTION: Ensure that the KEYDSN parameter has been correctly specified and that the data set is accessible and contains a valid license key. Also verify that the SyncSort SVC has been properly installed. If you require further assistance, execute the SYNCLIST program on the system where this message is occurring and contact SyncSort for z/OS Product Services with the SYNCLIST output available for reference.

WER906I

**INVALID KEY DATA SET RECORD:
invalid record text**

EXPLANATION: One or more invalid records were found in the license key data set when performing KEYUPDATE. The first invalid record is displayed in the message text. Only comment statements, key statements and valid PARMs statements are permitted. All invalid statements are ignored.

ACTION: Correct any errors in the key data set record that was displayed in the message text and rerun the KEYUPDATE application.

WER907I

**SYNCSORT EXPIRING LICENSE KEY WARNING MESSAGE
{ENABLED,DISABLED}
or
SYNCSORT INVALID LICENSE KEY WARNING MESSAGE
{ENABLED,DISABLED}**

EXPLANATION: These KEYUPDATE messages document whether SyncSort may issue certain license key warning messages. The default is for SyncSort to issue either the WER901I expiring license key warning message or the WER903I invalid license key warning message when applicable. During KEYUPDATE, a PARMs statement read from the key data set can disable the issuance of either of these messages. The WER907I message is intended to alert you that these warning messages may no longer be posted, though the warning period countdowns will continue. During the last seven days before the warning period ends, SyncSort will issue the warning messages **regardless** of whether they have been disabled. This is done to try to prevent termination of all SyncSort applications with either WER902A or WER900A.

ACTION: No action is required if both of these warning messages are enabled and you have a valid license key that is not expiring. If you do not have a valid key or if your key is expiring, call SyncSort for z/OS Product Services as soon as possible to obtain a new license key and rerun the KEYUPDATE procedure using the new key. If any of the messages had been disabled, either remove the PARMs statement or set the warning message parameters to ON to re-enable the issuance of license key warning messages.

Troubleshooting Abends

Troubleshooting with WER999A UNSUCCESSFUL SORT

WER999A indicates that an error condition occurred, preventing the successful completion of the sort. *This message does not necessarily mean that SyncSort was responsible for the error.* If, for example, the error is in the COBOL Input or Output Procedure of an invoked sort, WER999A will appear. WER999A indicates that SyncSort got control **after** the error, printing this SyncSort message.

The documentation accompanying WER999A varies with the error involved. It may consist of a standard system dump (SYSUDUMP or SYSABEND) and/or a SyncSort-generated SNAP dump. The SyncSort SNAP is formatted very much like a SYSUDUMP. In debugging the SNAP, care must be taken to avoid reliance on the PSW AT ENTRY TO SNAP and the general registers. (A SNAP dump produced through the SyncSort DEBUG PARM or with a W-abend (i.e., WER999A UNSUCCESSFUL SORT xxxW) is only useful to a sort analyst at SyncSort for z/OS Product Services. See "What to Do Before Calling SyncSort for z/OS Product Services".)

SyncSort Internal Abend

A W-type abend code indicates that program termination was forced by an error condition internally detected by SyncSort; the problem cannot be resolved by the user. See "What to Do Before Calling SyncSort for z/OS Product Services", below.

User-Issued Abend

If any of the following abend codes appear in WER999A, it may indicate a SyncSort error (in which case, see "What to Do Before Calling z/OS Product Services"). These are the *only* U-type abend codes that SyncSort issues; any U-type abend code which is not on this list indicates an error in a user-written exit routine or invoking program. Note that WER999A gives the abend code in hexadecimal.

User Abend 4093

User abend 4093 (RC=1C) is related to LOCALE processing. This abend is issued from the LE/370 environment when the REGION is not large enough. Increase the REGION by 1 megabyte and resubmit the application.

User-Type Abend Codes Issued by SyncSort			
Decimal	Hexadecimal	Decimal	Hexadecimal
10	A	1103	44F
16 *	10 *	1104	450
69	45	1107	453
936	3A8	1108	454
999 **	3E7 **	1110	456
1024 ***	400 ***	1111	457
1025	401	1112	458
1026	402	1115	45B
1027	403	1116	45C
1028	404	1117	45D
1030	406	1118	45E
1031	407	1119	45F
1032	408	1120	460
1050	41A	1121	461
1051	41B	1122	462
1052	41C	1123	463
1053	41D	1124	464
1054	41E	1125	465
1060	424	1126	466
1061	425	1127	467
1070	42E	1128	468
1071	42F	1129	469
1072	430	1131	46B
1073	431	1188	4A4
1074	432	1189	4A5
1075	433	1190	4A6
1076	434	1191	4A7
1077	435	1192	4A8
1078	436	1193	4A9
1079	437	1194	4AA
1086	43E	1197	4AD
1087	43F	1198	4AE
1088	440	2048	800
1102	44E	2049	801
		2081	821

* The RC16=ABE option is specified and there has been a critical error.
** The IOERR=ABE option is specified and there has been an I/O error.
*** This is most commonly caused by the release of SyncSort's SVC not matching the release of the SyncSort module.

What to Do Before Calling SyncSort for z/OS Product Services

All pertinent information (listings, dumps, etc.) should be available for easy reference when calling SyncSort for z/OS Product Services. For error conditions producing the WER999A message, the system dump and/or SyncSort SNAP dump will prove helpful to a SyncSort analyst. For other conditions cited with an "A" class message (e.g., WER039A

INSUFFICIENT VIRTUAL STORAGE), additional diagnostic information may be required - a diagnostic SNAP dump can be produced by passing the DEBUG PARM in the \$ORTPARM DD statement or (for a JCL sort) in the // EXEC statement. When using DEBUG, supply a SPYSET or SYSUDUMP DD statement to define an appropriate SYSOUT data set for the dump.

In the United States and Canada, please call SyncSort for z/OS Product Services directly at (201) 930-8260. The address is:

Syncsort Incorporated
SyncSort for z/OS Product Services
50 Tice Boulevard
Woodcliff Lake, New Jersey 07677
FAX: (201) 930-8284
E-mail: zos_tech@syncsort.com

Index

Symbols

\$ORTPARM Statement 4.1, 4.11–4.14, 5.1, 6.2, 7.10, 7.30, 12.5
\$ORTPARM, for Century Window 4.13
\$ORTPARM, with CENTWIN 4.13
&DATE 2.106
&DATENS=(xyz) 2.71, 2.75, 2.106
&DATE_x 2.20, 2.28–2.29
&DATE_x(c) 2.20, 2.28–2.29
&DATE_xP 2.20, 2.28–2.29
&TIME=(hp) 2.106
&TIMENS=(tt) 2.72, 2.76

A

AC Format 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
ACS 2.146, 5.14
ALTSEQ Control Statement 2.13–2.14, 6.9, 7.57
AMODE 6.12
AND Operator 2.21, 3.3
ANSI Control Characters 2.83–2.85
AQ Format 2.13, 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
ASL Format 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
Assembler Programs 1.1
 Invoking SyncSort from 6.1, 6.18
AST Format 2.19, 2.29, 2.38, 2.102, 2.105, 2.131

ATTACH Macro 6.2, 7.9, 7.29
Authorization Messages 16.66
Averages, Example 3.47
AVG 2.78

B

B Messages
 See BMSG Option
BALANCE Option 5.3, 13.4
BALN Option 5.2, 5.34, 6.7, 6.10, 12.3–12.5, 13.8
BatchPipes/MVS 2.68, 4.5, 4.7–4.8
BI Format 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
Binary Zeros, Insertion of 2.90–2.104, 3.18–3.20
Bit Level Comparison 2.20
Bit Level Logic 2.20–2.21, 2.31–2.33
BKPTDSN Option (MAXSORT) 5.1, 9.10
BLKSIZE Parameter 4.4, 5.5, 5.25, 5.28–5.29
Block Size 5.28–5.29
BMSG Option 5.3, 5.7
BSAM 4.5, 4.8
BUFOFF Parameter 4.4

C

C E15 7.19–7.27
C E35 7.43–7.51

- C Exits 7.19–7.27, 7.43–7.51
- C Programs 1.1
- Century Window Processing 2.41–2.51, 2.100–2.102, 2.121, 2.124, 2.134–2.144
- Century Window, with \$ORTPARM 4.13
- CENTWIN Option 2.41–2.51, 2.134–2.144, 5.3, 5.7–5.9
- CENTWIN Processing, with OUTREC 2.121, 2.124
- CENTWIN, with \$ORTPARM 4.13
- CH Format 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
- CHANGE 2.113
- Channel Separation 13.14
- Checkpoint-Restart 4.14, 13.10–13.14
 - Automatic 13.12–13.13
 - Deferred 13.13–13.14
- CKPT/CHKPT Parameter (MERGE) 2.52
- CKPT/CHKPT Parameter (SORT) 2.145
- CLO Format 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
- CMP Option 5.3, 13.3
- CMP=CLC 2.41, 5.3, 13.3
- CMP=CPD 2.41, 5.3, 13.3
- COBEXIT Option 5.3, 5.10, 7.10, 7.31
- COBEXIT=COB1 5.3
- COBEXIT=COB2 5.3
- COBOL 35
 - Writing Exit in OS/V5 COBOL 7.31
 - Writing Exit in VS COBOL II or COBOL/370 7.31
- COBOL E15 5.14, 7.10–7.19
 - DATA DIVISION 7.13
 - ENVIRONMENT DIVISION 7.13
 - EXIT-STATUS Codes 7.14
 - Fixed-Length Records 7.11, 7.15–7.16
 - IDENTIFICATION DIVISION 7.13
 - LINKAGE SECTION 7.10–7.13
 - PROCEDURE DIVISION 7.13
 - RETURN-CODE Codes 7.14
 - Variable-Length Records 7.12–7.13, 7.17–7.19
 - WORKING-STORAGE SECTION 7.13
 - Writing Exit in OS/V5 COBOL 7.10
 - Writing Exit in VS COBOL II or COBOL/370 7.10
- COBOL E35 5.15, 7.30–7.42
 - DATA DIVISION 7.34
 - ENVIRONMENT DIVISION 7.34
 - EXIT-STATUS Codes 7.35
 - Fixed-Length Records 7.32–7.33, 7.37–7.39
 - IDENTIFICATION DIVISION 7.34
 - LINKAGE SECTION 7.31–7.34
 - PROCEDURE DIVISION 7.35
 - RETURN CODE Codes 7.35–7.36
 - Variable-Length Records 7.33–7.34, 7.40–7.42
 - WORKING STORAGE SECTION 7.34
- COBOL Exits 7.5, 7.10–7.19, 7.30–7.42
 - COBOL/370 7.10, 7.31
 - OS/V5 COBOL 7.10, 7.31
 - See COBEXIT Option
 - VS COBOL II 7.10, 7.31
- COBOL Programs 1.1, 6.1, 13.2
- COBOL, and Century Window 4.13
- COBOL/370
 - See COBOL E35
 - See COBOL Exits
- CODE Parameter (ALTSEQ) 2.13
- Coding Conventions 4.3–4.4
- Collating Sequence 2.13–2.14, 2.38, 7.57
- Combining Records in a File 3.11
- COMMAREA Option 5.3, 5.11, 7.4
- Communication Area 5.11
- Communication Area for Exits 7.4
- Comparing Fields 2.17–2.34, 2.134, 3.2–3.11
 - Bit Level Criteria 2.20–2.21
 - Constants 2.19
 - Field to Constant Comparison 2.26
 - PD and ZD Field Comparison 2.41, 2.134
 - Rules for Specifying Fields 2.41
- Concatenating Input Data Sets 4.5
- COND Parameter (INCLUDE/OMIT) 2.17–2.34
- Control Statement Syntax 2.8–2.11
- Control Statements 2.1–2.152
 - ALTSEQ 2.13–2.14, 6.9, 7.57
 - Coding in Invoked Programs 6.2
 - Comments in 2.10
 - Continuation of 2.10–2.11
 - DEBUG 6.9
 - Defaults 2.2
 - END 2.15, 6.3, 12.6
 - for MAXSORT 9.10
 - for Tape Sort 12.6
 - INCLUDE/OMIT 2.16–2.34, 3.2–3.6, 5.16, 5.30, 6.9, 13.2
 - INREC 2.35–2.36, 3.6–3.11, 6.9, 13.2–13.3
 - Labels in 2.11
 - MERGE 2.35, 2.37–2.53, 5.14, 5.17, 5.30, 6.2,

6.8, 7.56–7.57
 MODS 2.54–2.57, 5.14, 6.3, 6.5, 6.9, 12.6
 Notational Conventions 2.11
 OMIT 2.58
 OUTFIL 2.35, 2.59–2.87, 3.27–3.55, 6.2, 6.9,
 13.2
 OUTREC 2.35, 2.88–2.124, 3.14, 6.9,
 13.2–13.3
 Performance Considerations 13.2–13.3
 Processing Sequence 2.6–2.8
 RECORD 2.125–2.128, 6.3, 12.6
 Requirements for Disk Sort 2.5
 Requirements for MAXSORT 2.5
 Requirements for Tape Sort 2.5
 Rules for Specifying 2.8–2.11
 See \$ORTPARM Statement
 See Job Control Language
 See SYSIN Statement
 SORT 2.35, 2.129, 5.14, 5.17, 5.30, 6.2, 7.52,
 7.56–7.57, 12.6
 Specifying Field Formats in 2.9
 Specifying Field Lengths in 2.9
 Specifying Field Positions in 2.9
 Specifying Parameters in 2.8–2.9
 SUM 2.35, 2.149–2.152, 3.11, 5.15–5.16, 6.9,
 7.27, 13.2
 Summary of Functions 2.1
 Summary of Parameters and Defaults 2.2
 Summary of the Chapter 1.7
 Use in Invoked Applications 6.3–6.4
 CONVERT Parameter (OUTFIL) 2.67
 CONVERT Parameter (OUTREC) 2.116
 Converting Data 2.93, 2.95–2.96, 2.102–2.107,
 3.20–3.27, 5.8
 Format 2.93–2.95
 Converting Fixed-Length Records to Variable-
 Length Records 2.66
 Converting SMF Formats 2.102
 Converting Variable-Length Records 2.67, 2.90,
 2.116–3.27
 Converting Year Data 2.100–2.102
 Copy
 Creating Input Data Sets for 4.5
 Flow of 8.1–8.6
 Copying 1.3
 Phases of 1.3
 CORE Option 5.3, 5.12, 13.4–13.6
 COUNT 2.79
 COUNT15 2.79
 CPU Option 5.3, 5.7, 13.4
 CRCX Option 5.34, 6.7, 6.10
 CSF Format 2.19, 2.29, 2.102, 2.105
 CSL Format 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 CST Format 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 CTO Format 2.19, 2.29, 2.38, 2.40, 2.102, 2.105,
 2.131, 2.133
 Cultural Environment 1.5, 2.37, 2.129

 D
 DASD Data Set 4.4
 Data Set Placement 13.14
 Data Utility 3.1–3.55
 Duplicate Records 3.11–3.13
 Examples, Index to 3.2
 Features of 1.3, 3.1
 Input Record Selection 3.2–3.11
 Input Records, Selection of Relevant Fields
 3.6–3.11
 Output Files, Multiple 3.53–3.55
 Output Records
 Converting Data 3.23–3.25
 Converting Data to Hexadecimal Format
 3.22–3.23
 Converting Data to Readable Form
 3.20–3.22
 Editing Data 3.23–3.25
 Editing of 3.14–3.27
 Formatting Data Fields 3.25–3.27
 Inserting Binary Zeros 2.90–2.104,
 3.18–3.20
 Inserting Blanks 3.16–3.18
 Reordering Field Positions 3.14–3.18
 Output Reports
 Counting Data Records 3.49–3.53
 Headers and Trailers for 3.32–3.41
 Sectioning of 3.30–3.32
 Totaling and Subtotaling Data 3.41–3.46
 Sample Applications, Summarized 3.2
 Summary of the Chapter 1.7
 DATE (&DATE) 2.70, 2.75, 2.105
 DB2 Query Support 11.1
 DCB Information 2.125
 DCB Parameter 4.4, 5.5, 5.25, 5.28
 DD Statements 4.4–4.16, 9.4–9.10, 10.2–10.3,
 12.3–12.5
 \$ORTPARM Statement 4.11–4.14, 6.2, 12.5
 Coding Conventions 4.3–4.4
 JOBLIB Statement 4.4

Parameters

- AMP 4.4
- BLKSIZE 4.4
- BUFND 4.4
- BUFNI 4.4
- BUFOFF 4.4
- BUFSP 4.4
- DCB 4.4
- DISP 4.4
- DSNAME/DSN 4.4
- LABEL 4.4
- LRECL 4.4
- OPTCD 4.4
- RECFM 4.4
- SPACE 4.4
- UNIT 4.4
- VOLUME/VOL 4.4

See Exit Programs, Link-editing

- SORTBKPT Statement 9.6–9.7
- SORTCKPT Statement 4.14, 13.2
- SORTIN Statement 4.5–4.7, 6.2
- SORTINn Statement 4.7
- SORTINnn Statement 6.2
- SORTLIB Statement 6.2, 12.3
- SORTMODS Statement 4.15
- SORTOFx Statement 4.8, 6.2
- SORTOFxx Statement 4.8, 6.2
- SORTOU00 Statement 9.7–9.9
- SORTOUT Statement 4.8, 6.2
- SORTPARn Statement 10.5–10.7
- SORTWKnn Statement 4.9–4.11, 6.2, 12.4–12.5
- SORTXSUM 4.8
- STEPLIB Statement 4.4
- Summarized for Invoked Sort/Merge 6.1–6.2
- SYSIN Statement 4.11
- SYSLIN Statement 4.15
- SYSLMOD Statement 4.15
- SYSOUT Statement 4.4, 6.2
- SYSPRINT Statement 4.15
- SYSUT1 Statement 4.16

- ddname 4.1, 9.5, 10.3, 11.3, 12.3
- DEBUG Control Statement 6.9
- DEBUG Option 5.3, 5.13

Defining Output Data Sets

- See SORTOUT Files
- See SORTOUT Statement

Defining Work Areas

- See SORTWKnn Statement

Deleting Trailing Bytes From Fixed-Length Records 2.67

- Device Separation 13.14
- DIAG Option 5.3, 5.13, 6.7, 6.10
- Disk Sort 4.1, 13.8
 - Control Statements 2.1–2.152
 - DD Statements for 4.1–4.16
 - Device Types 4.9
 - EXEC Statement for 4.2
 - PARM Options 12.1
 - PARMS Options 5.1–5.34
 - Performance Considerations 13.1
- DISP Parameter 4.4, 5.28
- DSN Parameter
 - See DSNAME Parameter
- DSNAME Parameter 4.4
- DT1 2.102
- DT2 2.102
- DT3 2.102
- Duplicate Records 3.11–3.13
- DYNALLOC Option 4.1, 5.3, 5.13–5.14
- DYNALLOC Parameter (SORT) 2.145–2.146
- DYNATAPE Option (MAXSORT) 4.1, 5.1, 9.10, 9.14–9.16

E

- E10 2.56
- E11 2.56, 7.5
- E14 4.10, 5.16, 7.7–7.8, 9.13
- E15 2.56, 5.16, 7.7–7.10, 7.58, 9.13, 12.6, 12.9, 14.4
 - See COBOL E15
- E15 Option 5.3, 5.14, 7.10
- E15=COB 5.4
- E16 4.10, 7.51, 9.13
- E17 7.52
- E18 7.52–7.55
- E20 2.56
- E21 2.56, 7.5
- E25 7.7, 7.27–7.28, 9.13
- E27 7.52
- E30 2.56
- E31 2.56, 7.5
- E32 2.52, 6.2, 7.5, 12.6
- E35 4.2, 7.7, 7.28–7.30, 7.58, 9.13, 12.6, 12.9
 - See COBOL E35
- E35 Option 5.3, 5.15
- E35=COB 5.4
- E37 7.52

- E38 7.52–7.55
 - E39 7.52, 7.55–7.56
 - E61 2.56, 7.56–7.58, 9.13
 - EDIT 2.79
 - Edit Patterns 2.107–2.113, 3.25
 - EDIT Subparameter 2.107
 - Editing Data 2.97–2.116, 3.14–3.27
 - Editing Masks 2.79, 2.109–2.113
 - ELAP Option 5.3–5.4, 5.7, 5.15, 13.4
 - END Control Statement 2.15, 12.6
 - ENDREC Parameter (OUTFIL) 2.64
 - Equal-keyed Records 2.52, 2.146, 2.149, 5.15
 - EQUALS Option 5.4, 5.15, 13.3
 - EQUALS/NOEQUALS Parameter (MERGE) 2.52
 - EQUALS/NOEQUALS Parameter (SORT) 2.146
 - Esoteric Names 10.7
 - EXEC Statement 4.2, 9.4, 10.2, 11.2, 12.2
 - for Disk Sort 4.2
 - for MAXSORT 4.3, 9.4
 - for PARASORT 10.2, 11.2
 - for Tape Sort 4.3, 12.2–12.3
 - Exit Conventions 7.3
 - Exit Programs 2.54–2.57, 7.1–7.60
 - Acting on Insufficient Storage 7.51
 - Adding Records 7.28–7.30
 - Analyzing Sort Input File 7.8–7.10
 - Changing Records 7.7–7.8, 7.27–7.30
 - Checking Labels 7.52–7.55
 - Closing Exit Data Sets 7.52
 - Communication Area 7.4
 - Creating Input Records 7.5–7.10
 - Creating Sort Input File 7.8–7.10
 - Definition of 7.1
 - Deleting Records 7.7–7.8, 7.27–7.30
 - End-of-File Routines 7.52–7.55
 - for Invoked Merge 7.5
 - Identified in MODS Control Statement 7.3
 - Link-editing 7.3
 - Link-editing at Execution Time
 - DD Statements Required for 4.15–4.16
 - Loading into Main Storage 7.3
 - MAXSORT 9.13
 - Modifying Collating Sequence 7.56–7.58
 - Phases 7.1–7.2
 - Preparing for Other Exit Programs 7.5
 - Processing Read Errors 7.52–7.55
 - Processing Write Errors 7.52–7.55
 - Program Labels 7.1
 - Register Conventions 7.4
 - Revising Sort Input File 7.8–7.10
 - Summarizing Records 7.7–7.8, 7.27–7.28
 - Summary of Tasks Performed 7.2
 - Summary of the Chapter 1.7
 - VSAM Processing 7.52, 7.55
 - with Disk Sort 2.56
 - with MAXSORT 2.56, 9.13
 - with PARASORT 2.56
 - with Tape Sort 2.56, 12.6, 12.9
 - Exit Programs, Link-editing 4.16
 - Exit-Name Parameter (MODS) 2.54–2.56
 - EXTCOUNT 5.4, 5.16
 - EXTCOUNT Option 5.3
- F**
- FI Format 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 - Field Format Codes 2.38–2.41, 2.130–2.133
 - AC 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 - AQ 2.13, 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 - ASL 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 - AST 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 - BI 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 - CH 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 - CLO 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 - CSF 2.19, 2.29, 2.102, 2.105
 - CSL 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 - CST 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 - CTO 2.19, 2.29, 2.38, 2.40, 2.102, 2.105, 2.131, 2.133
 - FI 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 - FL 2.38, 2.131
 - FS 2.19, 2.29, 2.102, 2.105
 - List of Valid Formats 2.38–2.41, 2.131–2.133
 - LS 2.19, 2.29, 2.102, 2.105
 - OL 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 - OT 2.19, 2.29, 2.40, 2.102, 2.105, 2.131, 2.133
 - P2ID 2.101
 - P2IP 2.101
 - PD 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 - PD0 2.19, 2.29, 2.39, 2.44, 2.102, 2.105, 2.132, 2.138
 - TS 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 - Y2B 2.19, 2.29, 2.39, 2.42, 2.101–2.102, 2.105, 2.132, 2.134
 - Y2C 2.19, 2.29, 2.39, 2.42, 2.101–2.102, 2.105, 2.132, 2.135
 - Y2D 2.19, 2.29, 2.40, 2.43, 2.101–2.102,

2.105, 2.132, 2.135
 Y2P 2.19, 2.29, 2.40, 2.43, 2.101–2.102, 2.105,
 2.132, 2.136
 Y2S 2.19, 2.29, 2.40, 2.44, 2.101–2.102, 2.105,
 2.132, 2.136
 Y2Z 2.19, 2.29, 2.40, 2.42, 2.101–2.102, 2.105,
 2.133, 2.135
 ZD 2.19, 2.29, 2.40, 2.102, 2.105, 2.131, 2.133
 Fields 2.88–2.124
 Binary 2.16, 2.20–2.21, 2.130
 Bit Level Comparison 2.20
 Comparison of 2.17–2.34, 2.134, 3.2–3.11
 Constants 2.19
 Format 2.38–2.41, 2.64–2.98, 2.130–2.133
 Insertion of Binary Zeros 3.18–3.20
 Insertion of Blanks 3.16–3.18
 Length 2.130
 Position 2.95–2.96, 2.102, 2.130, 5.8
 Reordering 3.14–3.18
 Rules for Specifying 2.41, 2.133
 Selection of 3.6–3.11
 Specifying Format 2.9
 Specifying Length 2.9, 3.6
 Specifying Position 2.9, 3.6
 Substring Comparison 2.30
 FIELDS Parameter (INREC) 2.35
 FIELDS Parameter (MERGE) 2.37–2.52
 FIELDS Parameter (OUTREC) 2.90–2.116
 FIELDS Parameter (SORT) 2.129–2.144
 FIELDS Parameter (SUM) 2.149–2.150
 FIELDS Subparameters 2.90
 Operator 2.93
 FIELDS=COPY 2.51, 2.53, 2.144
 FIELDS=NONE 2.149
 File Size
 See FILSZ Option
 FILES Parameter (MERGE) 2.52
 FILES Parameter (OUTFIL) 2.62, 3.53–3.55
 FILSZ Option 5.4, 5.16–5.17
 FILSZ Parameter (SORT) 2.147
 Fixed-Length Records 2.9, 2.41, 2.90, 2.116, 2.125,
 7.11, 7.15–7.16
 Maximum Length 4.5
 FL Format 2.38, 2.131
 FLAG Option 5.4, 5.17, 6.10
 Flow of the Sort 8.1–8.6
 FNames Parameter (OUTFIL) 2.63
 FORMAT 2.37–2.38, 2.64, 2.129–2.130,
 2.148–2.149, 2.152
 SORT 2.148
 FORTRAN Programs 1.1, 6.1
 FS Format 2.19, 2.29, 2.102, 2.105
 FTOV Parameter (OUTFIL) 2.66
 Full-date formats 2.45, 2.139

 G
 Generating Run-Time Constants 2.104
 GETMAIN 2.55, 6.3
 Group, OUTFIL 2.62–2.63

 H
 HBSI Option 4.5, 5.4, 5.17
 HBSO Option 4.9, 5.4, 5.18
 HEADER1 / HEADER2 Parameter (OUTFIL)
 2.68–2.73, 3.32
 HFS 4.5, 4.7–4.8
 Hiperbatch 5.18
 Hiperbatch Processing 5.17–5.18
 HISTOGRM 2.127–2.128, 5.19, 7.8, 14.1–14.13
 Control Parameters 14.2–14.4
 Executing through E15 14.4–14.6
 Job Control Language 14.4
 Messages 14.9–14.13
 Output Samples 14.7–14.8
 Summary of the Chapter 1.8

 I
 INCLUDE/OMIT Control Statement 2.16–2.34,
 3.2–3.6, 5.16, 5.30, 6.9, 13.2
 INCLUDE/OMIT Parameter (OUTFIL)
 2.63–2.64
 INCORE Option 5.4, 5.18, 13.6
 Incore Sort 2.52, 2.145, 2.151, 5.18, 8.1, 13.6
 INCORE=OFF 5.4
 INCORE=ON 5.4
 Input Data Sets
 Concatenation of 4.5
 Input Records
 Selection of 3.2–3.11
 INREC Control Statement 2.35–2.36, 3.6–3.11,
 6.9, 13.2–13.3
 Installation Guide 1.9
 Installation Options

- Precedence Rules 5.2
- Intermediate Storage 4.9–4.10, 12.4
- Invoking SyncSort from a Program 6.1–6.18
 - Assembler Programs 6.2
 - DD Statements 6.1
 - Macro Instructions for Invoking SyncSort from an Assembler Program
 - ATTACH 6.2
 - LINK 6.2
 - LOAD 6.2
 - XCTL 6.2
 - MAXSORT 9.13
 - Performance Considerations 6.1
 - Restrictions on Control Statements 6.3–6.4
 - Tape Sort 12.9–12.11
- IO Option 5.3–5.4, 5.7, 5.18, 13.4
- IOERR Option 5.4, 5.18, 6.10
- IOERR=ABE 5.4

J

- JCL
 - for PARASORT 10.2
 - See Job Control Language
- Job Control Language 4.1, 4.17
 - Control Statement Examples
 - Copy without Exit Routines 4.25–4.26
 - Merge without Exit Routines 4.23–4.24
 - Multiple Output Files 4.32–4.33
 - Sort with Exit Routine to be Link-edited 4.29–4.31
 - Sort with Link-edited Exit Routine 4.27–4.28
 - Sorts without Exit Routines 4.22
 - DD Statement 4.1
 - Elements of 4.1
 - EXEC Statement 4.1, 5.1, 6.1
 - for HISTOGRM 14.4
 - for MAXSORT 9.4, 9.13
 - for Tape Sort 12.7–12.8
 - Initiating SyncSort 4.11, 6.1
 - JOB Statement 4.1
 - Sorts without Exit Routines 4.17
 - Summary of the Chapter 1.7
- JOBLIB Statement 4.4

K

- KEY Messages 16.66

L

- L6 Option 5.4, 5.19, 14.1
- L7 Option 5.4, 14.1
- LABEL Parameter 4.4, 5.28
- Language Environment for MVS 1.5, 5.20
- LENGTH 2.79
- LENGTH Parameter (RECORD) 2.125–2.127
- LENGTH Subparameter 2.108
- LINES Parameter (OUTFIL) 2.82–2.85
- LINK Macro 6.2, 7.9, 7.29
- LIST Option 5.4, 5.19, 6.7, 6.10
- LOAD Macro 6.2
- LOCALE
 - Processing with INCLUDE/OMIT 2.16
- LOCALE Option 5.5, 7.57
- LRECL 2.72, 2.80, 2.82, 2.126–2.127
- LRECL Parameter 4.4, 5.5, 5.25, 5.28–5.29, 5.33
- LS Format 2.19, 2.29, 2.102, 2.105

M

- Macro Instructions (Assembler) 6.2–6.3
 - ATTACH Macro 6.2
 - LINK Macro 6.2
 - LOAD Macro 6.2
 - XCTL Macro 6.2
- Masks 2.79
- MAX 2.78
- Maximums, Example 3.47
- MAXSORT 2.59, 4.1, 4.7, 4.11, 9.1–9.21, 12.1
 - DD Statements for 4.16, 9.4–9.10
 - EXEC Statement for 4.3, 9.4
 - Exit Programs 9.13
 - File Size 4.7
 - Invoking from a Program 9.13
 - JCL/Control Stream Examples 9.17–9.21
 - Operator Interface 9.14–9.16
 - PARM Options 4.3, 5.1, 9.10–9.13
 - BKPTDSN 5.1, 9.10
 - DYNATAPE 5.1, 9.10, 9.14–9.16
 - MAXSORT 9.10
 - MAXWKSP 5.1, 9.11
 - MINWKSP 5.1, 9.11
 - NODYNATAPE 5.1, 9.10
 - RESTART 5.1, 9.12
 - SORTSIZE 5.1, 9.12
 - SORTTIME 5.1, 9.12
 - TAPENAME 5.1, 9.13
 - Performance Considerations 9.21, 13.1
 - PGM Names 4.3

- Starting 9.13–9.14
- Starting through JCL 9.4, 9.13
- Summary of Control Statements for 2.5
- Summary of the Chapter 1.8
- Tuning 9.21
- MAXSORT Option (MAXSORT) 9.10
- MAXWKSP Option (MAXSORT) 5.1, 9.11
- Merge
 - Creating Input Data Sets 4.7
 - Creating Input Records for Invoked Merge 7.5
 - Flow of 8.1–8.6
- MERGE Control Statement 2.35, 2.37–2.53, 5.14, 5.17, 5.30, 6.2, 6.8, 7.5, 7.56–7.57
- Merging 1.2
 - Phases of 1.2
- Message Data Set 5.23–5.25
- Messages 4.4, 5.7, 5.17, 5.22–5.23, 16.1
 - Authorization Messages 16.66
 - KEY Messages 16.66
 - PROC SYNCSORT Messages 16.65–16.66
 - See FLAG Option
 - Statistical Record Facility Messages 16.65
 - Summary of the Chapter 1.8
- MIN 2.78
- Minimums, Example 3.47
- MINWKSP Option (MAXSORT) 5.1, 9.11
- Missing Field Bytes, Record 2.66
- Mm Subparameter 2.109
- MODS Control Statement 2.54–2.57, 5.14, 6.3, 6.5, 6.9, 7.3, 7.5, 7.9–7.10, 7.29–7.31, 12.6
- MSG Option 5.4, 5.22–5.23, 6.10
- MSGDD Option 5.4, 5.23, 7.10, 7.31
- Multiple Lines, Record 2.66
- Multiple Output 2.62–2.63, 3.53–3.55
- Multiple Output Files 2.61

N

- National Language 1.5, 2.16, 2.37, 2.129, 5.20
 - with INCLUDE/OMIT 2.16
- NOCOMMAREA Option 5.3, 5.11
- NODETAIL Parameter (OUTFIL) 2.85
- NODYNATAPE Option (MAXSORT) 5.1, 9.10
- NOEQUALS Option 5.4, 5.15
- NOIOERR Option 5.4, 5.18
- NOLIST Option 5.4, 5.19
- NORC16 Option 5.5
- NORESET Option 5.5, 5.27

- NORLSOUT Option 5.5, 5.27
- Notational Conventions 2.11
- NULLOFL Parameter (OUTFIL) 2.86, 2.96
- NULLOUT Option 5.23
- NZDPRINT 5.6
- NZDPRINT Option 5.34

O

- OL Format 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
- OMIT Control Statement 2.58, 3.6
- Operator 2.93
- Operator Interface 9.14–9.16
- OPTCD Parameter 4.4
- OR Operator 2.21, 3.3
- OS/VIS COBOL
 - See COBOL E15
 - See COBOL E35
 - See COBOL Exits
- OSCL Option 5.2, 5.34, 6.7, 6.10, 12.3–12.5, 13.8
- OT 2.133
- OT Format 2.19, 2.29, 2.40, 2.102, 2.105, 2.131
- OUTFIL Control Statement 2.35, 2.59–2.87, 3.27–3.55, 6.2, 6.9, 13.2
- OUTFIL Group 2.62–2.63
- Output Data Sets 4.8–4.9
 - See SORTOUT Files
 - See SORTOUT Statement
- Output Files, Multiple 2.61, 3.53–3.55, 4.32–4.33
- Output Lines, Multiple 2.66, 2.70, 3.28
- Output Records
 - Converting Data 3.20–3.27
 - Editing Data 3.23–3.27
 - Formatting 2.65–2.66, 2.88–2.124, 3.14–3.27
- Output Records, Distributing 2.68
- Output Reports 2.59–2.87, 3.30–3.55
 - Averaging Data 3.47
 - Ending Record Number 2.64
 - Headers 2.69, 2.73, 2.82–2.85, 3.32–3.41
 - Obtaining maximums 3.47
 - Obtaining minimums 3.47
 - Pages, Logical 2.82–2.85
 - Records Included 2.64
 - Saving Records 2.65
 - Sections 2.80–2.82, 3.30–3.32
 - Starting Record Number 2.64
 - Subtotaling Data 3.41–3.46
 - Totaling Data 3.41–3.46

Trailers 2.73–2.80, 2.82–2.85, 3.32–3.53
Output Space
 See Secondary Allocation
OUTREC Control Statement 2.35, 2.88–2.124,
 3.14, 6.9, 13.2–13.3
OUTREC Parameter (OUTFIL) 2.65–2.66
OVFLO option 5.24

P

PAD Option 5.24
PAGE (&PAGE) 2.72, 2.77
Pages, Defining Logical 2.82–2.85
Parameter List
 24-bit 6.4–6.12, 7.3
 Optional Parameters 6.8
 31-bit 6.12–6.18, 7.3
Parameters
 Summary of the Chapter 1.7
Parameters (Control Statements)
 CKPT/CHKPT 2.52, 2.145
 CODE 2.13
 COND 2.17–2.34
 CONVERT 2.67, 2.116–3.27
 DYNALLOC 2.145–2.146
 ENDREC 2.64
 EQUALS/NOEQUALS 2.52, 2.146
 Exit-Name 2.54–2.56
 FIELDS 2.35, 2.37–2.52, 2.90–2.116,
 2.129–2.144, 2.149–2.150
 FILES 2.52, 2.62, 3.53–3.55
 FILSZ 2.147
 FNAMES 2.63
 FTOV 2.66
 HEADER 13.3
 HEADER1/HEADER2 2.68–2.73, 3.32
 INCLUDE/OMIT 2.63–2.64
 LENGTH 2.125–2.127
 LINES 2.82–2.85
 NODETAIL 2.85
 NULLOFL 2.86, 2.96
 OUTREC 2.65–2.66, 13.3
 REMOVECC 2.85
 SAVE 2.65
 SECTIONS 2.80–2.82, 3.30, 13.3
 SIZE 2.127, 2.147
 SKIPREC 2.53, 2.147, 13.3
 SPLIT 2.68
 STARTREC 2.64

STOPAFT 2.53, 2.147, 13.3
TRAILER 13.3
TRAILER1/TRAILER2 2.73–2.80, 3.32, 3.49
TYPE 2.125
VLFILL 2.66
VLTRIM 2.67
XSUM 2.150

Parameters (DD Statements)

AMP 4.4
BLKSIZE 4.4, 5.5, 5.25, 5.28–5.29
BUFND 4.4
BUFNI 4.4
BUFOFF 4.4
BUFSP 4.4
DCB 4.4, 5.5, 5.25, 5.28
DISP 4.4, 5.28
DSNAME/DSN 4.4
LABEL 4.4, 5.28
LRECL 4.4, 5.5, 5.25, 5.28–5.29, 5.33
OPTCD 4.4
RECFM 4.4, 5.5, 5.25, 5.28–5.29
SPACE 4.4, 5.5, 5.26
UNIT 4.4
VOLUME/VOL 4.4
PARASORT 4.1, 4.7, 10.1–10.9
 DD Statements for 4.16, 10.2–10.3
 EXEC Statement for 10.2, 11.2
 JCL for 10.2
 JCL/Control Stream Examples 10.9
 PARM Options 4.3
 PARMS Options 5.2
 PGM Names 4.3
 SORTIN Statement for 10.3
 Summary of the Chapter 1.8
PARASORT Sort
 DD Statements for 4.16
PARM Options 5.1–5.34
 BALANCE 5.3, 5.7, 13.4
 BALN 5.2, 5.34, 6.7, 6.10, 12.3–12.5, 13.8
 BKPTDSN 5.1
 BMSG 5.3, 5.7
 CENTWIN 2.41–2.51, 2.134–2.144, 5.3,
 5.7–5.9
 CMP 13.3
 CMP=CLC 2.41
 CMP=CPD 2.41
 COBEXIT 7.10, 7.31
 COMMAREA 5.11, 7.4

CORE 5.12, 13.4–13.6
 CPU 5.3, 5.7, 13.4
 CRCX 5.34, 6.7, 6.10
 DEBUG 5.13
 DIAG 5.13, 6.7, 6.10
 DYNALLOC 4.1, 5.13–5.14
 DYNATAPE 4.1, 5.1
 E15 5.14, 7.10
 E35 5.15
 ELAP 5.3–5.4, 5.7, 5.15, 13.4
 EQUALS 5.4, 5.15, 13.3
 EXTCOUNT 5.4, 5.16
 FILSZ 5.4, 5.16–5.17
 FLAG 5.4, 5.17, 6.10
 for Disk Sort 5.1–5.34
 for MAXSORT 5.1, 9.10–9.13
 for PARASORT 5.2
 for Tape Sort 5.2
 Format of 5.1
 HBSI 4.5, 5.4, 5.17
 HBSO 4.9, 5.4, 5.18
 INCOR 5.4
 INCORE 5.4, 5.18, 13.6
 IO 5.3–5.4, 5.7, 5.18, 13.4
 IOERR 5.4, 5.18, 6.10
 L6 5.4, 5.19, 14.1
 L7 5.4, 14.1
 LIST 5.4, 5.19, 6.7, 6.10
 LOCALE 5.5, 7.57
 MAXWKSP 5.1
 MINWKSP 5.1
 MSG 5.4, 5.22–5.23, 6.10
 MSGDD 5.4, 5.23, 7.10, 7.31
 NOCOMMAREA 5.11
 NODYNATAPE 5.1
 NOEQUALS 5.4, 5.15
 NOIOERR 5.4, 5.18
 NOLIST 5.4, 5.19
 NORC16 5.5
 NORESET 5.5, 5.27
 NORLSOUT 5.5, 5.27
 NULLOUT 5.23
 NZDPRINT 5.6, 5.34
 OSCL 5.2, 5.34, 6.7, 6.10, 12.3–12.5, 13.8
 OVFLO 5.24
 PAD 5.24
 PEER 5.34, 6.7, 6.10
 POLY 5.2, 5.34, 6.7, 6.10, 12.3–12.5, 13.8
 Precedence Rules 5.2
 PRINT121 5.5
 RC16 6.10
 RELEASE 5.5, 13.6
 RESERVE 5.5, 5.26
 RESERVEX 5.5, 5.27
 RESET 5.5, 5.27
 RESTART 5.1
 RLSOUT 5.5, 5.27
 SDB 5.5, 5.28–5.29
 SECOND 5.5, 13.6
 See \$ORTPARM Statement
 SIZE 5.12
 SKIPREC 5.5, 5.30, 7.52, 13.3
 SORTSIZE 5.1
 SORTTIME 5.1
 Specification in JCL-initiated Applications
 5.1
 Specification in Program-initiated
 Applications 5.1
 STOPAFT 5.5, 5.30, 13.3
 Summarized 5.2–5.3
 TAPENAME 5.1
 TRUNC 5.31
 UNINTDS 5.6, 5.31
 VLTEST 5.5, 5.32, 13.3
 VLTESTI 5.6, 5.33
 VSAMEMT 5.6, 5.34
 ZDPRINT 5.6, 5.34
 PD Format 2.19, 2.29, 2.38, 2.41, 2.102, 2.105,
 2.131, 2.134
 PD0 Format 2.19, 2.29, 2.39, 2.44, 2.102, 2.105,
 2.132, 2.138
 PDID Format 2.101
 PDIP Format 2.101
 PEER Option 5.34, 6.7, 6.10
 Performance Considerations 13.1–13.15
 Control of System Resource Usage 13.3
 Control Statements 13.2–13.3
 JCL- vs. Program-Invoked Sort 13.2
 Memory Management 13.3–13.6
 PARMS Options 13.3
 Summary of the Chapter 1.8
 Phases of Copying, Merging, Sorting 8.1–8.6
 PipeSort 1.7, 2.61, 3.53, 15.3
 PL/1 Programs 1.1, 6.1
 POLY Option 5.2, 5.34, 6.7, 6.10, 12.3–12.5, 13.8
 Precedence Rules 5.2

PRINT121 Option 5.5
PROC SYNCSORT - An Accelerator for SAS™
 Sorting 1.6, 15.3
PROC SYNCSORT Messages 16.65–16.66
Program Exits
 See Exit Programs

R

RC16 Option 6.10
RDW 2.9, 2.116, 2.126
RECFM Parameter 4.4, 5.5, 5.25, 5.28–5.29
RECORD Control Statement 2.125–2.128, 6.3,
 12.6
Record Counts 3.49–3.53
Record Format 2.125–2.128
Record Length 2.125–2.128
 Maximum for Fixed-Length Records 4.5
 Maximum for Variable-Length Records 4.5
Record Selection 2.16–2.34, 2.53, 2.63–2.64, 2.147,
 3.2–3.11, 5.30
 Using Bit Level Logic 2.31–2.33
Records, Distributing Output 2.68
Reference Guide 1.9
Reformatting Records 2.35–2.36, 2.65–2.66,
 2.88–2.124, 3.14–3.18
 CHANGE Subparameter 2.113
 Column Alignment 2.69
 Data Conversion and Editing 2.97–2.121,
 3.20–3.27
 Inserting Binary Zeros 2.103, 3.18
 Inserting Blanks or Spaces 2.103
 Inserting Hexadecimal Characters 2.103
 Inserting Literal Characters 2.103
 Missing Field Bytes 2.66
 OUTREC Parameter 2.65–2.66
 Record on Multiple Lines 2.66
 Replace 2.113
 Search and Replace 2.113
 Variable-Length Records 2.90
Register Conventions 7.4
Registers
 See Register Conventions
RELEASE Option 5.5, 13.6
REMOVECC Parameter (OUTFIL) 2.85
Replace 2.113
Report Writing 2.61, 2.68–2.87, 3.30–3.55
Repositioning Record Fields 3.14–3.18

RESERVE Option 5.5, 5.26
RESERVEX Option 5.5, 5.27
RESET Option 5.5, 5.27
RESTART Option (MAXSORT) 5.1, 9.12
RETRY 2.146, 5.14
Return Codes 6.10, 6.15
REXX Exits 2.55, 7.58–7.60
REXX Programs 1.1
RLSOUT Option 5.5, 5.27
Run-Time Constants, Generating 2.104

S

SAVE Parameter (OUTFIL) 2.65
SC 2.146, 5.14
SDB 4.18, 4.20, 4.22, 4.25, 4.30, 5.28
SDB Option 5.5, 5.28–5.29
Search and Replace 2.113
SECOND Option 5.5, 13.6
Secondary Allocation 4.9–4.10
SECTIONS Parameter (OUTFIL) 2.80–2.82, 3.30
SEQNUM 2.35, 2.103
SIGNS 2.79
SIZE Option 5.3, 5.12
SIZE Parameter (SORT) 2.127, 2.147
SKIPREC Option 5.5, 5.30, 7.52, 13.3
SKIPREC Parameter (MERGE) 2.53
SKIPREC Parameter (SORT) 2.147
SMF Formats 2.102
SMF Formats, Converting 2.102
SNAP Dump 5.13
Sort
 Creating Input Data Sets for 4.5
 Flow of 8.1–8.6
Sort Control
 Summary of the Chapter 1.8
SORT Control Statement 2.35, 2.129–2.148, 5.14,
 5.17, 5.30, 6.2, 7.52, 7.56–7.57, 12.6
SORT/MERGE 6.2–6.4
SORT/MERGE Options
 Precedence Rules 5.2
SORTBKPT Statement 9.6–9.7
SORTCKPT Statement 4.1, 4.14
SORTIN End-of-File 7.9, 7.14
SORTIN File 4.1, 4.5, 5.4, 5.17, 6.8, 6.12, 6.17,
 7.8–7.9, 7.11–7.14, 7.51
SORTIN Processing 7.52–7.53
SORTIN Statement 4.1, 4.5–4.7, 5.28, 6.2, 7.9,

- 10.3
- for PARASORT 10.3
- Sorting
 - Phases of 1.2
 - Types of 1.2
- Sorting Technique
 - Disk Sort 4.1, 12.1, 13.1
 - MAXSORT 4.1, 4.7, 4.11, 4.16, 9.1–9.21, 12.1, 13.1
 - PARASORT 4.1, 4.16, 10.1–10.9
 - Performance Considerations 13.1, 13.15
 - Tape Sort 4.1, 4.10, 4.16, 12.1, 12.11, 13.1
 - Balanced (BALN) 12.4–12.5
 - Oscillating (OSCL) 12.4–12.5
 - Polyphase (POLY) 12.4–12.5
- SORTINn Statement 4.1, 4.7
 - SORTINnn Statement 4.7
- SORTINnn File 5.15, 7.6
- SORTINnn Statement 4.1, 4.7, 6.2, 7.5
- SORTLIB Statement 6.2, 12.3
- SORTMODS Library 7.3
- SORTMODS Statement 4.1, 4.15
- SORTOFx File 5.27–5.29
- SORTOFx Statement 4.1, 4.8, 6.2
- SORTOFxx File 5.6, 5.27–5.28, 7.30, 7.36
- SORTOFxx Statement 4.1, 4.8, 6.2
- SORTOU00 Statement 9.7–9.9
- SORTOUT File 4.1, 4.8, 5.4–5.6, 5.18, 5.27–5.29, 7.30, 7.36, 7.52
- SORTOUT Files 6.2
- SORTOUT Space 5.5, 5.27
- SORTOUT Statement 4.1, 4.8, 6.2, 7.29
- SORTOUT VSAM File 5.27
- SORTOUT Write Errors 7.55
- SORTPARn Statement 10.5–10.7
- SORTSIZE Option (MAXSORT) 5.1, 9.12
- SORTTIME Option (MAXSORT) 5.1, 9.12
- SORTWK
 - Dynamic Allocation of 5.13
- SORTWK Requirements 12.4
- SORTWKnn File 7.7
- SORTWKnn Statement 4.1, 4.9–4.11, 6.2, 12.4–12.5
 - Conditions of Use 4.10
 - See DYNALLOC Option
- SortWriter 2.59, 2.61, 2.68
 - Sample Report 1.4
- SORTXSUM 4.8, 5.27
- SORTXSUM Statement 6.2

- SPACE Parameter 4.4, 5.5, 5.26
- Special Esoteric Names 10.7
- SPLIT Parameter (OUTFIL) 2.68
- Starting SyncSort 1.1
 - Summary of the Chapter 1.7
- STARTREC Parameter (OUTFIL) 2.64
- Statistical Record Facility Messages 16.65
- STEPBLIB Statement 4.1
- STEPLIB Statement 4.4
- STOPAFT Option 5.5, 5.30, 13.3
- STOPAFT Parameter (MERGE) 2.53
- STOPAFT Parameter (SORT) 2.147
- Storage
 - Disk 4.9
 - Intermediate 4.9–4.10, 12.4
- SUBAVG 2.78
- SUBCOUNT 2.79
- SUBCOUNT15 2.80
- SUBMAX 2.78
- SUBMIN 2.78
- Substring Comparison 2.30
- SUBTOTAL 2.77
- SUM Control Statement 2.35, 2.149–2.152, 3.11, 5.15–5.16, 6.9, 7.27, 13.2
- SyncSort
 - Data Utility 1.3, 3.1–3.55
 - Description of 1.1
 - Features of 1.1–1.6
 - Initiation of 1.1
 - SortWriter 1.4
- SyncSort Messages 4.4
- SyncSort/COBOL Advantage 1.6, 15.2
- SYSIN Statement 4.1, 4.11
- SYSLIN Statement 4.1, 4.15
- SYSLMOD Statement 4.1, 4.15
- SYSOUT
 - See Message Data Set
- SYSOUT Statement 4.1, 4.4, 6.2
- SYSPRINT Statement 4.1, 4.15
- System Abend (0C7) 5.9, 5.32
- SYSUT1 Statement 4.1, 4.16

T

- Tape Sort 4.1, 4.14, 12.1–12.11, 13.8–13.9
 - Control Statements for 12.6
 - Converting to MAXSORT 13.2
 - DD Statements for 4.16, 12.3–12.5

Device Types 4.10
 Devices 12.4
 EXEC Statement 12.2–12.3
 EXEC Statement for 4.3
 Invoking from a Program 12.9–12.11
 JCL/Control Stream Examples 12.7–12.8
 PARM Options 4.3, 12.2
 BALN 5.2, 12.3–12.5, 13.8
 OSCL 5.2, 12.3–12.5, 13.8
 POLY 5.2, 12.3–12.5, 13.8
 PARMS Options 5.2
 Performance Considerations 13.1
 PGM Names 4.3
 Starting 4.10, 12.7–12.11
 Starting through JCL 12.7–12.8
 Summary of Control Statements for 2.5
 Summary of Restrictions 12.1–12.2
 TAPENAME Option (MAXSORT) 5.1, 9.13
 The INCORE PARM is set to OFF. 4.10
 TIME (&TIME) 2.71, 2.76, 2.105
 TM1 2.102
 TM2 2.102
 TM3 2.102
 TM4 2.102
 TOTAL 2.77
 TRAILER1/TRAILER2 Parameter (OUTFIL)
 2.73–2.80, 3.32, 3.49
 TRAN subparameter 2.96
 TRUNC Option 5.31
 TS Format 2.19, 2.29, 2.38, 2.102, 2.105, 2.131
 Turnaround Sort
 See Incore Sort
 TYPE Parameter (RECORD) 2.125

U

UNINTDS Option 4.7, 5.3, 5.6, 5.31
 UNIT Parameter 4.4

V

Value-Added Products 1.6–1.7, 15.1–15.3
 PipeSort 1.7, 15.3
 PROC SYNCSORT - An Accelerator for SAS™
 Sorting 1.6, 15.3
 Summary of the Chapter 1.8
 SyncSort/COBOL Advantage 1.6, 15.2
 Variable-Length Records 2.9, 2.41, 2.90, 2.116,
 2.125, 3.9–3.11, 7.12–7.13, 7.17–7.19,
 7.33–7.34, 7.40–7.42
 Maximum Length 4.5
 See VLTEST Option
 Validity Testing 5.32
 Visual SyncSort 1.6, 15.1
 VLFILL Parameter (OUTFIL) 2.66
 VLTEST Option 5.3, 5.5, 5.32, 13.3
 VLTESTI Option 5.3, 5.6, 5.33
 VLTRIM Parameter (OUTFIL) 2.67
 VOL Parameter
 See VOLUME Parameter
 VOLUME Parameter 4.4
 VS COBOL II
 See COBOL E15
 See COBOL E35
 See COBOL Exits
 VSAM 4.5, 4.8, 5.27, 7.54–7.55
 VSAM SORTOUT 5.5
 VSAMEMT Option 5.6, 5.34

W

Work Areas
 See SORTWKnn Statement
 Work Space 5.6, 5.19, 5.26–5.28

X

XCTL Macro 6.2, 7.9, 7.29
 XSUM Parameter (SUM) 2.150

Y

Y'DATEx' 2.28–2.30
 Y2B Format 2.19, 2.29, 2.39, 2.42, 2.101–2.102,
 2.105, 2.132, 2.134
 Y2C Format 2.19, 2.29, 2.39, 2.42, 2.101–2.102,
 2.105, 2.132, 2.135
 Y2D Format 2.19, 2.29, 2.40, 2.43, 2.101–2.102,
 2.105, 2.132, 2.135
 Y2P Format 2.19, 2.29, 2.40, 2.43, 2.101–2.102,
 2.105, 2.132, 2.136
 Y2S Format 2.19, 2.29, 2.40, 2.44, 2.101–2.102,
 2.105, 2.132, 2.136
 Y2T Format 2.19, 2.29, 2.40, 2.45, 2.101–2.102,
 2.105, 2.132, 2.134, 2.139
 Y2U Format 2.19, 2.29, 2.40, 2.45, 2.101–2.102,
 2.105, 2.132, 2.134, 2.139

Y2V Format 2.19, 2.29, 2.40, 2.45, 2.101–2.102,
2.105, 2.132, 2.134, 2.139
Y2W Format 2.19, 2.29, 2.40, 2.45, 2.101–2.102,
2.105, 2.132, 2.134, 2.139
Y2X Format 2.19, 2.29, 2.40, 2.45, 2.101–2.102,
2.105, 2.132, 2.134, 2.139
Y2Y Format 2.19, 2.29, 2.40, 2.45, 2.101–2.102,
2.105, 2.132, 2.134, 2.139
Y2Z Format 2.19, 2.29, 2.40, 2.42, 2.101–2.102,
2.105, 2.133, 2.135
Year Data, Converting 2.100–2.102

Z

ZD Format 2.19, 2.29, 2.40–2.41, 2.102, 2.105,
2.131, 2.133–2.134
ZDPRINT 5.6
ZDPRINT Option 5.34
ZSPACE 1.6, 16.53

We welcome comments on the usefulness and readability of this Manual. Your comments, along with suggested additions or deletions, will help us to improve future editions of the publications.

If you would like a reply, please indicate your name, business title, and business address. All comments and suggestions become the property of Syncsort Incorporated.

READERS' COMMENTS

SyncSort for z/OS
Programmer's Guide

SI-4301-4

Fold the form on the two lines, staple, and mail.

No postage stamp is necessary if form is mailed in the United States.



Fold

Fold



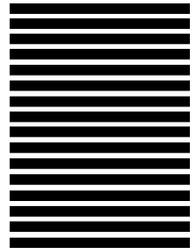
No postage
necessary
if mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 80 WOODCLIFF LAKE, NJ

POSTAGE WILL BE PAID BY ADDRESSEE

**Syncsort Incorporated
SyncSort for z/OS Product Services
50 Tice Boulevard
Woodcliff Lake, NJ 07677**



Fold

Fold