P R O V E N

performance

SyncSort for z/VSE

Programmer's Guide

Release 3.7

SI-0328-G
070809

Table of Contents

SyncSort for z/VSE Release 3.7 Summary of Changes

Performance Improvements

SyncSort for z/VSE performance has been improved by the following.

- Improved elapsed time for sort applications that use the Virtual Sortwork feature of the SyncSort Dynamic Storage Manager (DSM) due to improved I/O techniques.
- Improved elapsed time for sort applications that do not use the SyncSort DSM due to more efficient buffer utilization for Sortwork I/O.
- Elapsed time improvements for applications using VSAM files as input to SyncSort.

Data Utility Features

The SyncSort for z/VSE data utility features have been enhanced by the following:

DUPKEYS Control Statement

- The new ALLDUPS parameter specifies that only records with duplicate SORT/MERGE fields are retained.
- The new FIRSTDUP parameter specifies that only the first record of those with duplicate SORT/MERGE fields is retained.
- The new LASTDUP parameter specifies that only the last record of those with duplicate SORT/MERGE fields is retained.
- The NODUPS parameter specifies that only records with unique SORT/MERGE fields are retained.

INPFIL/OUTFIL Control Statements

- The new BUILD parameter is identical to the OUTREC parameter of the OUTFIL control statement.

- The new IFTHEN parameter uses conditional logic which enables you to reformat records based on specific criteria.
- The new IFOUTLEN parameter overrides the maximum record length, which is automatically set by the IFTHEN parameter.
- The new OVERLAY parameter enables you to change particular columns within a record, and add fields to the end of a record, without rebuilding the entire record.
- The SECTIONS parameter now supports non-contiguous data fields to be used as the break control field.

INREC/OUTREC Control Statements

- The new IFTHEN, IFOUTLEN, and OVERLAY parameters can be used to conditionally reformat records or reformat only selected portions of records.
- The new RESTART subparameter of the SEQNUM parameter can be used to restart the sequence numbering.

JOINKEYS Control Statement

- Data formats of CH, FI, PD, and ZD are now supported in addition to BI.

Other Features

- Support for UFF and SFF data formats in the INCLUDE/OMIT, MERGE, OUTREC, and SORT parameters has been added.
- Support for additional data formats in SSRAM applications has been added.

Operating System

SyncSort for z/VSE Release 3.7 supports z/VSE 4.2.

Messages

- Now message SHS118A indicates that the input file to SYNCHSTO is empty.
- Message WER127A now applies to the BUILD and OVERLAY parameters in addition to the INREC, OUTREC, and REFORMAT parameters.
- Message WER128A now applies to the BUILD, OVERLAY, and REFORMAT parameters in addition to the INREC and OUTREC parameters.

- Message WER129A now applies to the IFTHEN parameter in addition to the INCLUDE and OMIT parameters.
- Message WER130A now applies to the IFTHEN parameter in addition to the INCLUDE and OMIT parameters.
- Message WER132A now applies to the IFTHEN parameter in addition to the INCLUDE and OMIT parameters.
- Message WER133A now applies to the BUILD parameter in addition to the INREC, OUTREC, and REFORMAT parameters.
- New message WER157A indicates that the IFOUTLEN parameter has a specified length greater than the RECORD control statement's maximum record length.
- Message WER191A now indicates that the number of JOINKEYS fields specified on two JOINKEYS control statements are not equal.
- New message WER205A indicates that one or more of the corresponding fields on two JOINKEYS control statements are in incompatible formats.

Chapter 1. Introduction

An Introduction to SyncSort for z/VSE

SyncSort for z/VSE is a high performance sort/merge/copy program for IBM System/370, System/390, and z/Architecture machines running under VSE/ESA, z/VSE, and MVT/VSE (Software Pursuits).

SyncSort for z/VSE is designed to conserve system resources and provide significant performance benefits.

SyncSort for z/VSE can be initiated by job control language or invoked from a program written in COBOL, PL/1, or Assembler language. Eleven types of user exit routines can be specified for additional programming flexibility.

SyncSort's Basic Functions

SyncSort for z/VSE has three basic functions:

1. Sorting - rearranging data set records to produce a specific sequence

2. Merging - combining up to 32 presequenced data sets into one data set that has the same sequence

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

Sorting

A sort is the ordering of a data set of unspecified sequence into specific sequential form. For example, a sort can arrange records in numeric or alphabetic order.

Up to 32 files can be sorted at one time, and a number of different devices can be used for input, output, and work files.

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

- The control statements and job control information are read and analyzed, and the operational parameters for the sort are established.
- The input data is read into virtual 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 is used to combine files where data in each file is in sequential order. The merge combines up to 32 files into one output file.

A merge has only two phases that perform the following functions:

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

Copying

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

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

SyncSort's Data Utility and SortWriter Features

SyncSort for z/VSE 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.

Processing Multiple Input Files

The multiple input facility (INPFIL) allows multiple input files to be processed by just one pass of the sort. Each of these files can have unique specifications that determine which records are to be included and how the records are to be formatted. Moreover, all these files can originate from one input device, or each file can originate from a different device.

Generating Multiple Output Files

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 that can include data fields. Totals, subtotals, record counts, and subcounts can be produced at report, page, and section levels. 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 Summing Fields

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

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 and binary zeros, realign fields, convert numeric data to its printable format, convert fixed-length records to variable-length records or vice-versa, and convert data to its printable hexadecimal format. The ability to delete irrelevant data fields before sorting via INREC can provide important performance benefits.

The AVG, MAX, MIN, and SUM features allow deletion of records with equal sort control fields and optional replacement of specified numeric fields in the retained record with the average, maximum, minimum, or sum of the field for all records with the same control field. The deleted records can be written to a separate data set.

Join Facility

The join facility of SyncSort for z/VSE provides the capability to join records from two source files. Each record from the first file with a given value in one or more fields (the join key) is joined to each record from the second file that has an identical value in its join key. The joined records are passed to the sort/copy process. The power of this facility is enhanced by the ability to eliminate records from either or both files and to control the disposition of paired and unpaired records resulting from the join operation.

Join Processing Sequence

Join processing replaces the reading of the input data set (SORTIN) during a more traditional SORT or COPY. The records created by joining and reformatting are inserted into the SORT/COPY process immediately prior to record selection by the INCLUDE/OMIT control statement. Note that the specifications in all of the other control statements after join processing refer to the positions of the fields in the reformatted join records. See Figure 1 below.

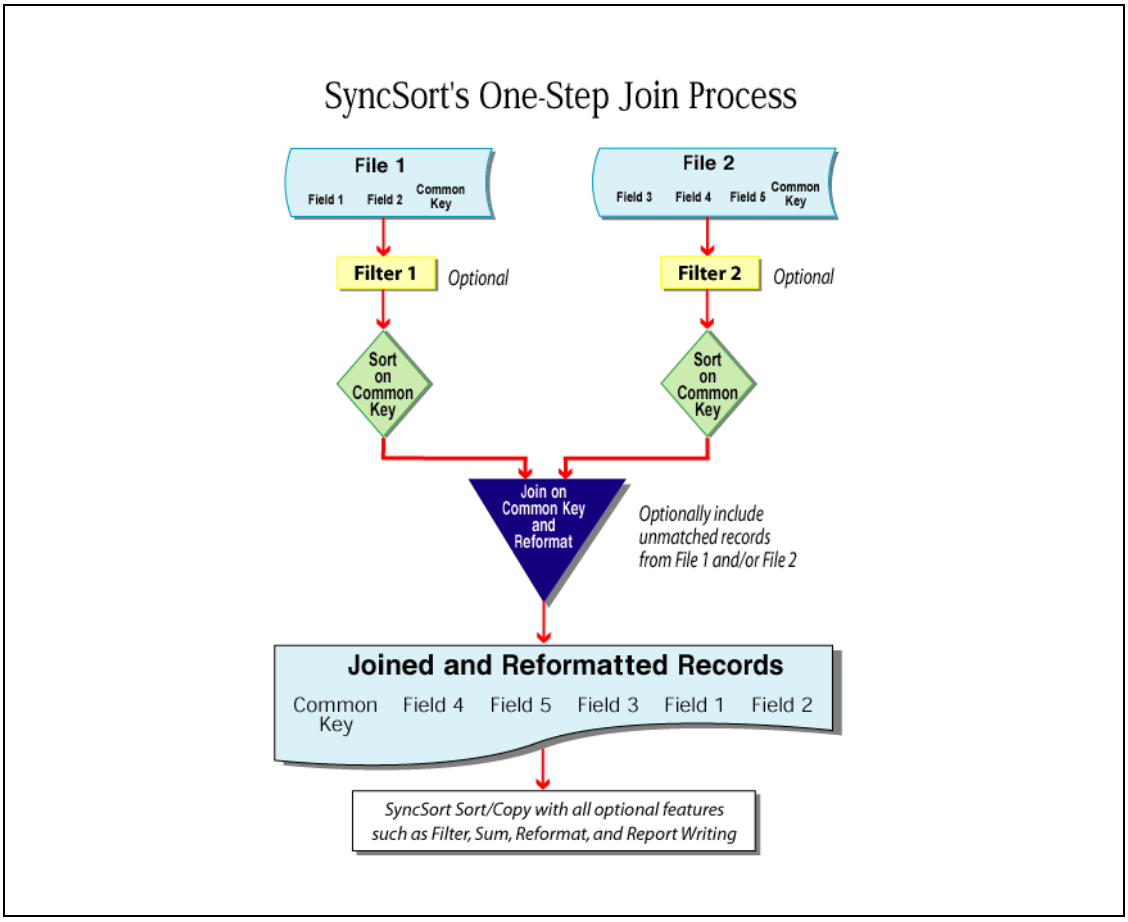


Figure 1. One-Step Join Processing Sequence

Sample SortWriter Report

The report in Figure 2 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, invoice date, and company name. 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.

```
                                                                                PAGE:     3
                                ACCOUNTS RECEIVABLE AGING REPORT FOR 01/30/92    DATE:04/22/92
                                ******************************************
                                             INVOICE STATUS:O
                                             ****************

------------            ------- ----   -----    --------  ----------------------  ----------------------
                                                                INVOICE                  BALANCE
COMPANY NAME            ADDRESS CO #    INV #    INV DATE   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
HOLOFERNES 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 2. Sample SortWriter Report

SyncSort's Operational Features

SyncSort for z/VSE has many operational features that contribute to its flexibility and functionality.

- SyncSort for z/VSE interfaces with the VSE/VSAM Space Management for SAM feature to permit VSAM space management of input, output, and sort work files. This capability simplifies JCL requirements and facilitates automatic secondary sort work allocation.

- SyncSort for z/VSE can be installed in the SVA. However, on VSE/ESA and z/VSE systems, most of the SVA can be saved by activating SyncSort for z/VSE's Virtual Library and Virtual Sortwork features at installation time.

- Default values can be dynamically modified at execution time via the PARMEXIT feature, allowing main storage allocations to reflect observed or anticipated VS paging activity.

- SyncSort for z/VSE interfaces with many disk space management packages. This capability provides many benefits, including automatic release of unused output disk space and automatic secondary sort work allocation.

Other operational features include support for input files on different devices (e.g., disk, tape, and card readers), support for mixed device types for disk sort work space, automatic incore sorting, support for ASCII tape input and output files, and support for spanned variable-length records.

SyncSort's SYNCBIX feature is a high performance replacement for the BLDINDEX process performed by Access Methods Services (IDCAMS). The SYNCBIX feature interfaces with SyncSort for z/VSE to allow efficient reading of the base cluster, fast extracting of the record pointer information, primary and alternate keys, and high performance sorting of these records.

Structure of the Programmer's Guide

The *SyncSort for z/VSE Programmer's Guide* is a reference manual designed for use by applications programmers when sorting, merging, or copying sequential data sets with SyncSort. 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 a functioning, efficient sort.

This manual is organized into the following chapters:

SyncSort Control Statements describes the coding and use of the ALTSEQ, ANALYZE, DUPKEYS, END, INCLUDE/OMIT, INPFIL, INREC, JOIN, JOINKEYS, MERGE, MODS, OMIT, OPTION, OUTFIL, OUTREC, RECORD, REFORMAT, SORT, SUM, XDUPFIL, and XSUMFIL statements. The discussion of a particular control statement includes these

topics: the statement's coding format, the versatility provided by the various parameters (many of them unique to SyncSort), the interaction between this control statement and other statements, and simple examples.

Using the SyncSort Dictionary Feature describes how to use a dictionary_name to represent most fields or constants recognized in SyncSort control statements.

How to Use SyncSort's Data Utility Features explains and illustrates the Data Utility features and SortWriter using a series of sample applications. Each application is self-contained and provides both the required JCL and the control statements necessary to accomplish tasks such as record and field selection, report writing, and multiple output file generation.

Job Control Language and Sample JCL/Control Statement Streams analyzes SyncSort's job control requirements and describes the job control statements that may be needed. Two methods for setting up disk work file statements are described. Sample JCL/control statement streams illustrate typical SyncSort applications and can be used as models.

EXEC PARM Options describes the parameters that can be passed to SyncSort with the PARM option on the JCL //EXEC statement.

Invoking SyncSort from a Program explains how to invoke SyncSort for z/VSE from programs written in Assembler, COBOL, and PL/1 and provides examples of invoked sorts.

User Exit Programs describes the different types of exits that can be written to perform various tasks at different stages of the sort/merge. Each exit point is fully documented together with the tasks that are appropriate at that point. Exits for VSAM files are also discussed.

Creating VSAM Alternate Index Files with SyncSort describes SyncSort's SYNCBIX feature, a high performance replacement for the BLDINDEX process performed by Access Methods Services (IDCAMS).

SyncSort Reentrant Access Method Operation describes the interface between an invoking program and SyncSort, which allows a program to invoke up to 16 concurrent sorts.

The SYNCHSTO Utility Program describes a separate program that determines information about variable-length record files, which then can be used to run more efficient sorts of those files.

Messages documents all of the error and informational messages generated by the SyncSort program. Errors that occur most frequently are explained in detail along with suggestions for correcting them.

Related Reading

SyncSort for z/VSE Installation Guide

This manual explains how to install and maintain SyncSort.

Exploiting SyncSort for z/VSE: JOIN

This booklet demonstrates how to use SyncSort's JOIN feature through the use of sample applications.

Exploiting SyncSort: SortWriter Data Utilities Guide

This booklet explains how to use SyncSort's Data Utility features, with special emphasis on report writing. Comprehensive sample applications illustrate how the control statements produce formatted reports.

Chapter 2. SyncSort Control Statements

The control statements tell SyncSort for z/VSE how to process files. These statements are summarized below.

Control Statement	Function
ALTSEQ	Specifies an alternate collating sequence for control fields with an AQ format.
ANALYZE	Determines how much disk work space an application will require before it executes.
DUPKEYS	Deletes records with equal control fields and replaces specified fields with calculated average, maximum, minimum, or sum numeric values.
END	Signals the end of control statements.
INCLUDE	Specifies the criteria that determine whether or not records are included in an application.
INPFIL	Describes the input file(s) and specifies processing options.
INREC	Reformats the input record before sort/merge processing.
JOIN	Specifies the disposition of paired and unpaired records in a join.

JOINKEYS	Enables join feature processing and identifies the fields used to select records for join processing.
MERGE	Defines a merge, copy, or compare application and specifies merge control fields.
MODS	Specifies user exit(s).
OMIT	Specifies the criteria that determine whether or not records are omitted from an application.
OPTION	Specifies processing options and overrides installation defaults.
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.
REFORMAT	Defines the record layout to be produced by the join processing specified on the application's JOINKEYS control statement.
SORT	Defines a sort or copy application and specifies sort control fields.
SUM	Deletes records with equal control fields and sums numeric fields on those records.
XDUPFIL	Describes the XDUP output file and any optional SortWriter processing desired.
XSUMFIL	Describes the XSUM output file and any optional SortWriter processing desired.

Control Statement Summary Chart

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

OPERATION NAME	PARAMETERS	DELIVERED DEFAULT
ALTSEQ	CODE=(ccpp$_1$,…,ccpp$_{256}$)	Standard EBCDIC series
ANALYZE	CALC	
DUPKEYS	{ ALLDUPS function [,function]...[,FORMAT=f] FIELDS=NONE FIRSTDUP [,NODUPS] LASTDUP [,NODUPS] NODUPS } where function is: { AVG MAX MIN SUM }=(p$_1$,l$_1$[,f$_1$] [,p$_2$,l$_2$ [,f$_2$]]...)	No reduction of equal-keyed records
	XDUP	
END		
INCLUDE	COND={ ALL NONE (c$_1$ [{ ,AND, ,&, ,OR, ,|, } c$_2$…]) [,FORMAT = f] }	Sort/Merge all records

Table 1. (Page 1 of 14) Control Table Summary Chart

OPERATION NAME	PARAMETERS	DELIVERED DEFAULT
INPFIL	BLKSIZE=n	Unblocked records
	BUFLIM=n	BUFLIM=255
	BUFOFF=n	BUFOFF=0
	BYPASS	
	CLOSE={RWD | NORWD | UNLD | (UNLD,RALL) | (UNLD,REL) | (UNLD[,Rinput$_1$…,Rinput$_{32}$]) | (UNLD,RNONE)}	CLOSE=RWD Release at EOJ

Table 1. (Page 2 of 14) Control Table Summary Chart

OPERATION NAME	PARAMETERS	DELIVERED DEFAULT
INPFIL	CONVERT	
	CRDSIZE=n	CRDSIZE=80
	DATA={E / A}	DATA=E
	EXIT	
	FILES={1 / n / (1,2,…,32)}	
	FNAMES={filename / (filename$_1$[,filename$_2$][,filename$_3$]…)}	
	FTOV	
	{INCLUDE / OMIT}=(…)	
	INPUT	Accepted but ignored
	{INREC / BUILD / OVERLAY / IFTHEN}= (…)	Record unchanged
	LRECL={nnn / (l_{max}[,l_{min}])}	
	L4FILL ={NONE / X'nn' / C'c'}	L4FILL = NONE
	NOCHAIN	
	OPEN={RWD / NORWD}	OPEN = RWD
	PRESEQ	Accepted but ignored
	SKIPBYTE	Accepted but ignored
	{SKIPREC / STARTEC}=n	SKIPREC=0

Table 1. (Page 3 of 14) Control Table Summary Chart

OPERATION NAME	PARAMETERS	DELIVERED DEFAULT
INPFIL	SPAN	Unspanned records
	{STOPAFT / ENDREC}=n	
	SYSIPT	
	TOL	Only if VSAM input
	VLFILL=b	
	VLLONG={YES / NO}	VLLONG=NO
	VLTRIM=b	
	VOLUME={n / (n_1,...,n_{32})}	VOLUME=1
	VSAM	(Optional if managed-SAM)
	VTOF	
INREC	{FIELDS / BUILD / OVERLAY / IFTHEN}=(...)	Input records unchanged
JOIN	UNPAIRED	
	F1	
	F2	
	ONLY	

Table 1. (Page 4 of 14) Control Table Summary Chart

OPERATION NAME	PARAMETERS	DELIVERED DEFAULT
JOINKEYS	FILE={F1 / F2}	
	FIELDS=(p_1,l_1,f_1,o_1[,p_2,l_2,f_2,o_2,...[,p_{64},l_{64},$f_{64}o_{64}$]])	
	FORMAT = f	FORMAT = BI
	LRECL ={nnn / (l_1[,l_4])}	
	TYPE={F / V}	
	BLKSIZE=n	
	BUFLIM=n	BUFLIM = 255
	BUFOFF = n	BUFOFF=0
	BYPASS	
	CLOSE={RWD / NORWD / UNLD}	CLOSE=RWD
	CRDSIZE = n	CRDSIZE=80
	DATA={E / A}	DATA=E
	{ENDREC / STOPAFT}=n	
	{INCLUDE / OMIT}=(...)	
	OPEN={RWD / NORWD}	OPEN=RWD

Table 1. (Page 5 of 14) Control Table Summary Chart

OPERATION NAME	PARAMETERS	DELIVERED DEFAULT
JOINKEYS	SORTED	
	SPAN	
	{ STARTREC / SKIPREC } =n	
	SYSIPT	
	TOL	
	VOLUME=n	
	VSAM	
MERGE	{ FIELDS=(p_1,l_1[,f_1],o_1, …,p_{64},l_{64}[,f_{64}],o_{64})[,FORMAT = f] / FIELDS=COPY / FIELDS=COMPARE [(n)] }	
	BIAS=n	Accepted but ignored
	CENTWIN={ s / f }	s=80
	{ EQUALS / NOEQUALS }	EQUALS
	{ FILES / ORDER } =n	FILES or ORDER=1 (FILES or ORDER=2 for JOIN and COMPARE)
	FILESOUT=n	FILESOUT=1
	JOINWORK=n	JOINWORK=1
MODS	PHn=(name,loading information,Enn_1,...,Enn_n)	
OMIT	COND={ ALL / NONE / (c_1 [{ ,AND, / ,&, / ,OR, / ,|, } c_2…])[,FORMAT=f] }	Sort/Merge all records

Table 1. (Page 6 of 14) Control Table Summary Chart

OPERATION NAME	PARAMETERS	DELIVERED DEFAULT
OPTION	ADDROUT={A | D}	
	CALCAREA	
	CENTWIN={s | f}	s=80
	{CHALT | NOCHALT}	NOCHALT
	CMP={CLC | CPD}	CMP=CPD
	CMPINOM={CLC | CPD}	CMPINOM=CLC
	DATE={ADATE | EDATE | IDATE | STD}	DATE=STD
	DEVIN=nn	DEVIN=49
	DEVOUT=nn	DEVIN=49
	DEVWK={n | (n_1,...,n_3) | NO}	
	DIAG	
	{DUMP | NODUMP}	
	{EQUALS | NOEQUALS}	NOEQUALS-Sort EQUALS-Merge
	{ERASE | NOERASE}	NOERASE

Table 1. (Page 7 of 14) Control Table Summary Chart

OPERATION NAME	PARAMETERS	DELIVERED DEFAULT
OPTION	FILNM=(output$_1$,...,output$_{32}$,input$_1$,...,input$_{32}$,cccc)	SORTOUT/ SORTOFx/ SORTOxx/ SORTINx/ SORTIxx/ SORTWKx
	GVSRANY={n / nK / nM}	GVSRANY=32K
	GVSRLOW={n / nK / nM}	GVSRLOW=128K
	{IGNRL4 / NOIGNRL4 / NOVLSHRT / VLSHRT}	Respect l_4
	INCOR=OFF	Incore sort if possible
	JOINWK={joinwork / (joinwork$_1$,...,joinwork$_9$)}	Default symbolic unit numbers
	KEYLEN=n	KEYLEN=0
	LABEL=(output$_1$,...,output$_{32}$,input$_1$,...,input$_{32}$,work)	Standard labels
	LOCALE={CURRENT / name / NONE}	No LOCALE processing
	NOINC	Incore sort if possible
	NOTPMK	Tape marks written
	NRECOUT={RC0 / RC4 / RC16}	RC0, continue processing
	NRECS=n	Output all records
	OVFLO={RC0 / RC4 / RC16}	RC0, continue processing

Table 1. (Page 8 of 14) Control Table Summary Chart

OPERATION NAME	PARAMETERS	DELIVERED DEFAULT
OPTION	PRINT={ALL | CRITICAL | CRITPLUS | NONE}	PRINT=ALL (JCL) PRINT=CRITICAL (Pgm)
	ROUTE={LAL | LOG | LST | nnn}	ROUTE=LST (JCL) ROUTE=LOG (Pgm)
	RPS={ON | OFF}	RPS=OFF
	SAMESDS={RECFMOUT | UNDEFOUT}	Accepted but ignored
	SKIPREC=n	SKIPREC=0
	SORTIN={input | (input$_1$,…,input$_{32}$)}	Default symbolic unit numbers
	SORTOUT={output | (output$_1$,…,output$_{32}$)}	Default symbolic unit numbers
	SORTWK={work | (work$_1$,…,work$_9$)}	Default symbolic unit numbers
	SPANINC={RC0 | RC4 | RC16}	RC16, and terminate if encounter incomplete spanned records
	STOPAFT=n	Output all records
	STORAGE=([n | nK | nM][,AUTO | ,NOVIRT | ,REAL | ,VIRT])	(1) SIZE parameter (2) program partition, AUTO
	SYMNAMES={lib.sublib.member.type | member.type | SYSIPT}	
	SYMNLIB=(lib$_1$.sublib$_1$[,lib$_2$.sublib$_2$]...[,lib$_{14}$.sublib$_{14}$])	

Table 1. (Page 9 of 14) Control Table Summary Chart

OPERATION NAME	PARAMETERS	DELIVERED DEFAULT
OPTION	SYMNOUT={LAL / LOG / LST / NONE}	
	TP	Accepted but ignored
	VERIFY	Accepted but ignored
	VSCORE={n / nK / nM}	VSCORE=4096K
	VSCORET={n / nK / nM}	VSCORET=8192K
	WORKNM=work	WORKNM=SORTWKx
	XDUPLAB={S / U}	Standard labels
	XDUPNM=filename	XDUPNM=SORTXDP
	XDUPOUT=nnn	Default symbolic unit number
	XSUMLAB={S / U}	Standard labels
	XSUMNM=filename	XSUMNM=SORTXSM
	XSUMOUT=nnn	Default symbolic unit number
	{NZDPRINT / ZDPRINT}	No conversion of ZD SUM results to printable number

Table 1. (Page 10 of 14) Control Table Summary Chart

OPERATION NAME	PARAMETERS	DELIVERED DEFAULT
OUTFIL	BLKSIZE=n	Unblocked records
	BUFLIM=n	BUFLIM=255
	BUFOFF=n	BUFOFF=0
	{CARDS=s / PAGES=p}	CARDS=32,767 PAGES=32,767
	CISIZE=n	
	CLOSE={NORWD / RWD / UNLD}	CLOSE=RWD
	{CONVERT / VTOF}	
	DISK	
	DUMP	
	ENDREC=n	
	ESDS	(Opt if managed-SAM)
	EXIT	
	FILES=(1[,2...,32])	One output file
	FNAMES={filename / (filename$_1$[,filename$_2$]...)}	One output file
	FREEOUT	Accepted but ignored
	FTOV	
	HEADER1=(field$_1$[,field$_2$]...)	No report heading
	HEADER2=(field$_1$[,field$_2$]...)	No page heading
	{INCLUDE / OMIT}={ALL / (comparison) / NONE}	Output all records
	KSDS	

Table 1. (Page 11 of 14) Control Table Summary Chart

OPERATION NAME	PARAMETERS	DELIVERED DEFAULT
OUTFIL	LINES={n | SYSTEM | ANSI | (ANSI,n) | (ANSI,SYSTEM)}	LINES=60 (print) LINES=0 (punch)
	LRECL=n	
	NODETAIL	Detailed report
	NOTPMK	Tape marks written
	OPEN={RWD | NORWD}	OPEN=RWD
	OUTPUT	Accepted but ignored
	{OUTREC | BUILD | OVERLAY | IFTHEN}=(…)	Record unchanged
	PRINT	Required for report writing
	PUNCH	
	REPEAT=n	
	REUSE	(Only if VSAM output)
	RRDS	
	SAMPLE={n | (n,m)}	
	SAVE	
	SECTIONS=($field_1$[,$field_2$]...)	No sections
	SPAN	Unspanned records
	SPLIT	
	SPLITBY=n	
	STARTREC=n	
	TAPE	
	TOL	(Only if VSAM output)

Table 1. (Page 12 of 14) Control Table Summary Chart

OPERATION NAME	PARAMETERS	DELIVERED DEFAULT
OUTFIL	TRAILER1=(field$_1$[,field$_2$]...)	No report trailer
	TRAILER2=(field$_1$[,field$_2$]...)	No page trailer
	VLFILL=f	
	VLTRIM=b	
OUTREC	{ FIELDS / BUILD / OVERLAY / IFTHEN } = (…)	Output records identical to input
RECORD	LENGTH=(l_1,...,l_7)	
	TYPE={ F / V / D / VS / VBS }	
	DELBANK={ (p,q) / p }	
REFORMAT	FIELDS=(Fn:p_1,l_1[,Fn:]p_2,l_2...[[,Fn:]p_m][,Fn:p_n])	
	FILL=f	

Table 1. (Page 13 of 14) Control Table Summary Chart

OPERATION NAME	PARAMETERS	DELIVERED DEFAULT
SORT	{FIELDS=(p_1,l_1[,f_1],o_1,…,p_{64},l_{64}[,f_{64}],o_{64})[,FORMAT=f] / FIELDS=COPY}	
	BIAS=n	Accepted but ignored
	CENTWIN={s / f}	s=80
	{CHKPT / CKPT}	
	{EQUALS / NOEQUALS}	NOEQUALS
	ERASEWK	NOERASE
	FILES=n	FILES=1 FILES=2 (for JOIN)
	FILESOUT=n	FILESOUT=1
	JOINWORK=n	JOINWORK=1
	SIZE={n / E1}	Sort Capacity
	WORK={DA / n / (n,s)}	WORK=DA
SUM	{FIELDS=(p_1,l_1[,f_1][,p_2,l_2[,f_2]]...)[,FORMAT=f] / FIELDS=NONE}	No summing of fields; no reduction of equal-keyed records
	XSUM	Records deleted by SUM processing are not written to a file
XDUPFIL	Same as OUTFIL except EXIT, FILES=, and FNAMES=	Same as OUTFIL
XSUMFIL	Same as OUTFIL except EXIT, FILES=, and FNAMES=	Same as OUTFIL

Table 1. (Page 14 of 14) Control Table Summary Chart

Data Utility Processing Sequence

Figure 3 on page 2.18 presents the sequence in which SyncSort control statements and parameters are processed. It includes those statements and parameters that modify the input file (e.g., INPFIL, 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.

The figure also illustrates the steps in processing where several record length parameters are used to describe the records. l_{min} and l_{max} are parameters of INPFIL LRECL; l_1, l_2, l_3, l_4, and l_5 are parameters of RECORD LENGTH.

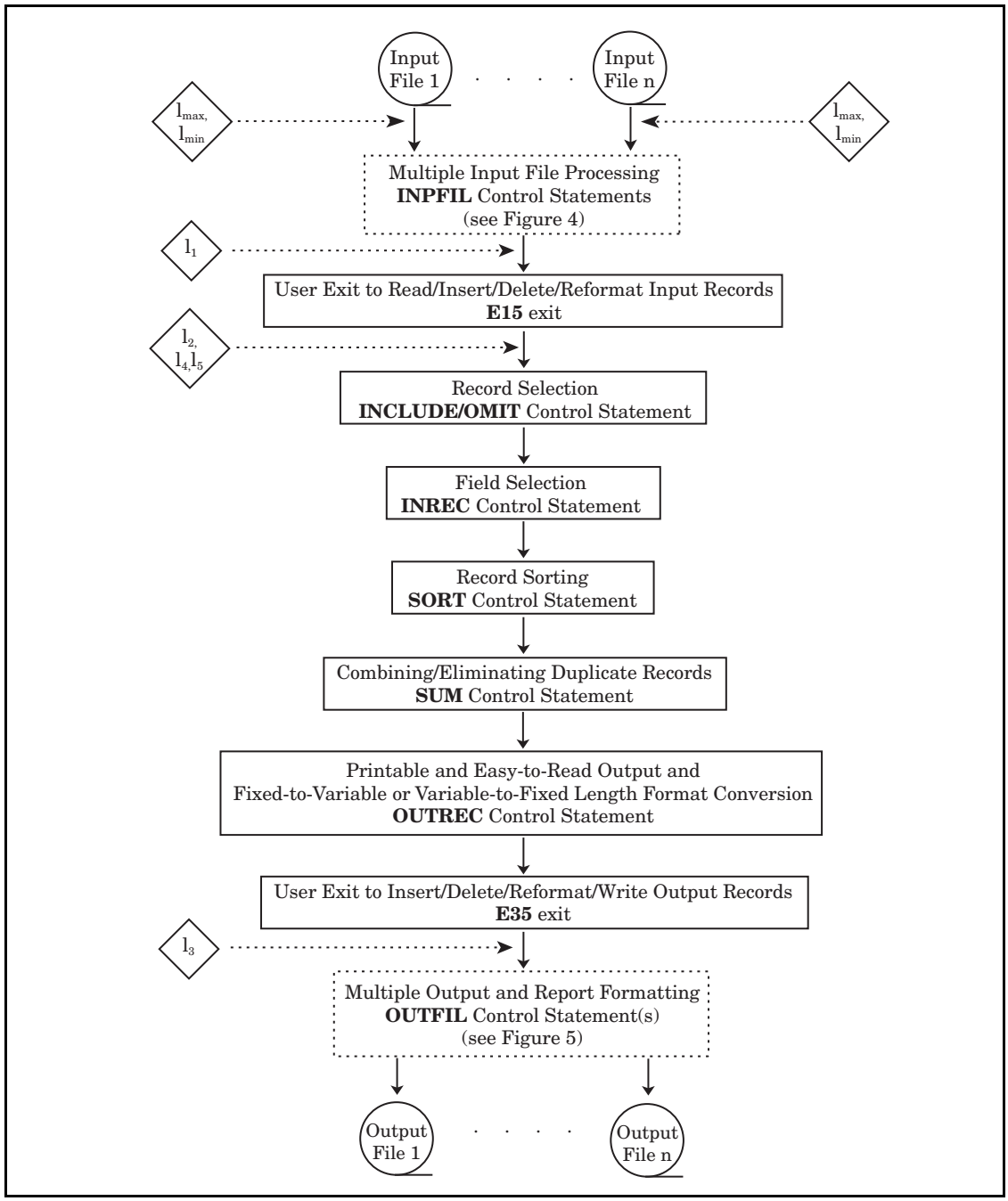


Figure 3. Data Utility Processing Sequence

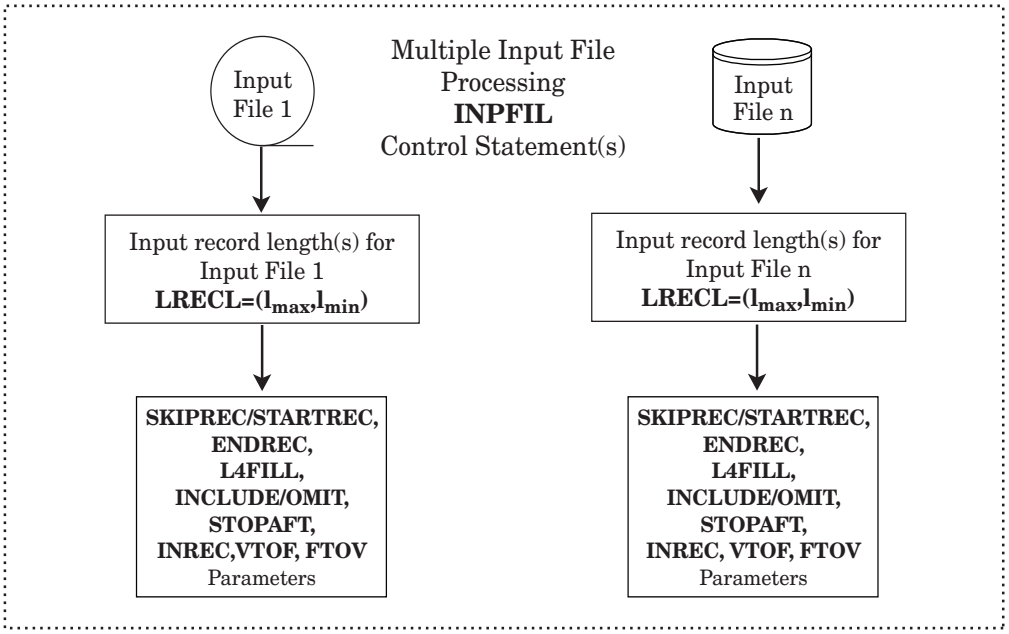


Figure 4. The INPFIL Control Statement Sequence

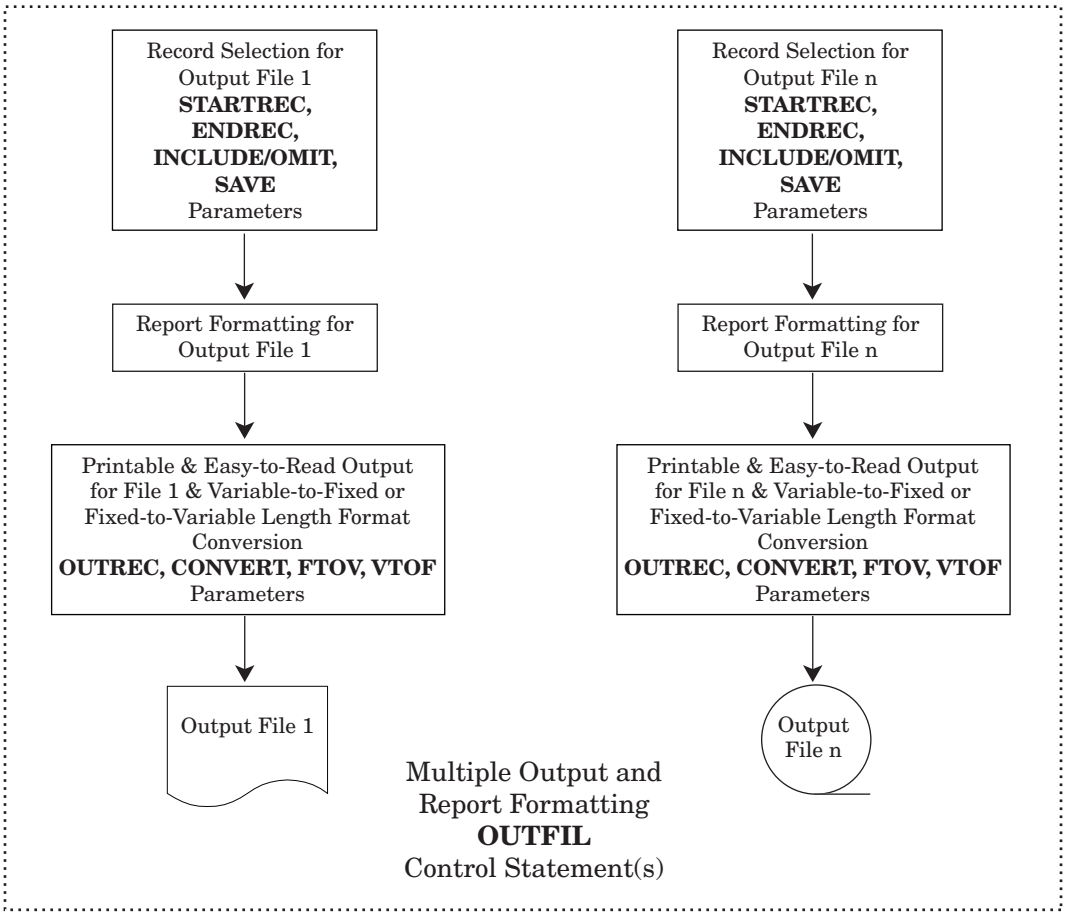


Figure 5. The OUTFIL Control Statement Sequence

Data Utility Processing Sequence

1. The INPFIL control statement describes the input file(s) and details certain processing options. In a JOIN application, the JOINKEYS control statements are used instead of the INPFIL control statement to describe the input files for JOIN and to detail certain processing applications.

2. 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.

3. The INREC control statement reformats the input records by adding, deleting, or reformatting fields *before* the records are sorted or merged.

4. The SORT control statement defines the application as a sort. It can also define a copy application.

5. The SUM control statement deletes records with equal control fields and optionally sums specified numeric fields on those records. Alternatively, the DUPKEYS control statement may be used, which includes all functions of the SUM control statement and more.

6. The OUTREC control statement reformats the output records by:

   - Deleting or repeating segments of the input records
   - Inserting character strings between data fields
   - Inserting binary zeros
   - Converting numeric data to printable format or to another numeric format
   - Performing arithmetic operations and minimum and maximum functions with numeric fields and constants
   - Converting data to printable hexadecimal format
   - Changing the order of or completely redesigning the original input records
   - Selecting, realigning, and reordering data fields

7. The OUTFIL control statement describes the output file(s) and:

   - Creates multiple output files
   - Uses the SortWriter facility

- Reformats records after E35 processing

- Converts a variable-length record input file to a fixed-length record output file or a fixed-length record input file to a variable-length record output.

INPFIL Processing Sequence

1. The INPFIL control statement is the initial function of SyncSort. Its specific processing varies, depending on parameters specified on INPFIL. If EXIT is specified, SyncSort gets the input records from an E15/E32 user exit routine and all other INPFIL parameters are ignored. If not, SyncSort processes each of the INPFIL parameters in sequence as depicted in Figure 4 on page 2.19.

2. If the SKIPREC, STARTREC, and ENDREC parameters are specified, SyncSort processes the functions as follows:

   - SKIPREC - skips a specified number of records per input file before the file is sorted, merged, or copied.

   - STARTREC - specifies the first record that will be processed for this input file.

   - ENDREC - specifies the last record that will be processed for this input file.

3. If the INCLUDE/OMIT parameter is specified, SyncSort performs processing of the INCLUDE/OMIT function by selecting from an input file based on comparison of the contents of one or more fields within the record.

4. If the STOPAFT parameter is specified, SyncSort performs processing of the STOPAFT function by ceasing to pass the records to the SORT/MERGE when the specified number of records has been reached.

5. If the INREC, BUILD, OVERLAY, or IFTHEN parameter is specified, SyncSort performs processing of that function by reformatting the input records of the input file.

6. If the VTOF parameter is specified, variable-length records will be converted to fixed-length records. If the FTOV parameter is specified, fixed-length records will be converted to variable-length records.

OUTFIL Processing Sequence

1. The OUTFIL control statement is the final function of SyncSort. Its specific processing varies, depending on parameters specified on OUTFIL. SyncSort processes each of the OUTFIL parameters in sequence as depicted in Figure 5 on page 2.19.

2. If the STARTREC and ENDREC parameters are specified, SyncSort processes the functions as follows:

   - STARTREC - specifies the first sorted record that is generated to the output file.

- ENDREC - specifies the last sorted record that is generated to the output file.

3. If the INCLUDE/OMIT parameter is specified, SyncSort performs processing of the INCLUDE/OMIT function by excluding records from an output file based on comparison of the contents of one or more fields within the record.

4. If the SAVE parameter is specified, SyncSort will include all the records that are not included in another OUTFIL group in the data sets for this group. SAVE retains all the input records that are not included in the other groups and would otherwise be lost.

If the OUTREC, BUILD, OVERLAY, or IFTHEN parameter is specified, SyncSort performs processing of that function by reformatting the output records of the output file.

If the CONVERT or VTOF parameter is specified, variable-length records will be converted to fixed-length records. If the FTOV parameter is specified, fixed-length records will be converted to variable-length.

Maximum Record Length Allowed

Certain control statements and user exits can adjust the length of the records at several points during sort processing. (See Figure 3. “Data Utility Processing Sequence” on page 2.18.) The maximum record lengths allowed are constrained by both the types of input, sortwork, and output devices used, and by the types of files used.

Observe the following device blocksize restrictions on record lengths:

1. Input files: for fixed-length records, the input records can be no longer than the maximum blocksize of the SORTIN device(s). For nonspanned variable-length records, the limit is 4 bytes less than the maximum blocksize.

2. Sortwork files: for fixed-length records, the length of the records processed at record sorting time (i.e. after any length adjustments due to INPFIL, E15, and/or INREC processing) can be no larger than 16 bytes less than the smallest maximum blocksize of the SORTWK devices used.

3. Output files: for fixed-length records, the output records (i.e. after any further length adjustments due to OUTREC, E35, and/or OUTFIL processing) can be no larger than the maximum blocksize of the SORTOUT device(s). For nonspanned variable-length records, the limit is 4 bytes less than the maximum blocksize.

The maximum blocksizes for several devices are listed in Table 2.

Device	**Maximum Blocksize**
FBA	32761
9345	46456
3380	47476
3390	56664
Tape	65535

Table 2. Maximum Blocksizes by Device

For ASCII tape records, the maximum blocksize is restricted to 9999 bytes.

Also, observe the device and file type restrictions on the maximum size of fixed-length and variable-length records shown in Table 3.

Device/File Type	**Maximum Record Length**	
	Fixed-Length	**Variable-Length**
FBA sequential disk	32761	32757
CKD sequential disk	max. blocksize	(max blocksize - 4) [c] 65535[b]
VSAM	32761[c] 65535[a]	32765[c] 32767[a]
SAM ESDS	32761	32757[c] 65535[b]
Tape	65535	65531[c] 65535[b]
[a] with SPANNED in DEFINE CLUSTER statement [b] with SPAN on sort's OUTFIL control statement [c] unspanned records		

Table 3. Maximum Record Lengths by Device and File Type

Control Statement Examples

Simple examples illustrating the syntax of each of the SyncSort for z/VSE control statements are included in this chapter. More complex applications are presented in "Chapter 4. How to Use SyncSort Data Utility Features". These applications demonstrate how the INPFIL, 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/VSE control statements.

Specifying Control Statements

- Control statements can be in any order, except for the END statement which, if specified, must be last.
- The control statement name can begin in column 2 through column 71. If labels are used, the control statement name can begin in column 3 through 71.
- The control statement name must be the first field of the first card image of the statement. It cannot be continued on a continuation card image.
- The control statement name must be preceded and followed by at least one blank.
- Each control statement, except for INPFIL/OUTFIL, can be specified only once for a particular application.
- Each application can include up to 255 card images (including continuation card images).
- Specify the OPTION control statement as the first card image if you want to override the PRINT and ROUTE defaults prior to all processing or if you want to use SyncSort's dictionary feature.

Specifying Parameters

- Parameters can take three forms:
  - Parameter
  - Parameter=value, Parameter=(value) *or* Parameter(value)
  - Parameter=($value_1$,$value_2$,...,$value_n$) *or* Parameter($value_1$,$value_2$,...,$value_n$)

Note: Multiple values must be enclosed in parentheses.

- Parameters can be in any order, but 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 a literal string, a parameter value cannot exceed eight alphanumeric characters.
- With the exception of a literal string, blanks are not permitted within parameters, and blanks, commas, equals signs, and parentheses are not permitted within parameter values. Parentheses within literal strings must be balanced.

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. For example, a position value of 7.4 designates a field beginning on the fifth bit of the seventh byte (the 8 bits in each byte are numbered 0 through 7). A length value of 7.4 designates a field seven bytes, four bits long.
- 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 71.

- A comment card image can be inserted between a control statement and its continuation.

- To add a comment to a control statement card image, leave one or more blanks after the last parameter 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.

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 combination or a complete parameter value before column 72. Begin the continuation on the next card image anywhere between columns 2 and 71. No continuation character is required.

--or--

- When the control statement extends through column 71 and cannot be broken at a parameter-comma 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 statement on the next card image anywhere between columns 2 and 16.

  - 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,FILESOUT=2,
               WORK=3
```

Figure 6. Continuing a Control Statement Without Specifying a Continuation Character

In Figure 6, no continuation character is required. The control statement is interrupted after a complete parameter value before column 72.

```
            COL. 16                                                COL.72
            ↓                                                         ↓
OUTFIL OUTREC=(1:10,8,30:40,10,50:75,25),HEADER2=(1:'DEPARTMENT NUMBX
            ER',30:'ITEM NUMBER')
```

Figure 7. Continuing a Control Statement with a Continuation Character

In Figure 7, 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/VSE 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.
- Labels must begin in column 1 with an alphabetic character.
- Labels can be any length provided the other rules that 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/VSE Programmer's Guide

- Braces { } indicate that a choice must be made from the alternatives listed.
- Brackets [ ] indicate an optional item. Two or more vertically listed 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 that must be replaced by actual values.
- Subscripts show position in a series, and three dots indicate an ellipsis.

  For example, $a_1,a_2,...,a_5$ is equivalent to a_1,a_2,a_3,a_4,a_5 and represents five "a" items (variables that 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)
```

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 or INCLUDE/OMIT 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 an ALTSEQ statement. AQ can be specified for one or more control fields so that those control fields all use the same alternate collating sequence.

ALTSEQ Control Statement Format

The format of the ALTSEQ statement is illustrated below:

```
ALTSEQ CODE=(ccpp1,...,ccpp256)
```

Figure 8. 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.

Each character (cc entry) can be moved only one time. However, a character represented by the pp value can be repositioned as a cc entry anywhere in the series.

Sample ALTSEQ Control Statements

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

Figure 9. Sample ALTSEQ Control Statement

The ALTSEQ statement in Figure 9 shows that the numbers 0 through 9 are to collate before the uppercase alphabet.

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

Figure 10. Sample ALTSEQ Control Statement

This ALTSEQ statement in Figure 10 specifies that the number 0 is to collate as equal to a blank (X'40').

ANALYZE Control Statement

The ANALYZE control statement determines how much disk work space a sort will require before it executes.

Specify the ANALYZE control statement only when a sort is initiated from JCL. The ANALYZE statement cannot be specified for a merge. The format of the ANALYZE statement is illustrated below.

```
ANALYZE CALC
```

Figure 11. ANALYZE Control Statement Format

CALC Parameter (Required)

The CALC parameter instructs SyncSort for z/VSE to determine how much disk work space a sort will require. It sets the following parameters of the OPTION control statement when the SIZE parameter is also specified on the SORT control statement:

- CALCAREA
- DIAG
- NODUMP
- PRINT=ALL
- ROUTE=LST

As a result, SyncSort will process the control statements and then terminate (without sorting). SyncSort will print the number of tracks (for CKD devices) or blocks (for FBA devices) required for work areas.

The SIZE parameter on the SORT statement must provide an accurate estimate of the number of input records in order for the CALCAREA information to be meaningful.

DUPKEYS Control Statement

The DUPKEYS control statement deletes all records with duplicate SORT/MERGE control fields and optionally replaces specified numeric fields in the retained record with calculated average, calculated sum, minimum, or maximum values from all records with equal control fields. The deleted records can optionally be written to a separate output file.

The DUPKEYS control statement cannot be used along with a SUM control statement, or when FIELDS=COPY is specified on the SORT or MERGE control statement, or when FIELDS=COMPARE is specified on the MERGE control statement.

If you need to add AVG, MAX, or MIN functionality to an existing application with a SUM control statement, you must move the SUM specification to the DUPKEYS statement, and remove the SUM statement. If you used XSUM, then specify XDUP on the DUPKEYS statement, and define a SORTXDP output file instead of the SORTXSM file.

The format of the DUPKEYS control statement is illustrated below.

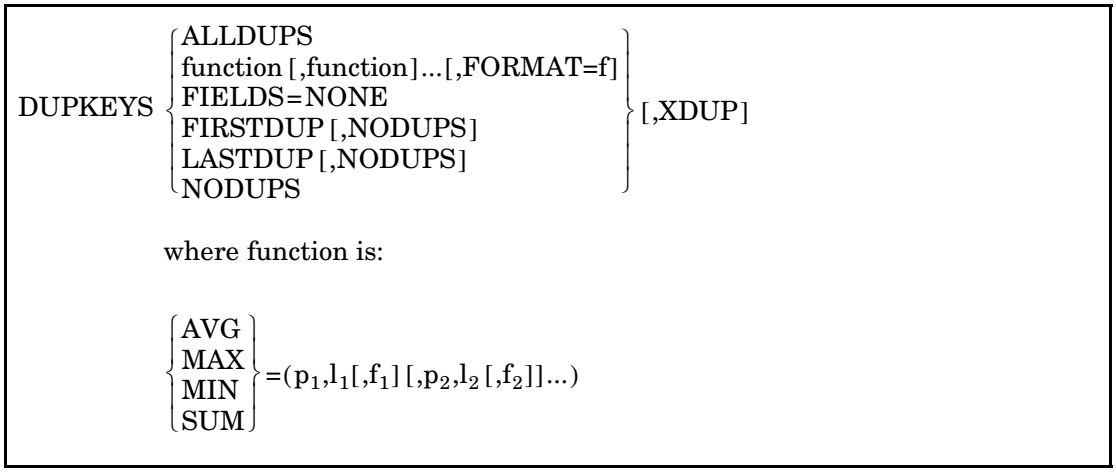


Figure 12. DUPKEYS Control Statement Format

Arithmetic functions, FIELDS=NONE, ALLDUPS, FIRSTDUP, LASTDUP and NODUPS are all mutually exclusive parameters, except that NODUPS can be specified along with FIRSTDUP or LASTDUP.

If FIELDS=NONE is specified, all records with duplicate control fields are simply deleted. If arithmetic functions are desired then the AVG, MAX, MIN, or SUM parameter must be used, along with the optional FORMAT parameter. The AVG, MAX, MIN, or SUM parameter specifies the position, length, and (optionally) the format of one or more numeric fields for the arithmetic function.

When the FORMAT parameter is specified, an individual AVG, MAX, MIN, or SUM field may omit a format value; in this case, the value specified by the FORMAT parameter is used for that field.

Each field specified in the AVG, MAX, MIN, and SUM parameters 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 integral number of bytes. Refer to Table 4 for the permissible lengths.

f The optional format value indicates the data format. Table 4 displays the valid formats. If the format value is omitted, then the value in the FORMAT parameter is used.

FORMAT CODE	PERMISSIBLE LENGTH		
	SUM Fields	MIN or MAX Fields	AVG Fields *
BI	2, 4, or 8 bytes	1 to 256 bytes	2, 4, or 8 bytes
FI	2, 4, or 8 bytes	2, 4, or 8 bytes	2, 4, or 8 bytes
FL	4, 8, or 16 bytes	4, 8, or 16 bytes	4, 8, or 16 bytes
PD	1 to 16 bytes	1 to 16 bytes	1 to 10 bytes
ZD	1 to 31 bytes	1 to 31 bytes	1 to 18 bytes
* 8-byte BI, FI and 16-byte FL AVG fields require an ESA-mode VSE machine.			

Table 4. Allowed DUPKEYS Field Lengths

ALLDUPS Parameter (Optional)

The ALLDUPS parameter specifies that only records with SORT/MERGE fields that occur more than once are retained.

AVG Parameter (Optional)

Use the AVG parameter to specify numeric fields to retain the average value among all records of the same control fields. Multiple fields separated by commas may be specified in the same parameter. Multiple AVG parameters may be specified on the same DUPKEYS control statement.

- Adding AVG fields to an existing sort application may result in an increase in the amount of SORTWORK space required. This occurs because AVG postpones all DUPKEYS processing until phase 3.

- BI, FI, PD, and ZD AVG fields contain average values truncated to integers. For example, the average value of (1,1,1,1,0) is 0 and the average value of (-1,-1,-1,0) is also 0.

FIELDS Parameter (Optional)

The only valid value for FIELDS is NONE. Specify FIELDS=NONE only if no arithmetic functions are desired. The sorted data will be reduced to one record per sort key value.

FIRSTDUP Parameter (Optional)

The FIRSTDUP parameter specifies that only the first record of those with SORT/MERGE fields that occur more than once is retained. If the NODUPS parameter is also specified, all records with SORT/MERGE fields that occur exactly once are also retained.

FORMAT Parameter (Optional)

Use the FORMAT parameter to specify the default field format for fields specified in the AVG, MAX, MIN, and SUM parameters. If any field in the AVG, MAX, MIN, or SUM parameter does not include the format value (f), then this default format value applies to that field.

LASTDUP Parameter (Optional)

The LASTDUP parameter specifies that only the last record of those with SORT/MERGE fields occurring more than once is retained. If the NODUPS parameter is also specified, all records with SORT/MERGE fields occurring exactly once are also retained.

MAX Parameter (Optional)

Use the MAX parameter to specify numeric fields to retain the maximum value among all records with the same control fields. Multiple fields separated by commas may be specified in the same parameter. Multiple MAX parameters may be specified on the same DUPKEYS control statement.

MIN Parameter (Optional)

Use the MIN parameter to specify numeric fields to retain the minimum value among all records with the same control fields. Multiple fields separated by commas may be specified in the same parameter. Multiple MIN parameters may be specified on the same DUPKEYS control statement.

NODUPS Parameter (Optional)

The NODUPS parameter specifies that only records with SORT/MERGE fields that occur exactly once are retained.

SUM Parameter (Optional)

Use the SUM parameter to specify numeric fields to contain the summed value among all records with the same control fields. Multiple fields separated by commas may be specified in the same parameter. Multiple SUM parameters may be specified on the same DUPKEYS control statement.

XDUP Parameter (Optional)

Specify the XDUP parameter if you want the records deleted by DUPKEYS processing to be written to a file named SORTXDP (this name may be changed by the XDUPNM parameter in the OPTION statement). These records will be written to SORTXDP at the time of DUPKEYS processing. These records will not undergo OUTREC, E35, and OUTFIL processing as such processing occurs after DUPKEYS processing.

When the XDUP parameter is specified, at least one additional parameter also must be specified.

Characteristics of the SORTXDP file, such as BLKSIZE, can be specified with the XDUPFIL control statement. The default is unblocked output.

The SORTXDP file will be sequenced in the same order as the SORTOUT file.

Note that XDUP may increase system resource requirements:

- Adding XDUP to an existing sort application may result in an increase in the amount of SORTWORK space required. This occurs because XDUP delays all DUPKEYS processing until phase 3.
- XDUP may require additional storage. Do not specify a VSCORE value less than 512K on the OPTION control statement.

Rules for Specifying DUPKEYS

- When using arithmetic functions or FIELDS=NONE, if EQUALS is in effect, the record that is retained is the first record read for that sort key. If NOEQUALS is in effect, the record that is retained is arbitrarily determined by SyncSort.
- An AVG, MAX, MIN, or SUM field cannot include any or part of a SORT or MERGE control field, nor the first four bytes of a variable-length record that contains the Record Descriptor Word.
- AVG, MAX, MIN, and SUM fields cannot overlap each other.
- If arithmetic overflow occurs during the summing or averaging of two records, those records are not summed or averaged and neither record is deleted. All other DUPKEYS functions are also suspended between those two records. AVG, MAX, MIN, and SUM

arithmetic restarts when a subsequent set of records with equal control fields can be summed or averaged without overflow. To avoid arithmetic overflow with SUM, use the INREC control statement to insert binary zeros (or X'F0's if ZD) immediately before the SUM field.

- Refer to "OVFLO Parameter (Optional)" on page 2.131 for other options for overflow processing.

Sample DUPKEYS Control Statement

The following DUPKEYS statement deletes records with equal control fields but places arithmetic sum, minimum, maximum, and average values of some fields in the retained record.

```
DUPKEYS SUM=(20,8,32,4,FI),MIN=(40,6),MAX=(48,6),AVG=(54,6),FORMAT=ZD
```

Figure 13. Sample DUPKEYS Control Statement

When the control fields are equal, this statement sums the ZD field beginning in byte 20 and the FI field beginning in byte 32, selects the minimum value of the ZD field beginning in byte 40, the maximum value of the ZD field beginning in byte 48, calculates the average value of the ZD field beginning in byte 54, and then deletes the equal-keyed record.

END Control Statement

If present, the END control statement must be the last control statement. The END control statement is required only with card input.

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

When SORTIN is a card (SYSIPT) file, the card file must follow immediately after the END control statement. A /* (end-of-data) card must follow the last SORTIN card.

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. A binary field may enter into comparisons that involve testing the individual bits in the field; and substrings, including wildcards, may be specified. Only one INCLUDE/OMIT control statement can be specified for an application, either as an INCLUDE *or* as an OMIT statement.

If you specify an INCLUDE or an OMIT statement, do not use the DELBLANK parameter on the RECORD statement.

LOCALE-Based Comparison Processing

SyncSort supports alternate sets of collating rules based on a specified national language. The alternate collating applies to INCLUDE/OMIT (as well as to INPFIL 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 Parameter (Optional)” on page 2.128.

INCLUDE/OMIT Control Statement Format

The format of the INCLUDE/OMIT statement follows.

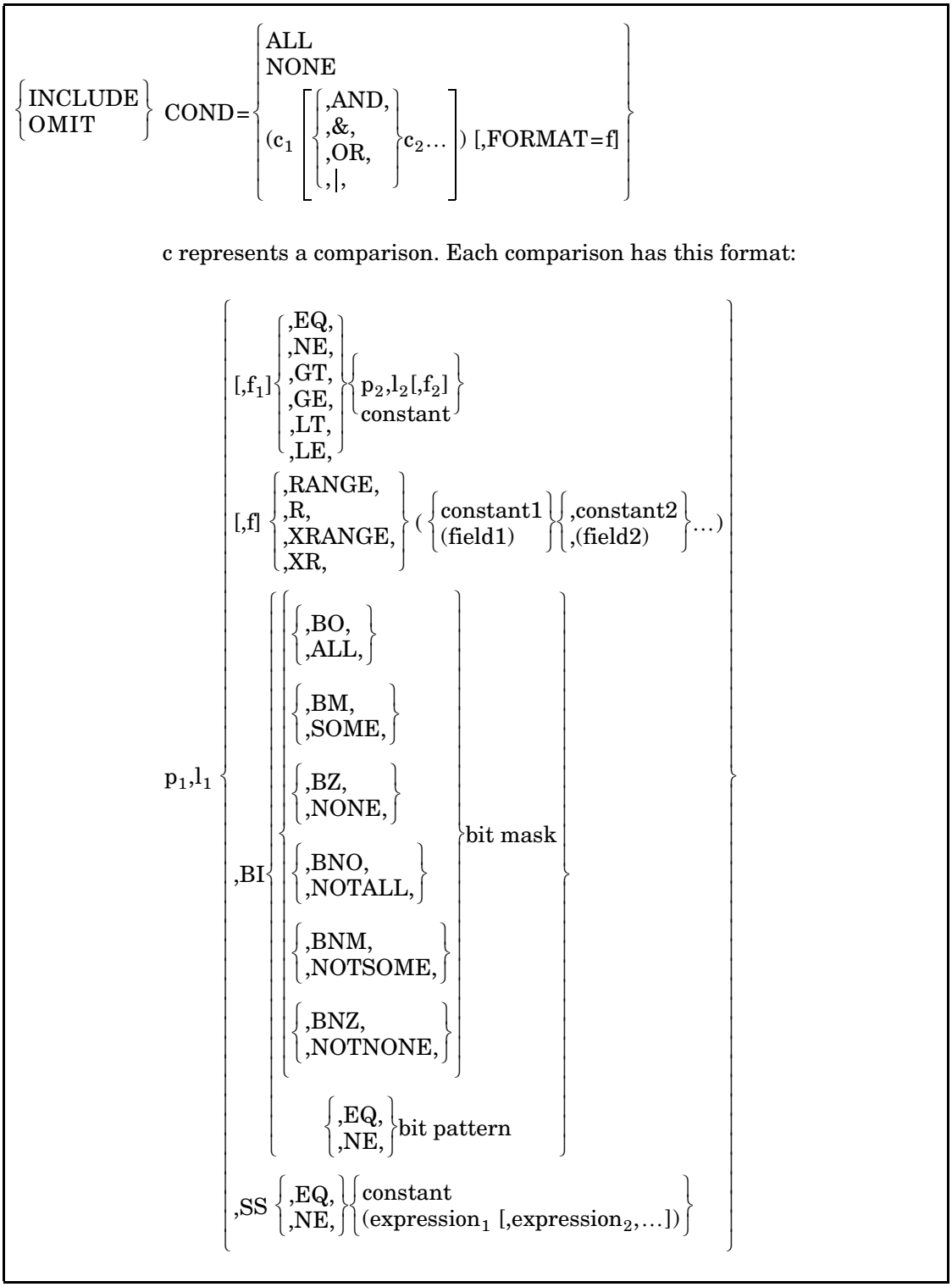


Figure 14. INCLUDE/OMIT Control Statement Format

where *expression* may be either a wildcard-plus-constant or constant-plus-wildcard combination, as follows:

Figure 15. Constant/Wildcard Expression

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 or omitted (depending on the control statement). This is the default.

COND=NONE None of the input records are to be included or omitted (depending on the control statement).

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** that allows the search for a constant within a field or for a field value within a constant.

The following several pages describe standard comparisons. For information on substring comparisons, see "Substring Comparisons" on page 2.55.

Each field specified in the COND 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* 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 type of the field. The supported formats for standard comparisons are all those indicated in Table 5 on page 2.43 except for SS. 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.)

R/RANGE This keyword indicates that a condition is true for a specified range of values. One or more pairs of range values can be specified.

XR/XRANGE This keyword indicates that a condition is true outside a specified range of values. One or more pairs of range values can be specified.

constant1 This constant specifies the starting value. This value must be less than the ending value of constant2 or field2.

constant2 This constant specifies the ending value. This value must be greater than the starting value of constant1 or field1.

field1 This first field contains the starting value to be compared. This value must be less than the ending value of constant2 or field2. It has three variables (p_1[, l_1, f_1]). p_1 is the position of the field relative to the beginning of the record. l_1 is the optional length value indicating the length of the field. If omitted, it will use the length defined in field 1 (l). f_1 is the optional format type of the field. If omitted, it will use the format defined in field 1 (l).

field2 This second field contains the ending value to be compared. This value must be greater than the starting value of constant1 or field1. It has three variables (p_2[, l_2, f_2]). p_2 is the position of the field relative to the beginning of the record. l_2 is the optional length value indicating the length of the field. If omitted, it will use the length defined in field 1 (l). f_2 is the optional format type of the field. If omitted, it will use the format defined in field 1 (l).

Notes:

1) A constant will be padded or truncated to the length of the field to which it is compared.

2) Left and right parentheses are mandatory for field1 and field2.

Data Format	Acceptable Field Length (Bytes)
AC	1 to 256
AQ	1 to 256
ASL	1 to 256
AST	1 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 255
PD0	2-8
SFF	1 to 44
UFF	1 to 44
Y2B	1
Y2C / Y2Z	2
Y2D	1
Y2P	2
Y2S	2
ZD	1 to 256
SS	1 to 32767

Table 5. Formats and Lengths of Include/Omit Fields

For definitions of the field formats in the above table, see Table 7 on page 2.47. The following table shows the full-date data formats:

Full-Date Format	Date 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	dddyy	5
		xxxxyy	mmddyy	6
Y2X	PD	xyy (X'xyys')	qyy	2
		xxxyy (X'xxxyys')	dddyy	3
Y2Y	PD	xxyy (X'0xxyys')	mmyy	3
		xxxxyy (X'0xxxxyys')	mmddyy	4

Note: The following symbols are used in the table:

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

Table 6. Valid Formats and Lengths of Include/Omit Fields: Full-Date Formats

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

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.

hexadecimal A hexadecimal constant should be preceded by an X and specified in pairs of valid hexadecimal values that must be enclosed in single quotes: X'hh...hh'. For example, X'ACBF05' is a valid hexadecimal constant. If a hexadecimal constant is compared to a CH field and LOCALE is in effect, the hexadecimal constant is governed by the LOCALE rules.

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.

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. If a character constant is compared to a CH field and LOCALE is in effect, the character constant is governed by the LOCALE rules.

Y constant A Y constant should be preceded by a Y and enclosed in single quotes: Y'yyx...x', where yy is a 2-digit year value. For example, Y'991225' is a valid Y constant. A Y constant may only be used with date data formats.

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'bbbbbbbb...bbbbbbbb', where each 'bbbbbbbb' 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 a 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 Standard Comparisons

The following rules apply to comparisons:

Terms within comparisons are evaluated as follows:

- Any number of comparisons can be specified.
- Multiple comparisons must be separated by ANDs or ORs.
- The symbols & and | may be used for AND and OR.
- Inner parentheses are evaluated first.

- AND conditions are evaluated before OR conditions.
- The following numeric data compare as equal: +0, -0, 0.
- Up to 24 levels of internal parentheses are allowed.

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. Table 7 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**	**SFF**	**UFF**	**ZD**
AC	X																
AQ		X															
ASL			X	X													
AST			X	X													
BI					X	X											
CH					X	X											
CLO OL							X	X	X	X	X				X	X	
CSF FS							X	X	X	X	X				X	X	
CSL LS							X	X	X	X	X				X	X	
CST TS							X	X	X	X	X				X	X	
CTO OT							X	X	X	X	X				X	X	
FI												X					
PD													X				X
PD0														X			
SFF							X	X	X	X	X				X	X	
UFF							X	X	X	X	X				X	X	
ZD													X				X

Table 7. Permissible Field-to-Field Comparisons for Non-Date Data Formats

For information on comparisons involving full-date data, see “Specifying Standard Comparisons for Date Fields” on page 2.50.

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.

Specifying Field-to-Constant Standard Comparisons for Non-Date Fields

The format of a data field determines the type of constant to which it can be compared. Table 8 below illustrates which field-to-constant comparisons are permitted.

Format	Decimal	Hexadecimal	Character	Binary (bit pattern)
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		
SFF	X			
SS		X	X	
UFF	X			
Y2B	X			
Y2C / Y2Z	X			
Y2D	X			
Y2P	X			
Y2S	X			
ZD	X			

Table 8. Permissible Field-to-Constant Comparisons

Table 8 above does not include full-date formats. For information on comparisons involving full-date formats, see "Specifying Standard Comparisons for Date Fields" below.

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'
EBCDIC Character string	X'40'
ASCII Character string	X'20'
Hexadecimal string	X'00'
Decimal fields	Zeros of proper format. Decimal constants for all Y2 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. 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.

Specifying Standard Comparisons for Date Fields

The date 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 belongs when processed by INCLUDE/OMIT and other control statements.

The date data formats that can be used with INCLUDE/OMIT and OUTFIL INCLUDE/OMIT control statements are of two types, 2-digit year formats and full-date formats, as follows:

- The 2-digit year formats are Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z.
- The full-date formats are Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y.

Date data format fields in a record can be compared to a constant or to another field in the record. For field-to-field comparisons, date data format fields can only be compared to other date data format fields.

Any of the comparison operators (EQ, NE, GE, LT, and LE) can be used.

The date 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 Parameter (Optional)” on page 2.101. For details on the date data formats, see “CENTWIN (Century Window) Processing with SORT: 2-Digit Year Formats” on page 2.235 and “CENTWIN (Century Window) Processing with SORT: Full-Date Formats” on page 2.240. For examples of INCLUDE statements not

involving date data, see “Sample INCLUDE/OMIT Control Statements without Date Data” on page 2.57. For examples of INCLUDE statements involving date data, see “Sample INCLUDE/OMIT Control Statements with Date Data” on page 2.60.

Table 9 summarizes the valid field-to-field comparisons for 2-digit year data formats:

	Y2B	**Y2C/Y2Z**	**Y2D**	**Y2P**	**Y2S**
Y2B	X				
Y2C/Y2Z		X	X	X	
Y2D		X	X	X	
Y2P		X	X	X	
Y2S					X

Table 9. Valid Comparisons with 2-Digit Year Formats

For full-date formats, two types of comparisons are available:

Field-to-field Compares ZD, PD, and BI date fields.

Field-to-constant Compares a date field to a Y date constant.

The sequencing of date and non-date characters is the same as for an ascending sort.

A full-date data field is comprised of a year field plus additional date components. You can compare a full-date data field to another full-date data field or to a Y date constant. In both cases, the compared elements must have the same number of non-year (x) digits.

Thus, you can compare a field of the form P'xxxxyy' to a field of the form Z'xxxxyy' or P'yyxxxx' but not Z'xxyy'. Similarly, you can compare a full-date data field of the form C'yyxxx' or P'xxxyy' to a Y date constant of the form Y'yyxxx' but not Y'yyxx'.

Note that the position of the year components need not be the same for the items being compared. Although compared date constants must be in the form yyxx... (two-digit year, month, day -- with month and day optional), the full-date formats take care of this internally.

CH and ZD formats are represented by X'FdFd...sd', and PD formats are represented by X'dd...ds', where “d” is a decimal digit (0-9) and “s” is a sign (0-F). For dates, the sign is not relevant and can be ignored.

Table 6 on page 2.44 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)

Table 10 below indicates the valid field-to-field and field-to-constant comparisons for full-date fields.

Date Form	**Full-Date Format**	**Length (bytes)**	**Y Constant Form**
yyx xyy	Y2U Y2X Y2T Y2W	2 2 3 3	Y'yyx'
yyxx xxyy	Y2V Y2Y Y2T Y2W	3 3 4 4	Y'yyxx'
yyxxx xxxyy	Y2U Y2X Y2T Y2W	3 3 5 5	Y'yyxxx'
yyxxxx xxxxyy	Y2V Y2Y Y2T Y2W	4 4 6 6	Y'yyxxxx'
yy	Y2B Y2D Y2C Y2P Y2S Y2Z	1 1 2 2 2 2	Y'yy'

Table 10. Valid Comparison with Full-Date Formats

For constants, do not omit leading zeros. Thus, for a constant of the form Y'yymm', specify Y'0003' for March 2000 and specify Y'0203' for March 2002.

Y constants can also specify non-date values, as shown in Table 11:

Y Constant Form	Non-Date Character	Full-Date Format
Y'0...0'	CH, ZD, PD zeros	Y2T, Y2U, Y2V, Y2W, Y2X, Y2Y
Y'9...9'	CH, ZD, PD nines	
Y'LOW'	BI zeros	Y2T, Y2W, Y2S
Y'BLANKS'	Blanks	
Y'HIGH'	BI ones	

Table 11. Y Constant and Non-Date Characters

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.

The format of a current date constant is illustrated below.

Figure 16. Current Date Constant Format

where:

- 'current date constant' is in the form of one of the &DATEx, &DATEx(c), &DATExP, or Y'DATEx' parameters where x is 1, 2, 3, or 4 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 zeros truncated. When the x in &DATEx, &DATEx(c), &DATExP, or Y'DATEx' is 1, 3, or 4, 'nnnn' can be from 0-9999 and represents offset days. When the x in &DATEx, &DATEx(c), &DATExP, or Y'DATEx' is 2, 'nnnn' can be from 0-999 and represents offset months.

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

- &DATEx and &DATEx(c) represent the current date as a character string (C'string') to which a field can be compared.
- &DATExP represents the current date as a decimal number (+n) to which a field can be compared.
- Y'DATEx' 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 &DATEx(c) represents a non-blank separator character, except open and close parentheses.

Current Date Constant	Generated Constant
&DATE1	C'yyyymmdd'
&DATE1(c)	C'yyyycmmcdd'
&DATE1P	+yyyymmdd
&DATE2	C'yyyymm'
&DATE2(c)	C'yyyycmm'
&DATE2P	+yyyymm
&DATE3	C'yyyyddd'
&DATE3(c)	C'yyyycddd'
&DATE3P	+yyyyddd
&DATE4	C'yyyy-mm-dd-hh.mm.ss'
Y'DATE1'	Y'yymmdd'
Y'DATE2'	Y'yymm'
Y'DATE3'	Y'yyddd'

Table 12. 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 13).

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 13. Full-Date Comparisons

Substring Comparisons

Substring comparison 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 comparisons, constants can be from 1 to 256 bytes in length, while fields in the record can be from 1 to 32767 bytes in length. Constants can be in either character or hexadecimal format. (Refer to the description of constants just after Table 6 on page 2.44.) For a sample INCLUDE control statement with substring comparisons, see “Sample INCLUDE/OMIT Control Statements without Date Data” on page 2.57.

Wildcards can be used with substring comparisons to search a string pattern anywhere within a field.

- The length value (l) can be a wildcard (*). This means the entire record, beginning from the starting position (p), is searched for the specified substring.

  Note: For variable-length records, you can specify a position (p) that exceeds the minimum record length (l_4). This excludes all records shorter than l_4.

- The constant to which a field is compared can be in either character format (C'AIRLINE') or hexadecimal format (X'0DFF').

  Note: If two or more short constants are specified one after the other and separated by a comma without any wildcard characters, then the constants will be concatenated and treated as one long constant. For example, C'AIR',X'81',C'LAND',X'82' will be combined to produce a 9-byte constant C'AIRaLANDb'.

- A wildcard can be either an asterisk (*) or percent sign (%). An asterisk (*) represents zero or more characters, and a percent sign (%) represents exactly one character.

- Ordinarily, if the length of a constant is greater than the length of a specified field, SyncSort will signify an error. This is not the case with substring comparisons.

For sample INCLUDE control statements with substring wildcards, see “Sample INCLUDE/OMIT Control Statements without Date Data” below.

Sample INCLUDE/OMIT Control Statements without Date Data

This section includes sample INCLUDE/OMIT control statements without date data. For sample INCLUDE/OMIT control statements with date data, see "Sample INCLUDE/OMIT Control Statements with Date Data" on page 2.60.

Example 1

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

Figure 17. Sample INCLUDE Control Statement

In this example, records will be included in the application if the numeric data in the field beginning in byte 24 is less than the numeric data in the field beginning in byte 28 *or* if the character data 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 18. Sample OMIT Control Statement

In this example, records will be omitted from the application if the numeric data in the field beginning in byte 1 is equal to 100 *and* if the character data 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 19. Sample INCLUDE Control Statement Using a Bit Mask

The record selection condition has the following elements (from left to right): a binary field (BI) with a length of 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 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 01001000, 01111000, or 11111111, for example, 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 20. 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), or 01001000 (both the second and fifth bit are ON). However, with the string 10000111 in the input field, for instance, 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.

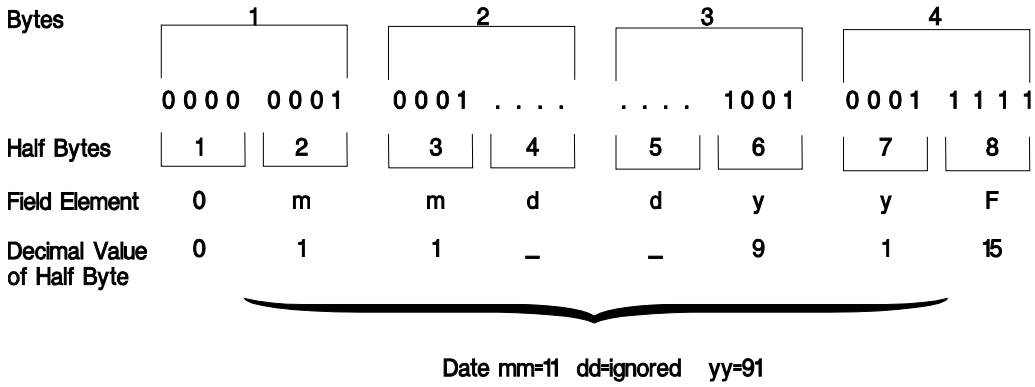
Example 5

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

Figure 21. 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'0mmddyyF'. 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 (**m**m,**d**d,**y**y). (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 (m**m**,d**d**,y**y**). (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.



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 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 22. 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 ('A02', 'A05', 'A06', or 'A09') in the constant.

Example 7

The following examples are variations of a substring comparison using wildcards. The following is the basic substring comparison, without wildcards:

```
INCLUDE COND=(11,40,SS,EQ,C'AIR')
```

Figure 23. Basic INCLUDE Control Statement Using Substring Compare

This control statement will search for 'AIR' anywhere in the 40-byte field starting at position 11.

Consider the following variation with the wildcard (*):

```
INCLUDE COND=(11,40,SS,EQ,(C'AIR',*,C'FLIGHT'))
```

*Figure 24. INCLUDE Control Statement Using Wildcard **

This control statement will search the 40-byte field for a string beginning with 'AIR' followed by any single character, multiple characters, or nothing and ending with 'FLIGHT'. The following variation uses the wildcard (%):

```
INCLUDE COND=(11,40,SS,EQ,(C'AIR',%%%%%,C'FLIGHT'))
```

Figure 25. INCLUDE Control Statement Using Wildcard (%)

This control statement will search the 40-byte field for a string beginning with 'AIR' followed by any five characters and ending with 'FLIGHT'.

The last variation uses a wildcard for the length (l) specification:

```
INCLUDE COND=(100,*,SS,EQ,(C'AIR',*,C'FLIGHT'))
```

Figure 26. INCLUDE Control Statement Using Wildcard (*) for Length

This control statement will search the record starting from position 100 to the maximum record length (L4 - 100 + 1). It searches for a string beginning with 'AIR' followed by any single character, multiple characters, or nothing and ending with 'FLIGHT'. All records shorter than 100 bytes will not be searched.

Sample INCLUDE/OMIT Control Statements with Date Data

Example 1

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

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

Figure 27. INCLUDE Control Statement with Y2C Format

In this example, only records whose data is 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 Position 20 and 21	Record Disposition
84	Omitted - represents 1984
99	Included - represents 1999
37	Included - represents 2037

Example 2

The following control statement selects records if the date field (20,6) of the form C'mmddyy' is Jan. 1, 1997 through Dec. 31, 2001. Records with a none-date value of C'999999' are omitted. The previously set century window is 1980-2079 (CENTWIN=1980).

```
MERGE FIELDS=COPY
OMIT FORMAT=Y2W,                      * Uses Y2W full-date format.
     COND=((20,6,GT,Y'011231'),OR,    * Keeps records with dates
           (20,6,LT,Y'970101'),OR,    * between Jan. 1, 1997
           (20,6,EQ,Y'9'))            * and Dec. 31, 2001.
```

Figure 28. OMIT Control Statement with Y2W Format

Table 6 on page 2.44 indicates that the 6-byte Y2W format is appropriate for a CH input field of the form C'mmddyy'. The OMIT control statement compares the 6-byte field starting in byte 20 to three different 6-byte character constants.

The following sample table indicates the include/omit status of input records:

SORTIN Input (C'mmddyy')	Record Disposition	Date Represented (yyyy/mm/dd)
021545	Omit	2045/02/15
999999	Omit	non-date
091899	Include	1999/09/18
062184	Omit	1984/06/21
070400	Include	2000/07/04

Example 3

The following control statement selects records if the date field (20,3) of the form P'yyddd' (X'yyddds') is between January 1, 1992 and December 31, 2007. Records with non-date value P'00000' are also selected.

The previously set century windows is 1980-2079 (CENTWIN=1980).

```
MERGE FIELDS=COPY
INCLUDE FORMAT=Y2U,                    * Uses Y2U full-date format.
       COND=((20,3,GE,Y'92001'),AND,   * Keeps records with dates
             (20,3,LE,Y'07365'),OR,    * between Jan., 1, 1992 and
             (20,3,EQ,Y'00000'))       * Dec. 31, 2007.
```

Figure 29. INCLUDE Control Statement with Y2U Format

Table 6 on page 2.44 indicates that the 5-byte Y2U format is appropriate for a PD input field of the form P'yydddd'. The INCLUDE control statement compares the 3-byte PD field starting in byte 20 to three different 5-byte character constants.

The following sample table indicates the include/omit status of input records:

SORTIN Input (P'yyddd')	**Record Disposition**	**Date Represented (yyyy/ddd)**
99237	Include	1999/237
00047	Include	2000/047
74123	Omit	2074/123
00000	Include	non-date
99999	Omit	non-date
87099	Omit	1987/099

Additional applications using the INCLUDE/OMIT control statement are illustrated in “Chapter 4. How to Use SyncSort Data Utility Features”. Sample INCLUDE/OMIT Range Control Statements

The following examples illustrate the typical range usages:

Example 1

To specify a range of dates to include:

```
INCLUDE COND=(10,4,Y2T,RANGE,(Y'9001',Y'0512'))
```

Figure 30. INCLUDE Control Statement with Y2T Format

This control statement would include all records with date fields in position 10, length 4 bytes, between January 1990 and December 2005.

```
INCLUDE COND=(10,4,Y2T,RANGE,((50),(60)))
```

Figure 31. INCLUDE Control Statement with Y2T Format

If field (50,4,Y2T) contains Y'9001' and field (60,4,Y2T) contains Y'0512', then this control statement would include all records with date fields in position 10, length 4 bytes, between January 1990 and December 2005.

```
INCLUDE COND=(10,4,Y2T,RANGE,(Y'9001',Y'9012',Y'9501',Y'9512'))
```

Figure 32. INCLUDE Control Statement with Y2T Format

This control statement would include all records with date fields in position 10, length 4 bytes, between January 1990 and December 1990 or between January 1995 and December 1995.

Example 2

To specify a range of dates to exclude:

```
INCLUDE COND=(10,4,Y2T,XRANGE,(Y'9001',Y'0512'))
```

Figure 33. INCLUDE Control Statement with Y2T Format

This control statement would exclude all records with date fields in position 10, length 4 bytes, between January 1990 and December 2005.

Example 3

To specify a range of values to include:

```
INCLUDE COND=(10,8,CH,RANGE,(C'00001000',C'00005000'))

INCLUDE COND=((10,8,CH,GE,C'00001000'),&,
              (10,8,CH,LE,C'00005000'))
```

Figure 34. INCLUDE Control Statement with RANGE

This control statement would include all records for an occurrence of the character field in position 10, length 8 bytes, within the constants between '00001000' and '00005000'.

INPFIL Control Statement

The INPFIL control statement describes the input file(s) and details certain processing options. The format of the INPFIL control statement is illustrated below:

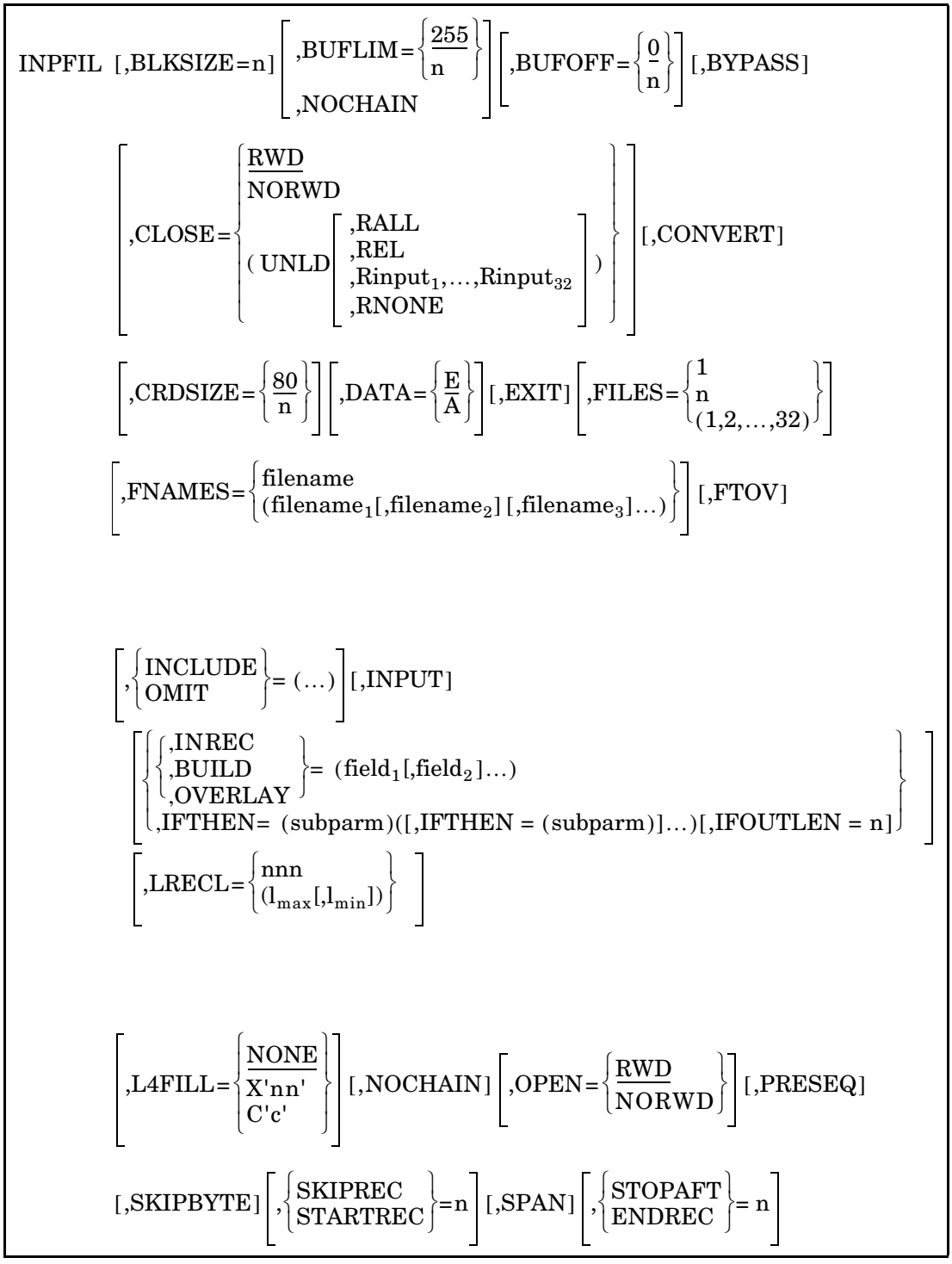


Figure 35. (Page 1 of 2) INPFIL Control Statement Format

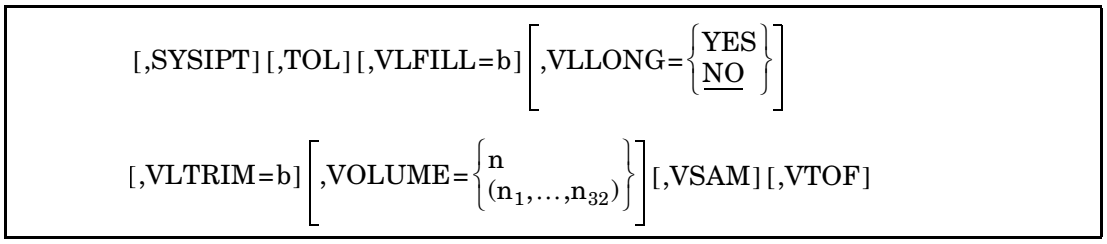


Figure 35. (Page 2 of 2) INPFIL Control Statement Format

For more information on the INPFIL processing sequence, see "INPFIL Processing Sequence" on page 2.21.

BLKSIZE Parameter (Optional)

The BLKSIZE parameter specifies the blocksize of the input records. For multifile input, use the largest blocksize.

If the BLKSIZE parameter is omitted (either by leaving it out of the INPFIL statement or omitting the INPFIL statement itself), SyncSort will assume that the records are unblocked.

The BLKSIZE parameter is not required for VSAM input.

Follow these guidelines in specifying the BLKSIZE value:

fixed-length EBCDIC records — BLKSIZE must be a multiple of the fixed record length (l_1).

variable-length EBCDIC records — BLKSIZE must be greater than or equal to the length of the longest record (l_1, which includes the Record Descriptor Word) plus 4 bytes (for the Block Descriptor Word).

fixed-length ASCII records — BLKSIZE must be a multiple of the fixed record length (l_1) plus the BUFOFF value, if specified.

variable-length ASCII records — BLKSIZE must be greater than or equal to the length of the longest record (l_1, which includes the Record Descriptor Word) plus the BUFOFF value, if specified.

See Table 2 on page 2.23 for the maximum BLKSIZE values allowed for different input devices.

BUFLIM/NOCHAIN Parameter (Optional)

The BUFLIM parameter places an upper limit of' 'n' on the number of input buffers SyncSort can create. Specify n as a decimal number between 1 and 255. The delivered default is 255. For multifile input, this is a global option that can be specified in one INPFIL statement.

The NOCHAIN parameter can be specified only for tape input cases and is equivalent to BUFLIM=2. NOCHAIN prevents chaining from taking place in a double buffer tape case. NOCHAIN is ignored if it is specified for input from other devices.

Note: In the vast majority of cases, SyncSort for z/VSE needs no external assistance in choosing the number of buffers. Therefore, avoid specifying this parameter, as inappropriate use may degrade performance.

BUFOFF Parameter (Optional)

The BUFOFF parameter can be specified for ASCII input to indicate the block prefix size of variable-length ASCII records. The 'n' value can be any number from 0 to 99. The default is 0.

Omit the BUFOFF parameter for fixed-length ASCII records, or specify BUFOFF=0.

Specify the BUFOFF parameter only when TYPE=D is specified on the RECORD control statement or DATA=A is specified on the INPFIL control statement.

When the BUFOFF parameter is specified, the BLKSIZE value must include the BUFOFF value.

BUILD Parameter (Optional)

The BUILD parameter is identical to the INREC parameter of the INPFIL Control Statement. See “INREC Parameter (Optional)” on page 2.72 for a description of this parameter.

BYPASS Parameter (Optional)

The BYPASS parameter can be specified to instruct SyncSort not to terminate abnormally when certain errors are encountered.

The BYPASS parameter functions as follows:

- For non-VSAM files, with fixed-length records or variable-length records, BYPASS instructs SyncSort to skip input data blocks that have been misread by the system.
- For all files with variable-length records, BYPASS instructs SyncSort to skip records that are greater than or less than the lengths specified on the RECORD statement.

If BYPASS is not specified and either of the above situations occurs, the application will terminate abnormally with a detailed error message.

Because this parameter instructs SyncSort to skip erroneous input records, do not specify BYPASS if all the input records must appear in the output file.

If the BYPASS parameter is specified in conjunction with either an E18 or E38 exit, it will be ignored for fixed-length records.

Because the BYPASS parameter reduces performance, specify this option only when necessary.

CLOSE Parameter (Optional)

Specified for tape input files only, the CLOSE parameter indicates what action should be taken at the end of the file(s).

CLOSE=RWD, the default, indicates that all tape input volumes should be rewound at the end of the file(s). CLOSE=NORWD indicates that the input volumes should not be rewound at the end of the file(s).

CLOSE=UNLD indicates that all input volumes should be rewound and unloaded at the end of the file(s). The Release option associated with CLOSE=UNLD is designed to assist installations that lack tape management systems and may degrade performance when specified in conjunction with a tape manager. The RNONE subparameter, the default with CLOSE=UNLD, indicates that the tape drives should not be released until the end of the job. The other subparameters permit the release of all (RALL) or specific assigned tape drives ($Rinput_1$,...,$Rinput_{32}$) after their use for input. When the Release option is specified, the UNLD and Release specifications must be enclosed in parentheses.

CLOSE=(UNLD,REL) indicates that the input tape drives assigned to this INPFIL control statement are to be released after their use.

The following example illustrates the CLOSE parameter:

```
CLOSE=(UNLD,REL)
```

Figure 36. Sample CLOSE Parameter

In this example, the tape drives assigned to the SORTIN logical device are released after the input phase. A SORTIN device is needed for SORTOUT to be released.

CONVERT Parameter

CONVERT is equivalent to VTOF. See “VTOF Parameter (Optional)” on page 2.76.

CRDSIZE Parameter (Optional)

The CRDSIZE parameter specifies the decimal number of bytes of data per physical input card. The delivered default is 80.

When LRECL is a factor of CRDSIZE, "blocked" card input is possible. For example, if CRDSIZE=80, multiple-record card input is possible if LRECL=40, 20, 10, etc. Thus, every data card contains two 40-byte records (or four 20-byte records, etc.). Since short blocks are not permitted, the /* card must appear in columns 1-2 of the last card after full cards of data. A short data card can be padded with identifiable "dummy" records, which then are excluded by the INCLUDE/OMIT control statement.

CRDSIZE removes any possible ambiguity with multi-record card input and multifile, mixed device type input. For example, to sort an 80-column card and 400-byte blocked tape input consisting of 40-byte logical records, use the following control statements:

```
RECORD TYPE=F,LENGTH=40
INPFIL CRDSIZE=80,BLKSIZE=400
```

Figure 37. Sample INPFIL Control Statement Illustrating CRDSIZE Parameter

In this example, the logical records are 40 bytes long. The input consists of both cards and tape. Each card contains two records. The blocksize of the tape input is 400. Note that the CRDSIZE parameter is necessary in this case only if the CRDSIZE default of 80 has been changed.

DATA=E/A Parameter (Optional)

The DATA parameter specifies either EBCDIC or ASCII input. DATA=E specifies EBCDIC input, and DATA=A specifies ASCII tape input. If this parameter is omitted, EBCDIC input is assumed unless TYPE=D is specified on the RECORD control statement to indicate variable-length ASCII input. Mixing ASCII and EBCDIC files on the INPFIL statements is not supported.

If DATA=A is specified, the following restrictions apply:

- Only the AC, AST, or ASL format codes can be specified on the SORT and MERGE control statements.
- The INCLUDE/OMIT, SUM, and ALTSEQ control statements cannot be specified.
- The INREC and OUTREC control statements are permitted, but the following parameters cannot be specified: ZD, PD, BI, or FI formats and Mm, EDIT, SIGNS, LENGTH, and HEX. If the nX parameter is specified, ASCII blanks (X'20') will be inserted instead of EBCDIC blanks (X'40'). Specifying an nC'literal string' on an INREC/OUTREC control statement results in the insertion of the ASCII representation of the character string.

- The following parameters cannot be specified on the INPFIL control statement: CRDSIZE, SPAN, SYSIPT, TOL, and VSAM.

- The following parameters cannot be specified on the OUTFIL control statement: CARDS, DISK, ESDS, HEADER1, HEADER2, INCLUDE, KSDS, LINES, LRECL, NODETAIL, OMIT, PAGES, PRINT, PUNCH, REUSE, RRDS, SECTIONS, SPAN, TOL, TRAILER1, and TRAILER2.

- If the OUTREC parameter is specified on the OUTFIL control statement, the following subparameters cannot be specified: ZD, PD, BI, or FI formats and Mm, EDIT, SIGNS, LENGTH, and HEX. Specifying an nC'literal string' on an INREC/OUTREC control statement results in the insertion of the ASCII representation of the character string.

- The value of the BLKSIZE parameter on the INPFIL and OUTFIL control statements cannot exceed 9,999 bytes.

- TYPE=D must be specified on the RECORD control statement for variable-length records.

- The DELBLANK parameter cannot be specified on the RECORD control statement.

- The KEYLEN and ADDROUT parameters cannot be specified on the OPTION control statement. The CMP parameter can be specified on the OPTION statement, but will be ignored.

- The date data formats Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z are changed into 4-digit character (EBCDIC) format by OUTREC data conversion processing. If DATA=A has been specified on the INPFIL statement, the above date data formats will be converted into 4-digit ASCII character formats.

When TYPE=D is specified on the RECORD control statement, the above restrictions apply even if the DATA=A parameter is not specified on the INPFIL control statement.

ENDREC Parameter (Optional)

The ENDREC parameter specifies the last record from the input file.

The record count is done before INPFIL INCLUDE/OMIT processing, if specified, takes place. (For example, if the first 100 records are to be deleted by INCLUDE/OMIT, the next record is still counted as record number 101. If ENDREC=90, then **no** records will be passed to the sort/merge.) For ENDREC=n, the n value can be 1 through 2,147,483,647.

The ENDREC parameter is incompatible with the STOPAFT parameter on the same INPFIL statement.

INPFIL

EXIT Parameter (Optional)

The EXIT parameter indicates that an E15 or E32 exit will provide all the input to the sort/merge. Write the E15 or E32 exit routine so that it reads the input data and passes one record at a time to the sort/merge. Include label processing, volume positioning, and file opening and closing information. Also, specify exit E15 or exit E32 in the MODS control statement.

If the EXIT parameter is specified, all other INPFIL parameters are checked for syntax but ignored.

FILES Parameter (Optional)

The FILES parameter connects the INPFIL statement with one or more input files. The default is FILES=1.

By using FILES=(1,2,...,32), data sets that have all their parameters in common may be specified by one INPFIL. If any one parameter differs, a separate INPFIL must be used.

Note: FNAMES and FILES are mutually exclusive parameters. Specifying both parameters on the same INPFIL statement will cause a syntax error. If neither parameter is used and only one INPFIL statement is specified, then that INPFIL statement applies to all input files.

FNAMES Parameter (Optional)

The FNAMES parameter connects the INPFIL statement with one or more input files. The name(s) used to specify the input file(s) can be up to seven characters long, and a corresponding DLBL/TLBL statement must be present. The filenames specified with the FNAMES parameter must be either the default filenames (SORTIN1, SORTIN2, ...) or a filename specified using the FILNM parameter of the OPTION statement.

When two or more input files have identical parameters on the INPFIL statement, write the specifications only once as illustrated below:

```
FNAMES=(filename1,filename2,...,filename32)
```

Figure 38. Sample FNAMES Parameter

Note: FNAMES and FILES are mutually exclusive parameters. Specifying both parameters on the same INPFIL statement will cause a syntax error. If neither parameter is used and only one INPFIL statement is specified, then that INPFIL statement applies to all input files.

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

Note: FTOV cannot be used with CONVERT or VTOF. VLTRIM can be used with the FTOV parameter to delete pad bytes at the end of a record.

The LENGTH parameter on the RECORD control statement specifies the variable-length output record to be produced by FTOV. l_1 specifies the maximum variable-length output record to be produced. l_4 specifies the minimum variable-length output record to be produced.

The following control parameters may not be specified when INPFIL FTOV is used:

- INPFIL EXIT
- RECORD TYPE F,D
- MERGE FIELDS=COMPARE
- INPFIL VTOF, CONVERT

FTOV, no INREC Parameter

When FTOV is used without the INREC parameter, the variable-length record is created from the fixed-length record.

FTOV, with INREC Parameter

When FTOV is used with the INREC parameter, the variable-length record is created from the specified fields of the fixed-length record. The maximum record length is the maximum of the reformatted input records plus 4.

The TYPE parameter on the RECORD control statement must have a variable-length record format (for example, VB or VBS).

The LENGTH parameter on the RECORD control statement must be given an LRECL that can contain the largest and smallest variable-length records to be produced by INPFIL.

IFOUTLEN Parameter (Optional)

The IFOUTLEN parameter overrides the maximum record length, which is automatically set by the IFTHEN parameter, and changes it to a specified value. The IFOUTLEN parameter may only be used in conjunction with the IFTHEN parameter.

INPFIL

The IFOUTLEN parameter automatically makes the following changes to the record to match the new length. A record longer than n is truncated to n. A fixed-length record shorter than n is padded with blanks to reach a length of n. The length n must not be larger than the length l_1 specified in the LENGTH parameter on the RECORD control statement.

IFTHEN Parameter (Optional)

The IFTHEN parameter uses conditional logic which enables you to reformat records based on specified criteria. Multiple IFTHEN parameters may be specified within the same control statement and are processed sequentially. See "IFTHEN Parameter (Optional)" on page 2.205 for a complete description of this parameter.

INCLUDE/OMIT Parameter (Optional)

The INCLUDE/OMIT parameter selects records from an input file based on comparisons of the contents of one or more fields within the record. See "INCLUDE/OMIT Control Statement Format" on page 2.39.

INPUT Parameter (Optional)

The INPUT parameter is ignored by SyncSort for z/VSE.

INREC Parameter (Optional)

The INREC parameter reformats the input records. See "INREC Control Statement Format" on page 2.83 for a complete description of this parameter's functions.

Note: The LENGTH parameter on the RECORD statement functions differently for fixed-length records than it does for variable-length records. For fixed-length records, LENGTH indicates the record length of the output records produced by INPFIL.

The length of the reformatted input record should match the length specified in the LENGTH parameter on the RECORD statement. If the reformatted input record is less than the length on the RECORD statement, SyncSort automatically pads the output record with blanks on the right to achieve the desired length.

For variable-length records, LENGTH indicates the maximum and minimum record length of the logical records produced by INPFIL INREC. l_1 specifies the maximum variable-length output record to be produced. l_4 specifies the minimum variable-length output record to be produced. This length should take into account any modifications made by an INREC, or FTOV. See "LRECL Parameter (Optional)" section below for a complete description of how to specify an input record length for each file.

LRECL Parameter (Optional)

The LRECL parameter specifies an input record length ***before*** INPFIL INREC processing, which may differ from the length specified in the LENGTH parameter on the RECORD statement (see Figure 3 on page 2.18). It is, therefore, possible to have one record length for the sort/merge and another record length for the input record. For variable-length records, l_{max} is the maximum input record length, and l_{min} is the minimum input record length.

Use the INREC control statement to reformat the input records.

When multiple INPFIL statements are used, LRECL parameters may be used on some or all of the INPFIL statements. When LRECL is omitted, the record length values for that file are those specified by the LENGTH parameter of the RECORD statement.

When INREC, FTOV, or VTOF is specified on ***any*** INPFIL statement, the LRECL parameter is ***required*** on ***all*** INPFIL statements.

L4FILL Parameter (Optional)

INPFIL L4FILL=X'hh' (or C'c') causes variable-length input records shorter than l_4 (before INREC processing) to be padded with hex byte “hh” (or character “c”) to a length of l_4. IGNRL4 is ignored for this file.

INPFIL L4FILL pads the records before any reformatting of records is done by the INREC parameter of the INPFIL control statement, or by the INREC control statement. When INPFIL INREC reformats the input records, the records are padded up to the minimum record length specified by the LRECL l_{min} parameter on INPFIL (the length before INPFIL INREC reformatting). Otherwise, the l_4 value specified on the RECORD control statement (or calculated by SyncSort), which is the length before any reformatting by the INREC control statement, is used. (See Figure 4 on page 2.19 and Figure 5 on page 2.19.)

When L4FILL=NONE is specified, no padding is done. This overrides any L4FILL installation default set by SYNCMAC.

When L4FILL is not specified on an INPFIL statement, padding will be done only if

- L4FILL has been specified as a default by SYNCMAC; and

- IGNRL4 has not been specified on the OPTION control statement.

When INPFIL L4FILL is used without FILES=f, the L4FILL specification applies to all input files.

L4FILL is not used to pad short variable-length input records that are to be converted to fixed-length records by VTOF or CONVERT; in these cases, specify the VLFILL parameter.

OPEN Parameter (Optional)

Specified for tape input files only, the OPEN parameter indicates what action should be taken when the file is opened.

OPEN=RWD, the default, indicates that the first volume of each tape input file should be rewound when the file is opened. OPEN=NORWD indicates that the first volume of each input file should not be rewound when the file is opened. This parameter is ignored if the input files are on direct access devices.

OVERLAY Parameter (Optional)

The OVERLAY parameter enables you to change particular columns within a record, and add fields to the end of a record, without rebuilding the entire record. When using the OVERLAY parameter you only need to specify the columns you want to change; the rest of the input record remains unchanged. See “OVERLAY Parameter (Optional)” on page 2.209 for a complete description of this parameter.

PRESEQ Parameter (Optional)

The PRESEQ parameter is ignored by SyncSort for z/VSE.

SKIPBYTE Parameter (Optional)

The SKIPBYTE parameter is ignored by SyncSort for z/VSE.

SKIPREC Parameter (Optional)

The SKIPREC parameter instructs SyncSort to skip a decimal number of 'n' records per input file before the file is sorted, merged, or copied. The input bypasses the skipped records before INCLUDE/OMIT and INREC processing, if specified, takes place. For SKIPREC=n, the n value can be 0 through 2,147,483,647.

The SKIPREC parameter is incompatible with the STARTREC parameter on the same INPFIL statement.

SPAN Parameter (Optional)

The SPAN parameter instructs SyncSort to examine the input file(s) for spanned records.

If TYPE=VS or TYPE=VBS is specified on the RECORD control statement, the SPAN parameter need not be specified.

STARTREC Parameter (Optional)

The STARTREC parameter specifies the first record that will be processed from the input file. The 'n'th record will be the first record eligible for the sort/merge, while the first n-1 records are skipped. The record count is done before INPFIL INCLUDE/OMIT processing, if specified, takes place. For STARTREC=n, the n value can be 1 through 2,147,483,647.

The STARTREC parameter is incompatible with the SKIPREC parameter on the same INPFIL statement.

STOPAFT Parameter (Optional)

The STOPAFT parameter specifies the number of records in a file for output to the sort/merge. These will be the first 'n' records after INCLUDE/OMIT, SKIPREC, and STARTREC processing. For STOPAFT=n, the n value can be 0 through 2,147,483,647.

The STOPAFT parameter is incompatible with the ENDREC parameter on the same INPFIL statement.

SYSIPT Parameter (Optional)

The SYSIPT parameter specifies that the input should be read from a card reader. Card input can be read with or without specific file assignments. Use the SYSIPT parameter instead of an E15 exit program for increased efficiency.

Note: SYSIPT must be assigned either to an IBM-supported card reader or to SYSRDR simulated input (diskette). The SYSIPT parameter cannot be used for card image input on tape or disk.

TOL Parameter (Optional)

The TOL parameter indicates that a warning code is to be accepted when a VSAM input file is opened. If the TOL parameter is not specified, the warning return code is recognized as an error, an error message is generated, and SyncSort for z/VSE terminates.

VLFILL Parameter (Optional)

The VLFILL parameter is used in conjunction with VTOF to specify a fill byte to be used for any short record. By default, spaces will be used.

VLLONG Parameter (Optional)

The VLLONG parameter specifies that SyncSort is to truncate long variable-length input records. A long record is one whose length is greater than the LENGTH=parameter specified in the RECORD control statement. VLLONG=YES indicates that SyncSort truncates long variable-length input records. VLLONG=NO indicates that SyncSort will terminate if a long variable-length input record is encountered. This is the default.

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 Record Descriptor Word (RDW), or a data byte less than or equal to the minimum record length.

For the VLTRIM parameter, specify a byte *b* 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 the FTOV parameter is not used.

VOLUME Parameter (Optional)

Specified for unlabeled or nonstandard labeled tape input files only, the VOLUME parameter indicates the number of volumes for each input file. The default is one volume for each input file. This parameter is ignored for disk input files.

For multifile input, the format of the VOLUME parameter is VOLUME=(n_1,...,n_{32}), where n represents the number of volumes in each file. Specify the n values in the same order as the input files are specified in the SORTIN portion of the job control stream. If the FILES or FNAMES parameter is specified, only the VOLUME=n parameter is supported.

SyncSort's installation procedure allows the user to choose IBM SM-483 compatibility for this parameter. (See the VOLUME parameter of SYNCMAC in the *SyncSort for z/VSE Installation Guide* for instructions.) With SM-483 compatibility for standard labeled files, SyncSort will process whichever is less: the user's actual number of volumes or the number specified in the VOLUME parameter. Without this change, processing of standard labeled files is compatible with IBM's 5743 and 5746 programs; i.e., the VOLUME parameter is ignored.

VSAM Parameter (Optional)

The VSAM parameter specifies VSAM access for VSAM managed SAM input files and is not required with VSAM input. If this parameter is not specified, SyncSort assumes SAM access for VSAM managed SAM.

Note: For improved performance when all input files are VSAM, specify the VSAM parameter.

VTOF Parameter (Optional)

The VTOF parameter converts variable-length records to fixed-length records. VTOF can be used either with or without the INREC parameter. When VTOF is used with the INREC parameter, the fixed-length record is created from the specified fields of the variable-length

record. When VTOF is not used with the INREC parameter, the fixed-length record is created from the whole variable-length record.

The length of the output record of INPFIL is specified in the LENGTH parameter on the RECORD control statement. If the output record of INPFIL is less than the length on the RECORD control statement, VTOF will automatically pad the output record with the fill character (VLFILL=*b*) on the right to achieve the desired length. The default pad character is blank (VLFILL=X'40').

If the output record is longer than the length on the RECORD statement, the result depends on the value of the VLLONG parameter:

- The sort terminates with an error message and gives a return code of 16 if the VLLONG=NO option is specified.
- VTOF truncates the long output records to achieve the desired length if the VLLONG=YES option is specified. This option should not be used unless you want the long variable-length output records to be truncated. Inappropriate use of VLLONG can result in unwanted loss of data.

The following control parameters may not be specified when VTOF is used:

- INPFIL EXIT
- RECORD TYPE=VS, VBS, D, V
- MERGE FIELDS=COMPARE
- INPFIL FTOV

Sample INPFIL Statements

```
INPFIL BLKSIZE=500,VOLUME=(3,10,1),CLOSE=UNLD
```

Figure 39. Sample INPFIL Statement

In Figure 39, the INPFIL statement specifies the following:

- The largest blocksize of the input files is 500.
- The first input file has three volumes, the second input file has ten volumes, and the third input file has one volume.
- All tape input volumes will be rewound and unloaded at the end of the file.
- Because the OPEN parameter is not specified, the first volume of each input file will be rewound when the file is opened. (This is the default).

```
INPFIL EXIT
```

Figure 40. Sample INPFIL Statement

In Figure 40, the INPFIL statement specifies that a user exit program will provide all the input records.

```
INPFIL INREC=(1,50,200,10),LRECL=250,FTOV
```

Figure 41. Sample INPFIL Statement

In Figure 41, the INPFIL statement reformats 250-byte fixed-length input records to 64-byte (including 4-byte RDW) variable-length records before the sort/merge.

Sample Multiple INPFIL Statements

The following examples show the use of the INPFIL control statement for multiple input files.

```
RECORD TYPE=V,LENGTH=(204,,,50)
INPFIL   FILES=1,FTOV,LRECL=80,VLTRIM=C'*'
INPFIL   FILES=2,FTOV,LRECL=100,INREC=(1,50,80,10),...
INPFIL   FILES=3,FTOV,LRECL=200,BLKSIZE=2000,VLTRIM=C''
```

Figure 42. Sample Multiple INPFIL Statements

In Figure 42, the control statements specify both the maximum (l_1) and minimum (l_4) variable-length output record. The largest variable-length output record to be produced for the three input files is 204 and the minimum record length is 50.

```
SORT FIELDS=(28,7,CH,A)
RECORD TYPE=V,LENGTH=204
INPFIL   FILES=1,FTOV,LRECL=80,VLTRIM=C'*'
INPFIL   FILES=2,FTOV,LRECL=100,INREC=(1,50,80,10),...
INPFIL   FILES=3,FTOV,LRECL=200,BLKSIZE=2000,VLTRIM=C''
```

Figure 43. Sample Multiple INPFIL Statements

In Figure 43, the control statements specify ONLY the maximum (l_1) variable-length output record. The largest variable-length output record to be produced for the three input files is 204 and the minimum record length is 34 calculated by the sort.

```
RECORD TYPE=V,LENGTH=(1200,,,80)
INPFIL FILES=1,LRECL=(160,80),STOPAFT=100,
  INREC=(1,26,5,7,34),BLKSIZE=1604
INPFIL FILES=2,STOPAFT=150,LRECL=(1200,80),
  INREC=(1,26,9,7,34),BLKSIZE=12004
```

Figure 44. Sample Multiple INPFIL Statements

In Figure 44, the INPFIL FILES=1 statement specifies the following:

- The largest blocksize of the input file is 1604.
- The maximum record length of the logical records is 160, and 80 bytes is the minimum.
- The number of records to read from this file is 100.
- The input record will be reformatted based on the specified fields.
- The output maximum record length of the logical records is 1200, and 80 is the minimum.

The INPFIL FILES=2 statement specifies the following:

- The largest blocksize of the input file is 12004.
- The maximum record length of the logical records is 1200, and 80 is the minimum.
- The number of records to read from this file is 150.
- The input record will be reformatted based on the specified fields.
- The output maximum record length of the logical records is 1200, and 80 is the minimum.

```
RECORD TYPE=F,LENGTH=80
INPFIL BLKSIZE=800,FILES=1,STOPAFT=150,
  LRECL=80,INREC=(1,70)
INPFIL BLKSIZE=9387,FILES=2,
  LRECL=63,INREC=(1,60),SKIPREC=100
```

Figure 45. Sample Multiple INPFIL Statements

In Figure 45, the INPFIL FILES=1 statement specifies the following:

- The largest blocksize of the input file is 800.
- The record length of the logical records is 80.

- The number of records to read from this file is 150.
- The first 70 bytes contain original data, and the last 10 bytes are blanks.
- The output record length is 80.

The INPFIL FILES=2 statement specifies the following:

- The largest blocksize of the input file is 9387.
- The record length of the logical records is 63.
- The number of records to be skipped is 100.
- The first 60 bytes contain original data, and the last 20 bytes are blanks.
- The output record length is 80.

```
RECORD TYPE=F,LENGTH=80
INPFIL FILES=1,BLKSIZE=800,STOPAFT=200,SKIPREC=100
INPFIL FILES=2,BLKSIZE=800,STOPAFT=100
INPFIL FILES=3,BLKSIZE=800
OUTFIL BLKSIZE=800
OPTION PRINT=ALL,ROUTE=LST,STOPAFT=500
```

Figure 46. Sample Multiple INPFIL Statements

In Figure 46, the INPFIL FILES=1 statement specifies the following:

- The largest blocksize of the input file is 800.
- The record length of the logical records is 80.
- The number of records to be skipped is 100.
- The number of records to read from this file is 200.
- The output record length is 80.

The INPFIL FILES=2 statement specifies the following:

- The largest blocksize of the input file is 800.
- The record length of the logical records is 80.
- The number of records to read from this file is 100.
- The output record length is 80.

The INPFIL FILES=3 statement specifies the following:

- The largest blocksize of the input file is 800.
- The record length of the logical records is 80.
- The number of records to read from this file is 200.
- The output record length is 80.

The OPTION STOPAFT parameter limits the number of records for the sort as follows:

- 200 records of SORTIN1 are accepted for processing.
- 100 records of SORTIN2 are accepted for processing.
- 200 records of SORTIN3 are accepted for processing (calculated 500-200-100=200).

```
RECORD TYPE=V,LENGTH=(204,,,50)
INPFIL FILES=1,FTOV,LRECL=80,VLTRIM=C'*'
INPFIL FILES=2,FTOV,LRECL=100,INREC=(1,50,80,10)
INPFIL FILES=3,FTOV,LRECL=200,BLKSIZE=2000,VLTRIM=C''
```

Figure 47. Sample Multiple INPFIL Statements

In Figure 47, the control statements specify both the maximum (l_1) and minimum (l_4) variable-length output records. The largest variable-length output record to be produced for the three input files is 204 and the minimum record length is 50.

```
SORT FIELDS=(28,7,CH,A)
RECORD TYPE=V,LENGTH=204
INPFIL FILES=1,FTOV,LRECL=80,VLTRIM=C'*'
INPFIL FILES=2,FTOV,LRECL=100,INREC=(1,50,80,10)
INPFIL FILES=3,FTOV,LRECL=200,BLKSIZE=2000,VLTRIM=C''
```

Figure 48. Sample Multiple INPFIL Statements

In Figure 48, the control statements specify ONLY the maximum (l_1) variable-length output record. The largest variable-length output record to be produced for the three input files is 204; the minimum record length of 34 is calculated by the sort.

```
SORT FIELDS=(30,7,CH,A)
RECORD TYPE=V,LENGTH=(83,,,53)
INPFIL FILES=(1,3)BLKSIZE=9387
INPFIL FILES=2,BLKSIZE=9387,LRECL=(63,58),INREC=(1,34,20X,40)
```

Figure 49. Sample Multiple INPFIL Statements

Figure 49 shows that the largest variable-length output record to be produced for the three input files is 83, and the minimum record length is 53.

```
SORT FIELDS=(30,7,CH,A)
RECORD TYPE=V,LENGTH=83
INPFIL FILES=(1,3)BLKSIZE=9387,LRECL=(63,58)
INPFIL FILES=2,BLKSIZE=9387,LRECL=(63,58),INREC=(1,34,20X,40)
```

Figure 50. Sample Multiple INPFIL Statements

Figure 50 shows that the largest variable-length output record to be produced for the three input files is 83, and the minimum record length is 58.

```
SORT FIELDS=(30,7,CH,A)
RECORD TYPE=V,LENGTH=(63,,,53)
INPFIL FILES=1,BLKSIZE=9387
INPFIL FILES=2,BLKSIZE=9387,OMIT=(20,3,CH,EQ,C'BAD')
INPFIL FILES=3,BLKSIZE=9387,STOPAFT=100
```

Figure 51. Sample Multiple INPFIL Statements

Figure 51 shows that the largest variable-length output record to be produced for the three input files is 83; the minimum record length of 37 is calculated by the sort.

INREC Control Statement

The INREC control statement reformats the input records. Use the INREC statement to add, delete, or reformat fields *before* the records are sorted or merged. Use the OUTREC statement or the OUTREC parameter of the OUTFIL 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 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 statement, such as inserting character strings or changing the data format of a numeric field, can also be performed by the INREC statement. (See "OUTREC Control Statement Format" on page 2.175 for an explanation of these functions.) For example, you can use the INREC 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 statement rather than the INREC statement to expand the record because OUTREC processing takes place *after* records are sorted or merged.

If you use the INREC 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 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 statement is illustrated below:

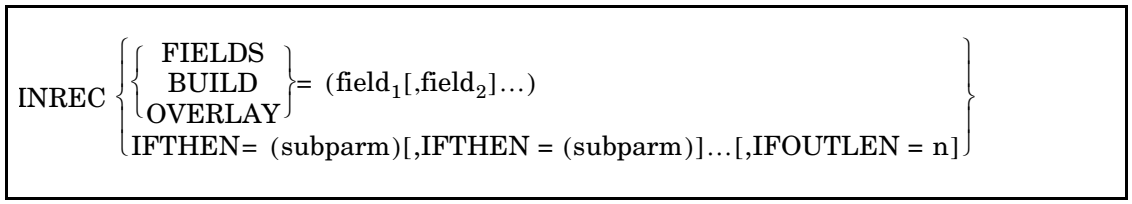


Figure 52. INREC Control Statement Format

FIELDS/BUILD Parameter

The FIELDS parameter specifies the data fields to be included in the application. BUILD is an alias for FIELDS. See "OUTREC Control Statement Format" on page 2.175 for a complete description of these parameters.

IFOUTLEN Parameter (Optional)

The IFOUTLEN parameter overrides the maximum record length, which is automatically set by the IFTHEN parameter, and changes it to a specified value. The IFOUTLEN parameter may only be used in conjunction with the IFTHEN parameter.

The IFOUTLEN parameter automatically makes the following changes to the record to match the new length. A record longer than n is truncated to n. A fixed-length record shorter than n is padded with blanks to reach a length of n.

IFTHEN Parameter (Optional)

The IFTHEN parameter uses conditional logic which enables you to reformat records based on specified criteria. Multiple IFTHEN parameters may be specified within the same control statement and are processed sequentially. See "IFTHEN Parameter (Optional)" on page 2.205 for a complete description of this parameter.

OVERLAY Parameter (Optional)

The OVERLAY parameter enables you to change particular columns within a record, and add fields to the end of a record, without rebuilding the entire record. When using the OVERLAY parameter you only need to specify the columns you want to change; the rest of the input record remains unchanged. See "OVERLAY Parameter (Optional)" on page 2.209 for a complete description of this parameter.

Sample INREC Control Statement

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

Figure 53. Sample INREC Control Statement

This INREC 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 4. How to Use SyncSort Data Utility Features”.

JOIN Control Statement

The JOIN control statement specifies the disposition of paired and unpaired records in a join. (See Figure 1 on page 1.5 for an illustration of join processing.)

When you do not provide a JOIN control statement in an application that has JOINKEYS control statements, SyncSort produces an output from the join operation that includes all paired records (an "inner join"). All unpaired records from both SORTIN1 and SORTIN2 are discarded. By providing a JOIN control statement, you can specify that unpaired records are to be included in the join output (an "outer join"). Parameters of the JOIN control statement provide options as to which of the unpaired records are to be retained for output.

See "JOINKEYS Control Statement" on page 2.88 and "REFORMAT Control Statement" on page 2.224 for additional information.

JOIN Control Statement Format

The format of the JOIN control statement is illustrated below:

```
JOIN UNPAIRED[,F1][,F2][,ONLY]
```

Figure 54. JOIN Control Statement Format

Retaining Unpaired Records

When joining files, a record from one file may or may not have a match in the other file. A match occurs when the contents of the join keys in the record from the first file equal the contents of the join keys in the record from the second file.

By specifying the JOIN control statement you can retain unpaired records from one or both files.

To retain unpaired records from SORTIN1 (a "left outer join") in addition to all joined records, specify:

```
JOIN UNPAIRED,F1
```

Figure 55. Retaining Unpaired Records from SORTIN1

To retain unpaired records from SORTIN2 (a "right outer join") in addition to all joined records, specify:

```
JOIN UNPAIRED,F2
```

Figure 56. Retaining Unpaired Records from SORTIN2

To retain unpaired records from both SORTIN1 and SORTIN2 (a "full outer join") in addition to all joined records, specify either:

```
JOIN UNPAIRED,F1,F2
```

Figure 57. Retaining Unpaired Records from both SORTIN1 and SORTIN2

or simply:

```
JOIN UNPAIRED
```

Figure 58. Retaining Unpaired Records from both SORTIN1 and SORTIN2

Discarding Paired Records

You have the option of discarding the paired records from a join and keeping only the unpaired ones. To do this, specify:

```
JOIN UNPAIRED,ONLY
```

Figure 59. Discarding Paired Records

If you want to keep only the unpaired records from one SORTIN1 or SORTIN2, add either the F1 or the F2 parameter.

Note: See "REFORMAT Control Statement" on page 2.224 for a discussion on what will appear in the record created by join processing when source fields from either SORTIN1 or SORTIN2 are not available due to a join unpaired operation.

JOINKEYS Control Statement

Use the JOINKEYS statement to enable join feature processing and to identify the fields used to select records for join processing. (See Figure 1 on page 1.5 for an illustration of join processing.)

The join feature joins records from two input files that are specified on the SORTIN1 and SORTIN2 label statements. By default, when the JOINKEYS fields from *m* records in SORTIN1 match the JOINKEYS fields from *n* records in SORTIN2, all combinations of the records are joined using the REFORMAT control statement, producing *m***n* records as input to subsequent SyncSort processing. (This is called an "inner join.")

See "REFORMAT Control Statement" on page 2.224 for a description of how a record is constructed from two records that have been selected as a match.

If the optional JOIN UNPAIRED control statement is specified, the unmatched records from the SORTIN1 and/or SORTIN2 files will also be REFORMATted and included in the input to SyncSort without being joined. (Including the unmatched records from SORTIN1 is called a "left outer join," and including all unmatched records is called a "full outer join.") Optionally, only these unmatched records will become input to SyncSort. See "JOIN Control Statement" on page 2.86 and "REFORMAT Control Statement" on page 2.224 for further details on their specifications.

The input files do not need to be presorted or have the same record type.

Two JOINKEYS control statements are required - one for each of the two files used in the join. Do not specify an INPFIL control statement when JOINKEYS control statements are used.

When JOINKEYS control statements are used, the RECORD control statement is ignored and can be omitted. The actual record type and length will be determined by the REFORMAT statement or the input files.

The JOINKEYS control statement cannot be used with any exits (except for E35).

JOINKEYS Control Statement Format

The format of the JOINKEYS control statement is illustrated below:

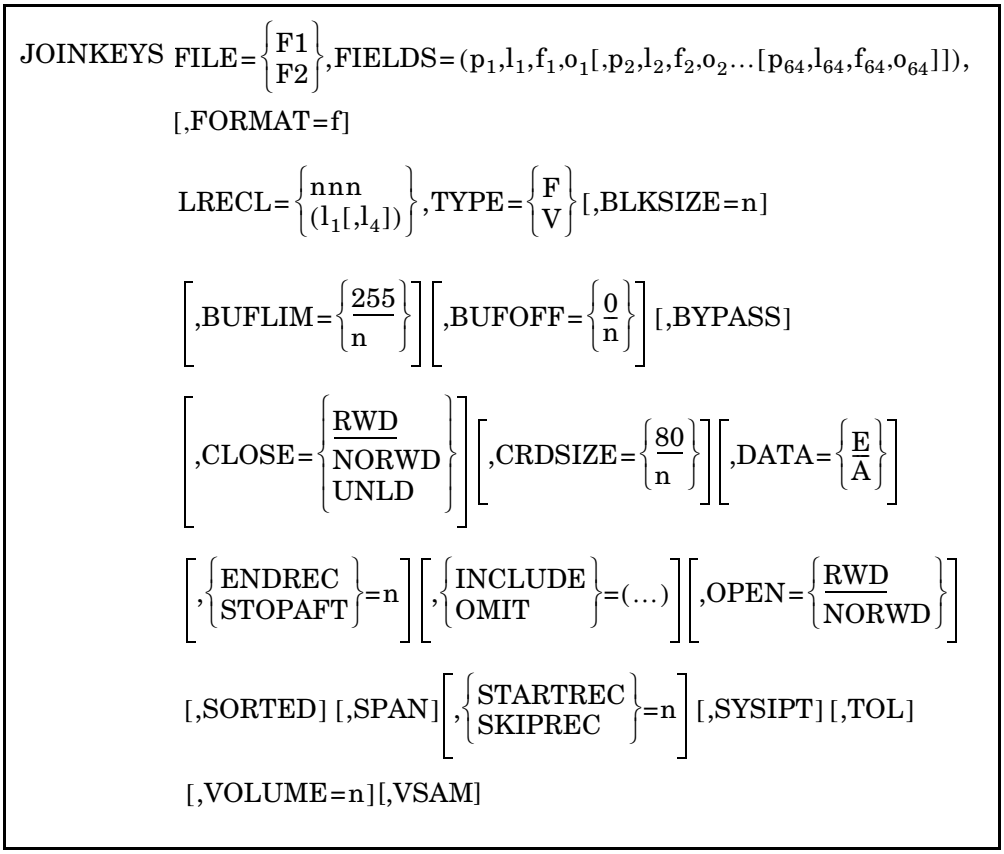


Figure 60. JOINKEYS Control Statement Format

FILE Parameter (Required)

The FILE parameter connects the JOINKEYS control statement with the input file to be read. The specification of F1 connects the JOINKEYS control statement with the SORTIN1 label statement. The specification of the F2 connects the JOINKEYS control statement with the SORTIN2 label statement.

FIELDS Parameter (Required)

The FIELDS parameter is required. It describes the fields to be used to match records from the two files, SORTIN1 and SORTIN2.

The number of JOINKEYS fields must be the same for both files, although their starting positions need not be the same.

These fields can be in any of CH, BI, FI, PD, or ZD formats. If not explicitly specified, the default format is binary (BI).

The join files do not need to be presorted on the fields specified on the JOINKEYS control statement. By default, SyncSort will sort the records to the proper sequence before performing the join operation. If one or both of the files is already in the JOINKEYS FIELDS sequence, the SORTED parameter (see below) of the JOINKEYS control statement can be specified. If the SORTED parameter is used, the performance of the application may be improved.

The maximum number of JOINKEYS fields is 64.

Each JOINKEYS field may be anywhere within the record, the maximum length of a field is 256 to 4080 bytes, and the sum of all fields on a JOINKEYS control statement cannot exceed 4080 bytes.

For variable-length records, all JOINKEYS fields must reside within the minimum record length.

Each field specified in the FIELDS parameter is identified by a position (**p**), length (**l**), optionally, format (**f**), and order (**o**).

p The position value indicates the first byte of the field relative to the beginning of the input record.

l The length value indicates the length of the control field.

FORMAT	PERMISSIBLE LENGTH	PADDING FOR SHORTER FIELD	
BI	1 to 4080 bytes	binary zeroes	on the right
CH	1 to 4080 bytes	blanks	on the right
FI	1 to 256 bytes	sign extended	on the left
PD	1 to 256 bytes	binary zeroes	on the left
ZD	1 to 256 bytes	binary zeroes	on the left

Table 14. Permissible JOINKEYS field lengths

f (optional) The format value indicates the format type of the control field. If omitted, the value specified in the FORMAT parameter is used. If the FORMAT parameter is also omitted then BI is assumed.

Corresponding JOINKEYS fields from the two joining files must be of comparable formats as indicated in the following table.

	BI	CH	FI	PD	ZD
BI	X	X			
CH	X	X			
FI			X		
PD				X	X
ZD				X	X

Table 15. Comparable formats for JOINKEYS fields

o The order value indicates the collating sequence of the field:

A=Ascending order
D=Descending order

FORMAT Parameter (Optional)

The optional FORMAT parameter specifies the default format for fields specified in the FIELDS parameter. If omitted, BI will be the default format for fields specified in the FIELDS parameter. Note that this parameter has no effect on the optional INCLUDE or OMIT parameter where field format must be specified explicitly for each field.

LRECL Parameter (Required)

The LRECL parameter is used to indicate the record length. The l_1 value indicates the maximum length record. For variable-length records, l_4 is optional and indicates the shortest length record. See "LENGTH Parameter (Required)" on page 2.219 for a detailed description of permissible record lengths.

TYPE Parameter (Required)

The TYPE parameter indicates the record format. TYPE=F indicates fixed-length records; TYPE=V indicates variable-length records.

BLKSIZE Parameter (Optional)

The BLKSIZE parameter is used to indicate the block size. See "BLKSIZE Parameter (Optional)" on page 2.65 for a detailed description of this parameter.

BUFLIM Parameter (Optional)

The BUFLIM parameter places an upper limit of 'n' on the number of input buffers SyncSort can create. The delivered default is 255. See "BUFLIM/NOCHAIN Parameter (Optional)" on page 2.66 for a detailed description of this parameter.

JOINKEYS

BUFOFF Parameter (Optional)

The BUFOFF parameter can be specified for ASCII input to indicate the block prefix size of variable-length ASCII records. The delivered default is 0. See "BUFOFF Parameter (Optional)" on page 2.66 for a detailed description of this parameter.

BYPASS Parameter (Optional)

The BYPASS parameter can be specified for all non-VSAM files to instruct SyncSort not to terminate abnormally when certain errors are encountered. See "BYPASS Parameter (Optional)" on page 2.66 for a detailed description of this parameter.

CLOSE Parameter (Optional)

Specified for tape input files only, the CLOSE parameter indicates what action should be taken at the end of file. The delivered default is RWD. See "CLOSE Parameter (Optional)" on page 2.67 for a detailed description of this parameter.

Note: The (REL, RNONE, RALL, and Rnnn) options on the CLOSE parameter are not supported on the JOINKEYS control statement.

CRDSIZE Parameter (Optional)

The CRDSIZE parameter specifies the decimal number of bytes of data per physical input card. The delivered default is 80. See "CRDSIZE Parameter (Optional)" on page 2.68 for a detailed description of this parameter.

DATA Parameter (Optional)

The DATA parameter specifies either EBCDIC or ASCII input. The delivered default is EBCDIC. See "DATA=E/A Parameter (Optional)" on page 2.68 for a detailed description of this parameter.

ENDREC Parameter (Optional)

The ENDREC parameter specifies the last record that will be processed from the input file. See "ENDREC Parameter (Optional)" on page 2.69 for a detailed description of this parameter.

INCLUDE/OMIT Parameter (Optional)

The INCLUDE/OMIT parameter selects from the input file based on comparisons of the contents of one or more fields within the record. See "INCLUDE/OMIT Control Statement" on page 2.39 for a detailed description of this parameter.

OPEN Parameter (Optional)

Specified for tape input files only, the OPEN parameter indicates what action should be taken when the file is opened. The delivered default is RWD. See "OPEN Parameter (Optional)" on page 2.74 for a detailed description of this parameter.

SKIPREC Parameter (Optional)

The SKIPREC parameter instructs SyncSort to skip a decimal number of records per input file. See "SKIPREC Parameter (Optional)" on page 2.74 for a detailed description of this parameter.

SORTED Parameter (Optional)

By default, SyncSort will presume that the records in the file are not presequenced in the JOINKEYS FIELDS specified. If both files are already collated in the proper sequence (sorted with the first field being the major field and the subsequent fields being progressively less significant fields), the SORTED parameter can be specified to improve the application's performance.

If sorting is skipped for a pre-sorted file, SyncSort will sequence check that file according to its JOINKEYS fields. If the sequence check fails, SyncSort will issue a critical error message containing the file name.

SPAN Parameter (Optional)

The SPAN parameter instructs SyncSort to examine the input file for spanned records. See "SPAN Parameter (Optional)" on page 2.74 for a detailed description of this parameter.

STARTREC Parameter (Optional)

The STARTREC parameter specifies the first record that will be processed from the input file. See "STARTREC Parameter (Optional)" on page 2.75 for a detailed description of this parameter.

STOPAFT Parameter (Optional)

The STOPAFT parameter specifies the number of records in a file for output. See "STOPAFT Parameter (Optional)" on page 2.75 for a detailed description of this parameter.

SYSIPT Parameter (Optional)

The SYSIPT parameter specifies that the input should be read from a card reader. See "SYSIPT Parameter (Optional)" on page 2.75 for a detailed description of this parameter.

TOL Parameter (Optional)

The TOL parameter specifies that a warning code is to be tolerated for a VSAM input file. See "TOL Parameter (Optional)" on page 2.75 for a detailed description of this parameter.

VOLUME Parameter (Optional)

The VOLUME parameter indicates the number of unlabeled or nonstandard tape input files to be read. See "VOLUME Parameter (Optional)" on page 2.76 for a detailed description of this parameter.

VSAM Parameter (Optional)

The VSAM parameter indicates VSAM accessed input files. See "VSAM Parameter (Optional)" on page 2.76 for a detailed description of this parameter.

Sample JOINKEYS Control Statements

```
JOINKEYS FILE=F1,FIELDS=(15,10,A),TYPE=F,LRECL=80,BLKSIZE=800,
         INCLUDE=(25,1,CH,EQ,C'A')
JOINKEYS FILE=F2,FIELDS=(35,10,A),TYPE=F,LRECL=80,BLKSIZE=1600,
         OMIT=(1,1,CH,EQ,C'B')
```

Figure 61. Sample JOINKEYS statements

In Figure 61, the JOINKEYS control statements specify the following:

- Records read from SORTIN1 (F1), which contains fixed-length 80 character records with a block size of 800, will be filtered using the INCLUDE parameter to include all records containing an 'A' in position 25.
- Records read from SORTIN2 (F2), which contains fixed-length 80 character records with a block size of 1600, will be filtered using the OMIT parameter to include all records containing a 'B' in position 1.
- The records from each of these files will be joined by their common keys (F1 15,10 and F2 35,10) based on the JOIN and REFORMAT control statements. See "JOIN Control Statement" on page 2.86 and "REFORMAT Control Statement" on page 2.224 for further information.

MERGE Control Statement

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

The format of the MERGE control statement is illustrated below:

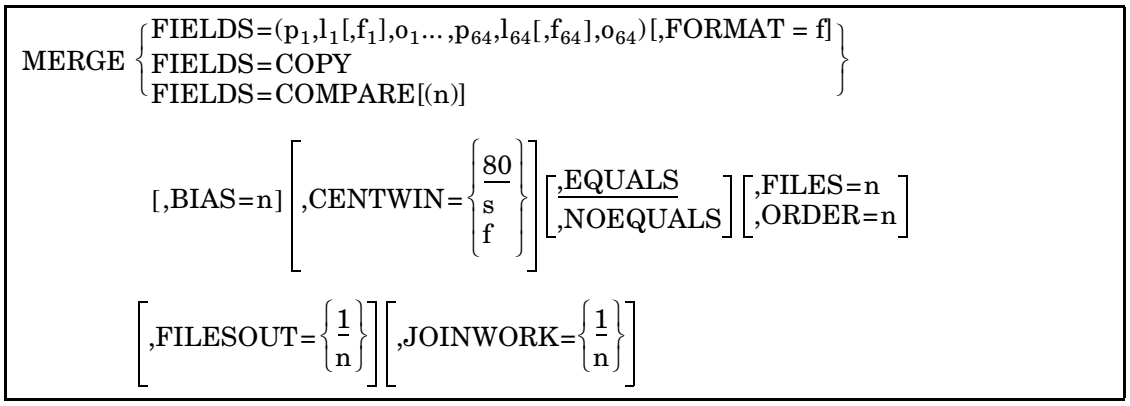


Figure 62. MERGE Control Statement Format

FIELDS Parameter (Required for a Merge)

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

As many as sixty-four control fields can be specified. List the control fields in order of greatest to least priority, with the primary control field (p_1,l_1,f_1,o_1) listed first, followed by progressively less significant fields (p_2,l_2,f_2,o_2,..., p_{64},l_{64},f_{64},o_{64}).

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

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. 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 integral number of bytes except for the length of a binary control field, which may be specified in bits. For example, a length value of 0.5 refers to a binary control field 6 bits long. For signed fields, the length value must include the area occupied by the sign.

f The format value indicates the data format. Refer to Table 16 for a list of valid formats. If all the control fields have the same format, you can specify the format value once by using the FORMAT=f subparameter.

o The order value indicates how the field is to be collated. A indicates Ascending order. D indicates Descending order.

Valid Formats for Control Fields

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

Code	Data Format	Acceptable field length in bytes
AC	EBCDIC characters are translated to their ASCII equivalents before sorting.	1 to 256
AQ	Character. Sorts records according to an alternate sequence specified either in the ALTSEQ control statement or as an installation default.	1 to 256
ASL	Leading separate sign. An ASCII + or - precedes a numeric field. One digit per byte.	2 to 256
AST	Trailing separate sign. An ASCII + or - trails a numeric field. One digit per byte.	2 to 256
BI	Binary. Unsigned.	1 bit to 4092
CH	Character. Unsigned. Characters are collated according to the EBCDIC collating sequence unless LOCALE has been used to chose an alternate set of collating rules based on a specific national language. LOCALE can be set on the OPTION statement or as an installation default.	1 to 4092
CLO or 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 or 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 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 or LS	Leading separate sign. An EBCDIC + or - precedes a numeric field. One digit per byte. CMP=CLC is forced.	2 to 256
CST or TS	Trailing separate sign. An EBCDIC + or - follows a numeric field. One digit per byte. CMP=CLC is forced.	2 to 256
FI	Fixed point. Signed.	1 to 256
FL	Signed floating point. Normalized.	2 to 16
PD	Packed decimal. Signed.	1 to 256

Table 16. (Page 1 of 3) Format Code Chart

Code	Data Format	Acceptable field length in 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
SFF	Signed free format. Decimal digits (0-9) are extracted from right to left to form a number value. A character of - or) found within the field will cause the value to be treated as a negative number. All other non-decimal digit values in the field are ignored.	1 to 44
UFF	Unsigned free format. Decimal digits (0-9) are extracted from right to left to form a number value. All non-decimal digit values in the field are ignored.	1 to 44
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 is 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 formats. Enable sorting or merging a variety of date fields. The formats can process dates ending or starting with year digits and non-date data commonly used with the dates. Two-digit years are interpreted according to the CENTWIN setting. In most cases, for CH, ZD, and PD date fields the full-date data formats are easier to use than the 2-digit year formats. For details on these formats see Table 17 on page 2.99.	

Table 16. (Page 2 of 3) Format Code Chart

Code	Data Format	Acceptable field length in 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 or CTO	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

Table 16. (Page 3 of 3) Format Code Chart

The following table describes the valid full-date formats: Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y.

Full-Date Format	Date 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	dddyy	5
		xxxxyy	mmddyy	6
Y2X	PD	xyy (X'xyys')	qyy	2
		xxxyy (X'xxxyys')	dddyy	3
Y2Y	PD	xxyy (X'0xxyys')	mmyy	3
		xxxxyy (X'0xxxxyys')	mmddyy	4

Note: The following symbols are used in the table:

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

Table 17. Full-Date Formats

For information on using the full-date formats, see "CENTWIN (Century Window) Processing with MERGE: Full-Date Formats" on page 2.106, "CENTWIN (Century Window) Processing with SORT: Full-Date Formats" on page 2.240, and "Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC or OUTFIL OUTREC: Full-Date Data" on page 2.196.

For information on the 2-digit year formats (Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z) plus the related data format PD0), see "CENTWIN (Century Window) Processing with SORT: 2-Digit Year Formats" on page 2.235 and "Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC or OUTFIL OUTREC: 2-Digit Year Data" on page 2.194.

Rules for Specifying Control Fields

- The sum of the lengths of all control fields cannot exceed 4092 bytes. When the EQUALS option is specified, the sum of the lengths cannot exceed 4088 bytes.
- Control fields can be in contiguous or noncontiguous 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.

Comparing PD and ZD Control Fields

When you specify CMP=CPD, SyncSort uses the Compare Decimal (CP) instructions for the comparison, which means that ZD data fields are packed first then compared. This method has performance advantages. However, invalid PD data may cause a Data Exception abend and program termination. Moreover, the integrity of ZD fields is only guaranteed when the fields contain valid ZD data. Since the zone bits (the left most 4 bits of each byte) are lost during PACKing, UNPKing the field only restores valid ZD data to its original state. For example, blanks are transformed into ZD zeros and alphabetic character data that packs to a valid PD field is converted into valid ZD data.

When CMP=CLC is in effect, SyncSort does not perform data validation and the integrity of the output is maintained.

FIELDS=COPY (Required for a Copy)

Use FIELDS=COPY to copy input file(s). Other control statements such as INCLUDE/OMIT, INREC, OUTREC and OUTFIL and user exit routines can be specified in conjunction with a copy application, allowing you to edit and reformat the file(s) without merging them.

The SUM control statement cannot be specified with FIELDS=COPY.

FIELDS=COMPARE (Required for a Compare)

Use FIELDS=COMPARE[(n)] to determine if two files are identical. When the COMPARE feature is specified, SyncSort compares the files record for record, byte for byte, and the sorting process is bypassed.

The value specified for 'n' indicates how many unequal pairs of record SyncSort should flag before terminating the job. The diagnostic information provided distinguishes among three cases:

- Fully identical files.
- Files of different sizes where the smaller file consists of the larger file's first n records, in the same order.
- Files with different records. In this case, dumps are produced that identify the first pair of unequal records or as many pairs of unequal records as the 'n' value specifies.

The SUM control statement cannot be used with FIELDS=COMPARE.

BIAS Parameter (Optional)

The BIAS parameter is ignored.

CENTWIN Parameter (Optional)

The CENTWIN option defines a sliding or fixed 100-year window that determines the century to which 2-digit year data belongs when processed by SORT, MERGE, INCLUDE, INREC, OMIT, OUTREC, or OUTFIL OUTREC control statements.

There are two types of date data formats: 2-digit year and full-date:

- For details on CENTWIN processing and the 2-digit year formats with the MERGE control statement, see "CENTWIN (Century Window) Processing with MERGE: 2-Digit Year Formats" on page 2.102. For similar information for the SORT control statement, see "CENTWIN (Century Window) Processing with SORT: 2-Digit Year Formats" on page 2.235.

  For information on using INCLUDE/OMIT with 2-digit year formats, see "Specifying Standard Comparisons for Date Fields" on page 2.50.

  For information on using the 2-digit year formats for OUTREC processing, see "Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC or OUTFIL OUTREC: 2-Digit Year Data" on page 2.194 and "Sample OUTREC Control Statements with CENTWIN Processing: 2-Digit Year Formats" on page 2.213.

- For details on CENTWIN processing and the full-date formats with the MERGE control statement, see "CENTWIN (Century Window) Processing with MERGE: Full-Date Formats" on page 2.106. For similar information for the SORT control statement, see "CENTWIN (Century Window) Processing with SORT: Full-Date Formats" on page 2.240.

For information on using INCLUDE/OMIT with full-date formats, see “Specifying Standard Comparisons for Date Fields” on page 2.50 and “Sample INCLUDE/OMIT Control Statements with Date Data” on page 2.60.

For information on using the full-date data formats for OUTREC processing, see “Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC or OUTFIL OUTREC: Full-Date Data” on page 2.196.

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 first input file are written before those from the second input file, etc. The NOEQUALS parameter specifies that equal-keyed records from different files be written in random order. EQUALS is the default.

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 or DUPKEYS control statement, the first of the equally-keyed records is retained with the sum; all other records are deleted after the specified field(s) have been summed.

FILES/ORDER Parameter (Optional)

The FILES parameter or the ORDER parameter specifies the number of input files to be merged. The 'n' value can be any number from 1 to 32. **NOTE:** When using the JOIN process or the COMPARE process, two input files are assumed.

FILESOUT Parameter (Optional)

The FILESOUT parameter specifies the number of output files and is *required* when creating multiple output files. The 'n' value can be any number from 1 to 32. If the FILESOUT parameter is not specified, one output file is assumed.

JOINWORK Parameter (Optional)

The JOINWORK parameter specifies the number of sortwork files that the join process will use. The ‘n’ value can be any number from 1 to 9. If the join process is used and JOINWORK is not specified, JOINWORK=1 is assumed

CENTWIN (Century Window) Processing with MERGE: 2-Digit Year Formats

The CENTWIN run-time or installation option acts on 2-digit year data that spans centuries. CENTWIN treats 2-digit year data as a 4-digit year and sequences the data according to the 4-digit representation.

For information on using full-date formats, see “CENTWIN (Century Window) Processing with MERGE: Full-Date Formats” on page 2.106.

CENTWIN generates a century window (for example, 1950 through 2049) that determines what century a two-digit year field belongs to. CENTWIN ensures that year data spanning centuries will be sequenced correctly. 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 more information on specifying the CENTWIN option, see “CENTWIN Parameter (Optional)” on page 2.101.

CENTWIN processing only applies to data defined as date data formats: Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z.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/MERGE processing, you should be familiar with the CENTWIN-related data formats.

The following describes each of the 2-digit year formats and provides example MERGE control statements:

Note: For simplicity, the sample date fields in the example MERGE statements below begin at byte 20. Also note that date data is always sorted in the following order: year (yy), month (mm), day (dd).

- Y2B

  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 18. Possible Values Representing Year Data with Y2B

- Y2C and Y2Z

  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 mmddyy 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,  * sort yy field as 4-digit year
              20,2,CH,A,   * sort mm field
              22,2,CH,A)   * sort dd field
```

Figure 63. Sample MERGE Control Statement with Y2C

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'.

- Y2D

  This format is used to sort 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

  yyddds

  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 years processing, use the following MERGE control statement:

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

Figure 64. Sample MERGE Control Statement with Y2D

- Y2P

  This format is used to sort or merge 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 two bytes. For example, a packed decimal date of the form yymmdd would be stored as four 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:

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

Figure 65. Sample MERGE Control Statement with Y2P

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

- PD0

  This format is used to sort or merge 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'ymmddC' as X'mmdd'.

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

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

Figure 66. Sample MERGE Control Statement with PD0

- Y2S

  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 begins 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:

  - 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.

CENTWIN (Century Window) Processing with MERGE: Full-Date Formats

SyncSort's full-date 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, dddyy, or mmddyy)

- 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 year formats. The 2-digit year 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. For information on the 2-digit year formats (Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z), see Table 16 on page 2.96 and "CENTWIN (Century Window) Processing with MERGE: 2-Digit Year Formats" on page 2.102.

The following two examples illustrate how you might use Table 16 on page 2.96 to develop SORT control statements:

- 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 date field starts in position 30, you would specify the following SORT control statement to sort the date in descending order:

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

Figure 67. Sample SORT Control Statement with Y2Y

  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)
```

Figure 68. Sample SORT Control Statement with Y2T

  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 be 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).

Table 16 on page 2.96 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 table shows the order for ascending collation when using full-date formats with the CENTWIN option:

Full-Date Format	Data 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 19. Collation Order for Full-Date Formats

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 then higher century dates.

Sample MERGE Control Statements

```
MERGE FIELDS=(1,5,CH,A,10,2,PD,D,30.2,4,BI,A)
```

Figure 69. Sample MERGE Control Statement

This sample MERGE 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
```

Figure 70. Sample MERGE Control Statement

This MERGE statement specifies a copy operation.

MODS Control Statement

The MODS control statement specifies a user exit routine and is required with an exit. Refer to “Chapter 8. User Exit Programs” for a detailed explanation of how to specify exit programs.

The format of the MODS statement is illustrated below.

```
MODS PHn=(name,loading information,Enn1,...,Ennn)
```

Figure 71. MODS Control Statement Format

PHn Parameter (Required)

The PHn parameter specifies the sort/merge phase in which the exit point occurs. Replace the 'n' value with the phase number.

- PH1 specifies Phase 1. E11, E15, E17 and E18 exits occur in Phase 1.
- PH2 specifies Phase 2. E21, E25 and E27 exits occur in Phase 2. (Exits E21 and E27 are currently unsupported.)
- PH3 specifies Phase 3. E31, E32, E35, E37, E38 and E39 exits occur in Phase 3.

Only E15, E32, and E35 may be specified if the exit program is a COBOL/VSE, C/VSE, or REXX/VSE program.

The PHn parameter also specifies the exit’s name, loading information and type (Enn).

name

When SyncSort is to load the exit program(s), all of the exits that occur in a sort/merge phase must be included in one loadable program and must have a unique name. This program must be catalogued in the core image library. User-supplied names are permitted, but they must follow standard VSE name conventions.

If the exit program(s) are *already* loaded in the partition in main storage, the name can be omitted. In that case, insert a comma to indicate the missing name.

If an exit program is written in COBOL/VSE or C/VSE, it must be compiled and link-edited into a loadable phase in a core image library. Specify the name of the phase as the name of the exit. COBOL/VSE exits or C/VSE exits must be loaded by SyncSort. They cannot be pre-loaded in the main storage.

Note that COBOL/VSE and C/VSE exits require activation of SyncSort LOCALE and COBOL/C Exit Facility as described in the *SyncSort for z/VSE Installation Guide*.

If the exit program is written in REXX/VSE, it must be catalogued in one of the libraries in the PROC search chain. Specify the name of the REXX/VSE exit here. SyncSort does not support a compiled REXX/VSE exit.

loading information

When SyncSort is to load the exit program(s), loading information must be supplied in one of the following ways:

- If the exit program is a compiled COBOL/VSE program, specify the word COBOL here.

  Note that COBOL/VSE requires additional GETVIS storage for run-time library loading. Allow at least 500K of GETVIS when executing SyncSort with the COBOL/VSE user exit.

- If the exit program is a compiled C/VSE program, specify C here.

  Note that C/VSE requires additional GETVIS storage for run-time library loading. Allow at least 4 MB of GETVIS when executing SyncSort with the C/VSE user exit.

- If the exit program is a REXX/VSE program, specify the word EXEC or REXX here.

- If none of the above conditions apply, do the following:

  – Specify the length of the program(s) as a decimal number of bytes. The letter 'L' must precede the number, for instance, L2500. This method of specifying loading information is preferred because it allows SyncSort to load the program(s) into the optimal location within the partition. However, the program(s) must be either self-relocating or a relocating loader must be present in the system in order to use this method.

  – Specify the absolute loading address of the program as a decimal number. This must be a virtual address if SyncSort is running in virtual mode and a real address if SyncSort is running in real mode. To use space most efficiently, place the program(s) at the highest address within the sort/merge partition. This method of supplying loading information must be used if the program(s) cannot be relocated.

Loading information can be omitted if the program(s) are already loaded into main storage. In that case, insert a comma to indicate the missing loading information.

type (Enn) Specify the type of exit as E15, E32, E35, etc. Supply the exit type for each exit used within a sort/merge phase. Replace Enn_1 with the number of the first exit used, Enn_2 with the number of the second, etc.

Only E15, E32, and E35 may be specified for COBOL/VSE exits, C/VSE exits, and REXX/VSE exits.

Specifying Exits in More Than One Phase

If an application has exits in two or three sort/merge phases, specify the PHn parameter two or three times. Include a comma between the specifications of the PHn parameter as illustrated below:

```
PH1=(information),PH2=(information),PH3=(information)
```

Figure 72. Multiple Specifications of the PHn Parameter

Sample MODS Control Statement

```
MODS PH1=(TEXAS1,L1000,E15,EI8),PH3=(TEXAS2,L2500,E31,E35,E38)
```

Figure 73. Sample MODS Control Statement

This sample MODS statement specifies the following information:

- User exit points occur in Phase1 and Phase 3.
- The catalogued name of the Phase 1 routines is TEXAS1. The total length of the routines is 1000 bytes. The exits are an E15 and an E18.
- The catalogued name of the Phase 3 routines is TEXAS2. The total length of the routines is 2500 bytes. The exits are an E31, E35 and E38.

OMIT Control Statement

Refer to “INCLUDE/OMIT Control Statement” on page 2.39 for an explanation of the OMIT control statement.

OPTION Control Statement

The OPTION control statement specifies options for the application.

The format of the OPTION control statement is illustrated in Figure 74, below. Delivered defaults are underlined, but may have been changed when SyncSort for z/VSE was installed.

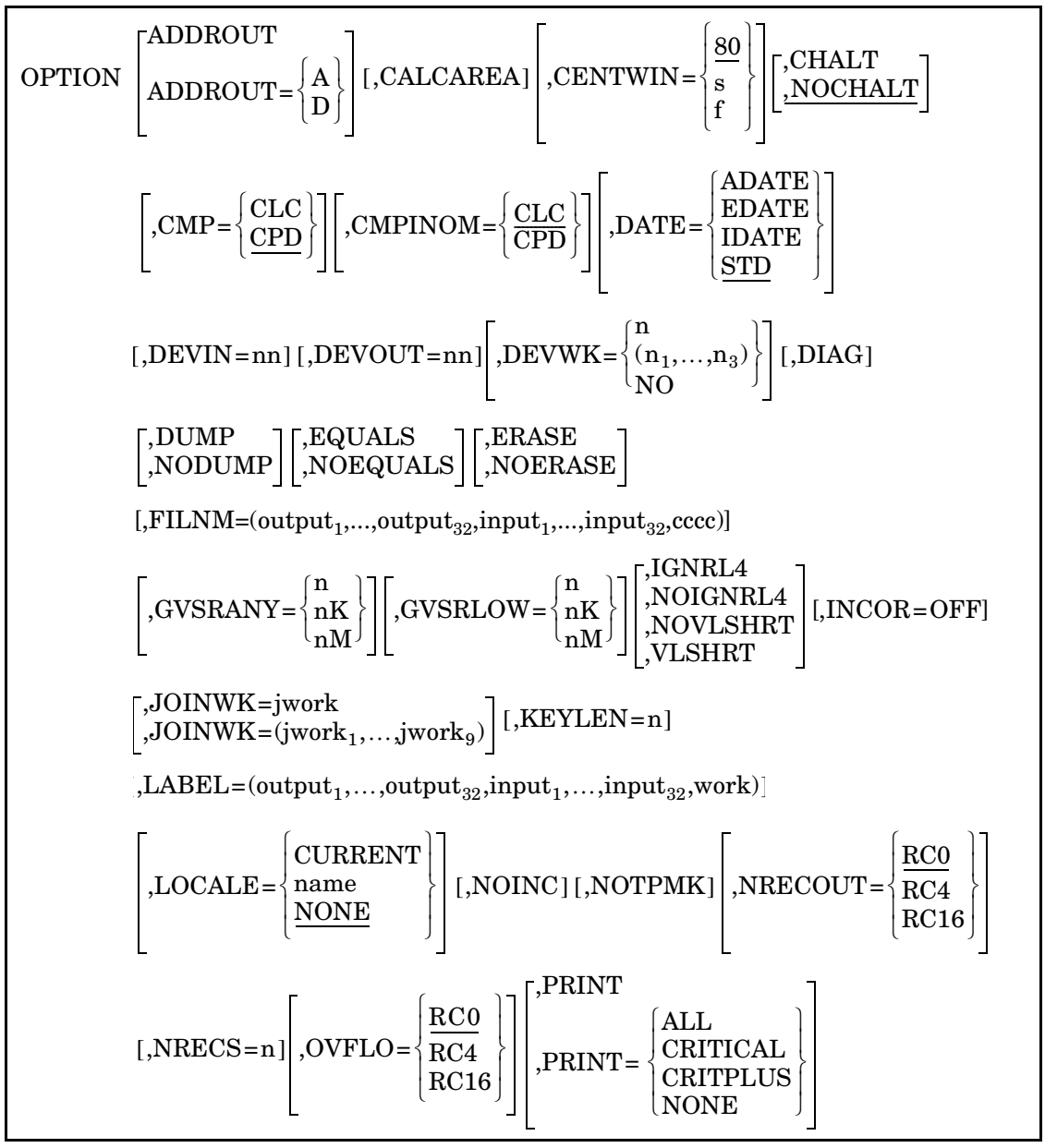


Figure 74. (Page 1 of 2) OPTION Control Statement Format

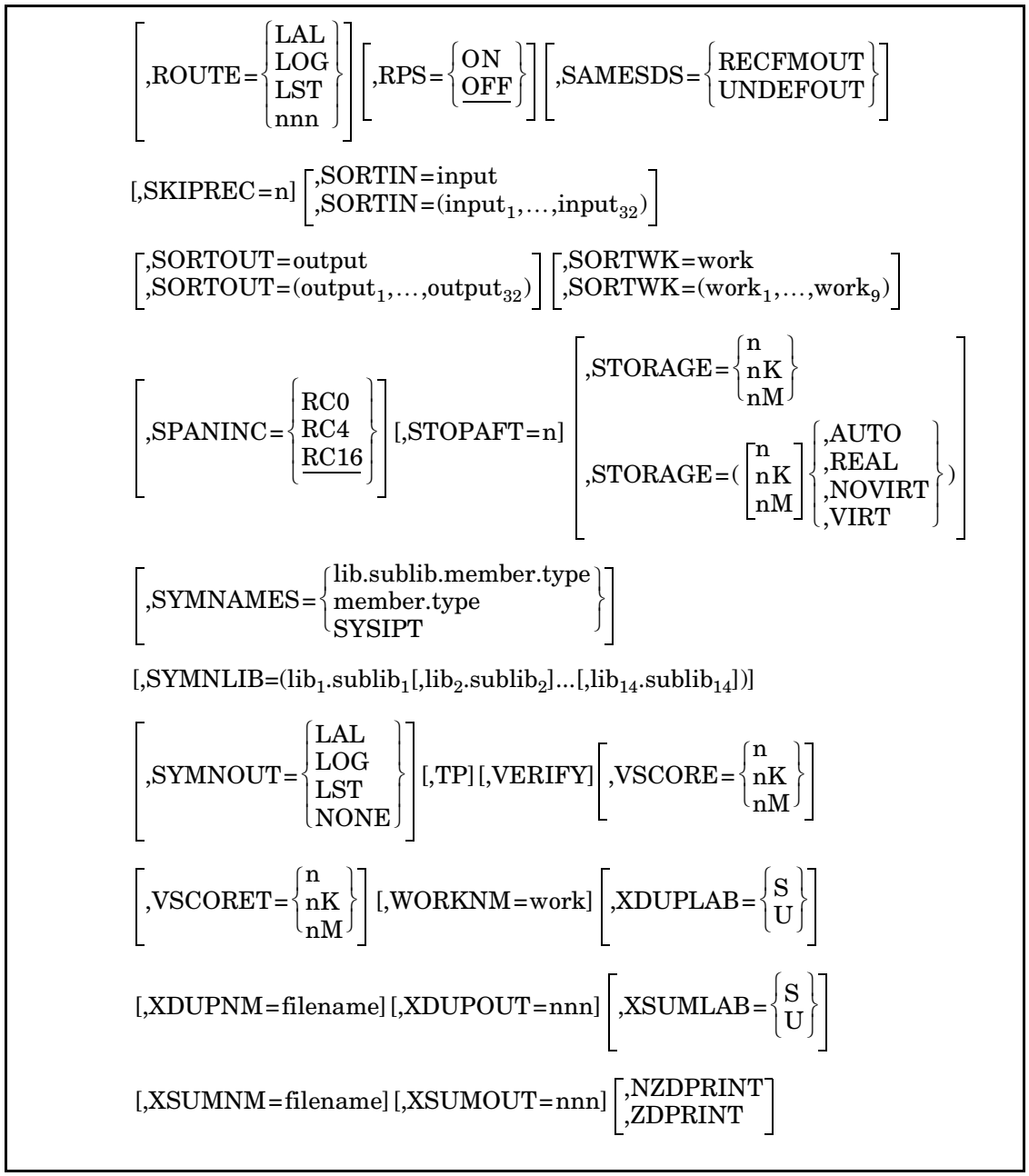


Figure 74. (Page 2 of 2) OPTION Control Statement Format

ADDROUT Parameter (Optional)

Specify ADDROUT or ADDROUT=A if you want the output file to contain only the addresses of the sorted input records. Specify ADDROUT=D if you want the output file to contain only the sort control fields and the addresses.

The ADDROUT option is supported for sequential disk input files, VSAM managed SAM files treated as VSAM, and VSAM files, except for VSAM KSDS-defined files with spanned records. The ADDROUT option is not valid for FBA devices or for ASCII data. For SAM

files, the addresses of the records will be given in the standard 10-byte binary form, which follows:

Fields	Description
M	Input file number. (The values 0 to 31 will appear in this byte, indicating SORTIN1-SORTI32.)
BB	Bin number. (Always 00.)
CC	Cylinder number
HH	Head number
R	The number of the record or the record block on the input file track.
DD	For fixed-length records, 00 for unblocked files or the byte displacement from zero of the record within the block. For variable-length records, 04 for unblocked files or the byte displacement from zero of the record within the block.

Table 20. ADDROUT Field Descriptions for SAM Files

If VSAM or VSAM managed SAM files are used, the addresses are 5-byte binary numbers in this format:

Fields	Description
X	Input file number. (The values 1 to 32 will appear in this byte, indicating SORTIN1-SORTI32.)
YYYY	Relative byte address (RBA) for key-sequenced (KSDS) or entry-sequenced (ESDS) data sets; relative record number for relative record data sets (RRDS).

Table 21. ADDROUT Field Descriptions for VSAM Files

Rules for Specifying the ADDROUT Parameter

- Input files must be on disk devices. Output files can be on any device.
- If input files are VSAM or VSAM managed SAM, the VSAM parameter must be specified on the INPFIL control statement.
- Mixing SAM input with VSAM or VSAM managed input is not permitted with ADDROUT.
- For SAM files, the output blocksize must be a multiple of 10 bytes if ADDROUT or ADDROUT=A is specified and a multiple of 10 bytes plus the sum of the control fields if ADDROUT=D is specified. For VSAM files, the output blocksize must be a multiple of 5

bytes if ADDROUT or ADDROUT=A is specified and a multiple of 5 bytes plus the sum of the control fields if ADDROUT=D is specified.

- If the output is written to tape, the output blocksize must be at least 18 bytes in length. If the last block is shorter than 18 bytes, it will be padded with blanks to some multiple of the record length so that the length is at least 18 bytes. For example, a too-short last block for 15-byte records will be padded to 30 bytes.

- ADDROUT is ignored if any of the following control statements is specified: INREC, OUTREC, SUM, MERGE, SORT FIELDS=COPY.

- If ADDROUT or ADDROUT=A is specified along with the KEYLEN parameter, the output records will not contain keyed fields. If ADDROUT=D is specified with KEYLEN, the keys will appear as part of the control fields rather than in count-key-data format.

- The ADDROUT parameter cannot be used for FBA devices or for VSAM key-sequenced (KSDS) input files that contain spanned records. ADDROUT addresses produced are meaningless in these cases.

- The ADDROUT parameter cannot be used with ASCII data.

Specifying the RECORD Statement and the ADDROUT and KEYLEN Parameters

If specified, the l_2 and l_3 values of the RECORD statement must reflect the ADDROUT and KEYLEN specifications. If an E35 exit changes the record length at output, the new length must be specified as the l_3 value.

The chart below illustrates how to calculate the length values on the RECORD control statement when the ADDROUT and KEYLEN parameters are specified.

FIXED-LENGTH RECORDS					
	l_1	**l_2**	**l_3**	**l_4**	**l_5**
KEYLEN	record length + key length	record length + key length	record length + key length	do not use	do not use
ADDROUT=A (with SAM files)	record length	sum of control fields + 10	10	do not use	do not use
ADDROUT=A (with VSAM files)	record length	sum of control fields + 5	5	do not use	do not use
KEYLEN and ADDROUT=A	record length + key length	sum of control fields + 10	10	do not use	do not use
ADDROUT=D (with SAM files)	record length	+ 10	sum of control fields + 10	do not use	do not use
ADDROUT=D (with VSAM files)	record length	sum of control fields + 5	sum of control fields + 5	do not use	do not use
KEYLEN and ADDROUT=D	record length + key length	sum of control fields + 10	sum of control fields + 10	do not use	do not use
VARIABLE-LENGTH RECORDS					
ADDROUT=A (with SAM files)	length of longest record + 4	sum of control fields + 10	10	length of shortest record + 4	length of most frequent records + 4
ADDROUT=A (with VSAM files)	length of longest record + 4	sum of control fields + 5	5	length of shortest record + 4	length of most frequent records + 4
ADDROUT=D (with SAM files)	length of longest record + 4	sum of control fields + 10	sum of control fields + 10	length of shortest record + 4	length of most frequent records + 4
ADDROUT=D (with VSAM files)	length of longest record + 4	sum of control fields + 5	sum of control fields + 5	length of shortest record + 4	length of most frequent records + 4

Table 22. RECORD Length Values Chart with the ADDROUT and KEYLEN Parameters

CALCAREA Parameter (Optional)

The CALCAREA parameter is used to obtain an estimate of the amount of work space required for a particular sort. The SIZE parameter must be specified on the SORT control statement to provide an accurate estimate of the size of the input file(s). The sort will terminate after Phase 0 with a message estimating the number of tracks needed for the work areas. Any control statement errors will also be noted.

When the CALCAREA parameter is specified, it is not necessary to mount any of the files or supply any JCL beyond ASSGN statements. Remember, however, to complete the JCL when you execute the job.

CHALT/NOCHALT Parameters (Optional)

If the CHALT parameter is specified, all CH format data will be treated as AQ format data during sort, merge, include and omit operations. Specifying NOCHALT prevents conversion. NOCHALT is also the delivered default.

CENTWIN Parameter (Optional)

Figure 75. CENTWIN Parameter Format

The CENTWIN option defines a sliding or fixed 100-year window that determines the century to which 2-digit year data belongs when processed by SORT, MERGE, INCLUDE, INREC, OMIT, OUTREC or OUTFIL OUTREC control statements.

Date data formats (Y2B, Y2C, Y2D, Y2P, Y2S, Y2T, Y2U, Y2V, Y2W, Y2X, Y2Y, and Y2Z)work with CENTWIN to treat a 2-digit year value as a 4-digit year. The date data formats can be specified on three types of SyncSort control statements:

- On a SORT or MERGE control statement to correctly collate 2-digit years that span century boundaries.
- On an OUTREC or OUTFIL OUTREC control statements to convert 2-digit years (yy) to 4-digit output (yyyy).
- On an INCLUDE or OMIT statement to select records to be processed based on dates.

There are two types of date data formats: 2-digit year and full-date:

- For details on CENTWIN processing and the 2-digit year formats with the MERGE control statement, see “CENTWIN (Century Window) Processing with MERGE: 2-Digit Year Formats” on page 2.102. For similar information for the SORT control statement, see “CENTWIN (Century Window) Processing with SORT: 2-Digit Year Formats” on page 2.235.

  For information on using INCLUDE/OMIT with 2-digit year formats, see “Specifying Standard Comparisons for Date Fields” on page 2.50.

For information on using the 2-digit year formats for OUTREC processing, see “Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC or OUTFIL OUTREC: 2-Digit Year Data” on page 2.194 and “Sample OUTREC Control Statements with CENTWIN Processing: 2-Digit Year Formats” on page 2.213.

- For details on CENTWIN processing and the full-date formats with the MERGE control statement, see “CENTWIN (Century Window) Processing with MERGE: Full-Date Formats” on page 2.106. For similar information for the SORT control statement, see “CENTWIN (Century Window) Processing with SORT: Full-Date Formats” on page 2.240.

  For information on using INCLUDE/OMIT with full-date formats, see “Specifying Standard Comparisons for Date Fields” on page 2.50 and “Sample INCLUDE/OMIT Control Statements with Date Data” on page 2.60.

  For information on using the full-date data formats for OUTREC processing, see “Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC or OUTFIL OUTREC: Full-Date Data” on page 2.196.

CENTWIN ensures that year data spanning centuries will be sequenced correctly. 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 that 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 default is s=80, which represents 80 years prior to the current year as the starting year of a century window.

- CENTWIN=f specifies a fixed century window. The variable f is a four-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 has a value less than the last 2 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. 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
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
01	2001
49	2049

CMP=CLC/CMP=CPD Parameter (Optional)

The CMP parameter specifies how PD and ZD collation fields will be compared.

When you specify CMP=CPD, SyncSort uses the Compare Decimal (CP) instructions for the comparison, which means that ZD data fields are packed first then compared. This method has performance advantages. However, invalid PD data may cause a Data Exception abend and program termination. Moreover, the integrity of ZD fields is only guaranteed when the fields contain valid ZD data. Since the zone bits (the left most 4 bits of each byte) are lost during PACKing, UNPKing the field only restores valid ZD data to its original state. For example, blanks are transformed into ZD zeros and alphabetic character data that packs to a valid PD field is converted into valid ZD data.

When CMP=CLC is in effect, SynsSort does not perform data validation and the integrity of the output is maintained.

Before specifying this parameter, find out whether the default setting (CMP=CPD) has been changed at installation time.

CMPINOM=CLC/CMPINOM=CPD Parameter (Optional)

The CMPINOM parameter is similar to the CMP parameter, but affects the processing of the INCLUDE/OMIT statement or OUTFIL INCLUDE/OMIT statement.

CMPINOM specifies the type of INCLUDE/OMIT comparison operation that will be used when comparing packed and zoned decimal fields.

DATE Parameter (Optional)

The DATE parameter specifies the format of the current system date generated by the &DATE parameter (see “HEADER1/HEADER2 Parameters (Optional)” on page 2.149 and “TRAILER1/TRAILER2 Parameters (Optional)” on page 2.164. The date format may be set as follows: ADATE signifies American date format (mm/dd/yy); EDATE signifies European date format (dd/mm/yy); IDATE signifies ISO date format (yy/mm/dd); STD signifies generated system date format. In the absence of any user provided formatting information, the standard default format, STD will appear as the date format.

DEVIN/DEVOUT Parameters (Optional)

The DEVIN/DEVOUT parameters identify the system’s tape input and output devices. Each tape device is represented by 'nn', a two-digit code. Use Table 23 below to determine the correct value to substitute for 'nn'. If the device is not manufactured by IBM, use the IBM model number, which corresponds to the device’s transfer rate and density. Accurate specification of tape input/output devices may improve sort performance. The DEVIN and DEVOUT parameters are usually specified at installation time.

nn VALUES	IBM MODEL NO.	TRACKS	DENSITY	
			BPI	KB/S
32	3420-3	9	1600	120
33	3420-3	7	800	60
34	3420-3	7	556	41.7
35	3420-5	9	1600	200
36	3420-5	7	800	100
37	3420-5	7	556	69.5
38	3420-7	9	1600	320
38	3430	9	6250	312.5
39	3420-7	7	800	160
40	3420-7	7	556	111.2
41	3420-4	9	6250	470
42	3420-6	9	6250	780
42	3422	9	6250	780
43	3420-8	9	6250	1250
44	3410(1-2)	9	800	20
45	3410-2	9	1600	40
46	3410-3	9	800	40
47	3430	9	1600	80
47	3410-3	9	1600	80
48	8809	9	1600	20(160)
49	3480	18	38K	1500
50	3490	18	38K	3000
51	3590	128	----	9000

Table 23. DEVIN and DEVOUT Values

DEVWK Parameter (Optional)

The DEVWK parameter overrides the default disk device types in SYNCMAC. All device types specified must match device types used for SORTWK.

You may specify up to three device types, but you may not specify both FBA and CKD devices (for example, models 3380 and 3370) in the same DEVWK parameter.

Specifying SORTWK device types can improve the performance of small sorts. However, do not specify multiple SORTWK device types needlessly since this may have a negative impact on performance.

When specifying device types, you may use a two- or three-digit 'n' value to represent the device type, or you may use a four-digit IBM model number. See Table 24 below for the 'n' values corresponding to particular model numbers.

To override the default in SYNCMAC and have SyncSort identify the SORTWK device types directly, specify DEVWK=NO.

n VALUE	IBM MODEL NO.
08	3380
09	3310, 3370, 9332, 9335, 9336
138	3390
139	9345

Table 24. DEVWK Values

Note: If your device is not listed, use the value corresponding to the equivalent IBM model number or the actual four-digit model number.

DIAG Parameter (Optional)

Specify the DIAG parameter to produce a listing of diagnostic messages for each SyncSort phase. If the application terminates successfully, additional PDUMPS will be provided.

DUMP/NODUMP Parameter (Optional)

Specify the DUMP parameter to produce a dump if the application terminates before the end of the sort or merge. Specify NODUMP if you do not want a dump. NODUMP is the default if the program terminates prematurely in Phase 0 and DUMP is the default if the program terminates prematurely in Phases 1, 2, or 3.

EQUALS/NOEQUALS Parameter (Optional)

Specify the EQUALS parameter to preserve the original order of records with equal control fields. These records are written in the same order in which they are read. Specify NOEQUALS if the original record order need not be preserved. The defaults are NOEQUALS for a sort and EQUALS for a merge.

When EQUALS 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.

The EQUALS and NOEQUALS parameters can also be specified on the SORT and MERGE control statements.

ERASE Parameter (Optional)

Specify ERASE to erase the contents of the work files opened by the sort. (If an incore sort is performed, no files are erased.)

FILNM Parameter (Optional)

Specify the FILNM parameter to direct SyncSort for z/VSE to use filenames other than the default names of SORTOUT, SORTOFx, SORTOxx, SORTINx, SORTIxx and SORTWKx. The alternate filenames must also be used in the TLBL/DLBL job control statements. If the FILNM parameter is not specified, the default names must be used.

Observe the following rules when specifying the FILNM parameter:

- The alternate filename can contain up to seven alphanumeric characters. The first character must be a letter.
- If user-supplied filenames are used with some default filenames, the FILNM parameter can be specified. Insert a comma to indicate when a default filename is being used. Or, write out the default filename(s).

The default names for multiple SORTIN files are: SORTIN1, SORTIN2, SORTIN3,...,SORTIN9, SORTI10,...,SORTI32.

The default names for multiple SORTOUT files are: SORTOUT, SORTOF2, SORTOF3,...,SORTOF9, SORTO10,...SORTO32.

The default names for multiple SORTWK files are: SORTWK1, SORTWK2, SORTWK3,...,SORTWK9. For SORTWK files, only the first four characters ('SORT') can be changed by specifying the FILNM parameter.

The format of the FILNM parameter is illustrated below.

FILNM=(output$_1$,...,output$_{32}$,input$_1$,...,input$_{32}$,cccc)

Figure 76. FILNM Format

Insert commas to represent files for which alternate filenames are not specified. The FILESOUT parameter of the SORT or MERGE control statement determines the number of commas SyncSort for z/VSE expects before finding the filename of the first input file.

If cccc is specified to change the prefix of the sort work files and fewer than nine input files are input to the sort/merge, you must insert commas to represent any files that are not named. It is not necessary to insert commas to represent unused input filenames if cccc is not specified and nine or more input files are used.

An example of the FILNM parameter is illustrated below.

```
FILNM=(OUT78,,,,IN78,,,,,,SYNC)
```

Figure 77. Sample FILNM Parameter

This example specifies one output file (OUT78), four input files (SORTIN1, SORTIN2, SORTIN3 and IN78). The sort work files will be named SYNCWKx.

If additional default input filenames were to be used after IN78 and the SORTWK prefix was not to be changed, it would not be necessary to code commas for the additional input filenames. SyncSort would reference them as SORTIN5, SORTIN6, etc.

GVSRANY Parameter

Figure 78. GVSRANY Parameter Format

This parameter instructs SyncSort to reserve at least the specified amount of 31-bit GETVIS storage for the system to use. The default is 32K.

GVSRLOW Parameter

Figure 79. GVSRLOW Parameter Format

This parameter instructs SyncSort to reserve at least the specified amount of 24-bit GETVIS storage for the system to use. The default is 128K.

INCOR=OFF Parameter (Optional)

The INCOR=OFF parameter instructs SyncSort for z/VSE *not* to perform an incore sort.

IGNRL4/NOIGNRL4/VLSHRT/NOVLSHRT Parameter (Optional)

The IGNRL4 parameter indicates that SyncSort for z/VSE should *not* check that the variable-length input record is at least as large as the value of l_4 specified by the user on the RECORD statement (or computed by the sort if the l_4 value was not specified). Specifying NOIGNRL4 will override IGNRL4. NOIGNRL4 is the default. The VLSHRT parameter is equivalent to the IGNRL4 parameter. The NOVLSHRT parameter is equivalent to NOIGNRL4 parameter. Note that only one of these parameters can be issued on an OPTION statement.

IGNRL4 overrides any default L4FILL set by SYNCMAC; no L4FILL padding will be done. However, the IGNRL4 parameter can be overridden on a per file basis by the INPFIL L4FILL parameter, which will allow padding for those specific files.

JOINWK Parameter (Optional)

See "SORTOUT, SORTIN, SORTWK, and JOINWK Parameters (Optional)" on page 2.133 for a description of this parameter.

KEYLEN Parameter (Optional)

The KEYLEN parameter indicates that keys precede the input records and all or part of these keys are to be used as control fields and/or the keys are to be included in the output record. The 'n' value represents the number of bytes in the key field.

If the input records have keys and the KEYLEN parameter is not specified, SyncSort will not read the keys and it will not be possible to sort on these keys nor to have them written to the output file.

The KEYLEN parameter is supported only for SAM files on CKD disk devices.

Rules for Specifying the KEYLEN Parameter

- Input and output files must be on CKD direct access devices unless the ADDROUT parameter is also specified. If ADDROUT is specified, output can be on any device type, but the keys will not be written in count-key-data format. The keys will be included in the output record if all or part of the key is specified as a control field.
- Keys are treated as the first n bytes of the input records. Replace n with the number of bytes of the key field.
- Every record must have one key.
- Keys may extend up to 255 bytes, but all keys must be the same length.
- Key fields must be included in the length values of the RECORD statement. The l_1 value must be specified; l_2 to l_5 may be calculated by default. Refer to Table 22 on page 2.118 under the ADDROUT parameter to determine how to calculate these values.
- Input records must be fixed-length and unblocked. If the input records are variable-length and/or blocked, write an E15 exit routine to unblock them and/or restructure them to fixed-length so that the KEYLEN parameter can be used.
- If the BLKSIZE parameter is specified on the INPFIL and OUTFIL statements, the BLKSIZE value must be the length of one record plus the length of the key. If these parameters are not specified, the correct value will be calculated by default.

- Key length must be taken into account when specifying the positions of control fields.
- The entire key, or part of it, can be used as a sort/merge control field.

LABEL Parameter (Optional)

The LABEL parameter specifies what types of labels the files have:

S - Standard labels (the default)

N - Nonstandard labels

U - Unlabeled

The files must be described in order (output, input, work) with the output files and the input files listed in sequence. Output files and input files can have different types of labels but work files must all have the same label type. Since all disk files must be labeled, U is ignored if specified and the default (S) is used instead.

An 'S' designation can be omitted for a standard labeled file within a series, but a comma must be inserted to represent the missing specification. If the files at the end of the series all have standard labels, it is not necessary to insert commas for them. Simply close the parenthesis after the last N or U value; for example, LABEL=(S,N,N,U).

Be sure that the number of specifications, either explicit or by default, equals the total number of output, input and work files.

If any information is added to a standard label, it is no longer considered standard and must be coded as 'N'. Any files with 'N' labels must be opened and closed by an exit program designed to process labels. (Refer to “Chapter 8. User Exit Programs”.) Standard labeled and unlabeled files will be processed by SyncSort.

LOCALE Parameter (Optional)

LOCALE controls cultural environment processing, allowing you to choose an alternate set of collating rules based on a specified national language.

Figure 80. LOCALE Parameter Format

For SORT/MERGE processing, the alternate collating applies to character (CH) fields. For INCLUDE/OMIT comparison processing, the alternate collating applies to character fields and to character constants and hexadecimal constants compared to character fields.

SyncSort employs the callable services of IBM's Language Environment for VSE (LE/VSE) 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.

Notes:

1. Make sure the LIBDEF chain gives SyncSort access to the library that contains the loadable LOCALE routines. For the IBM supplied LOCALEs, these are the dynamically loadable routines in the IBM LE/VSE library. For more information, see the IBM publication *LE/VSE Customization Guide*.

2. If LOCALE processing is used for fields specified in a SORT or MERGE control statement with variable-length records, then SyncSort will 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. LOCALE processing requires additional GETVIS storage to support the use of the IBM LE/VSE facilities. For those jobs that use LOCALE, a total partition size of at least 5M bytes is recommended to accommodate the storage needs of LE/VSE.

5. LOCALE cannot be used with CHALT, VLSHRT, or IGNRL4.

6. LOCALE processing of fields may greatly expand the size of a field.

NOERASE Parameter (Optional)

The NOERASE parameter is ignored.

NOINC Parameter (Optional)

The NOINC parameter instructs SyncSort not to perform an incore sort and is equivalent to INCOR=OFF.

NOTPMK Parameter (Optional)

The NOTPMK parameter specifies that a tape mark not be written before the first record of each volume of an unlabeled tape output file. The default is to write tape marks on each volume of an unlabeled tape output file. The NOTPMK parameter can also be specified on the OUTFIL control statement.

This parameter is ignored for disk files or labeled tape files.

NRECS Parameter (Optional)

The NRECS parameter specifies the number of records to be sorted or merged. These will be the first 'n' records after E15, INCLUDE/OMIT, and SKIPREC processing, if specified, have completed.

The NRECS parameter is identical to the STOPAFT parameter.

NRECOUT Parameter (Optional)

The NRECOUT parameter overrides the NRECOUT installation option. The NRECOUT option indicates how SyncSort will proceed when it does not write any output records.

Figure 81. NRECOUT Parameter Format

RC0 issues a return code of 0 and continues processing.

RC4 issues a return code of 4 and continues processing.

RC16 issues a return code of 16 and terminates processing.

If more than one return code is generated by SyncSort, the highest is issued.

NRECOUT is ignored when OUTFIL EXIT is indicated.

OVFLO Parameter (Optional)

The OVFLO parameter overrides the OVFLO installation option. OVFLO indicates how SyncSort will proceed when BI, FI, PD, or ZD fields overflow during DUPKEYS or SUM processing.

Figure 82. OVFLO Parameter Format

RC0 causes SyncSort to continue processing if overflow is encountered and to issue message WER207 with return code 0. DUPKEYS or SUM processing continues. The record pair that caused the overflow condition is not summed and the records are not deleted.

RC4 is identical to RC0 except that SyncSort issues return code 4.

RC16 causes SyncSort to terminate when an overflow condition is encountered and to issue message WER207 with return code 16.

If more than one return code is generated by SyncSort, the highest is issued.

PRINT Parameter (Optional)

The PRINT parameter determines which sort/merge messages are displayed.

PRINT/PRINT=ALL All sort/merge messages are displayed, including end of phase, end of SyncSort program and error messages.

PRINT=NONE No messages are printed on the printer. Critical messages, however, appear on the console.

PRINT=CRITICAL Only critical messages (A-level severity) are displayed. These messages will appear on the printer and/or console and will indicate premature termination of the SyncSort program.

PRINT=CRITPLUS Displays the same messages as CRITICAL, plus the WER227 and WER228 messages.

PRINT=ALL is the default for a JCL-initiated sort/merge. PRINT=CRITICAL is the default for an invoked sort/merge.

Specify the OPTION statement as the first control statement if you want PRINT=CRITICAL, PRINT=CRITPLUS, or PRINT=NONE to suppress the printing of control card images.

OPTION

ROUTE Parameter (Optional)

The ROUTE parameter directs the routing of messages. ROUTE=LST directs all messages to the printer (SYSLST). If the JCL has assigned a console to the application, critical messages will also appear on the console. ROUTE=LOG directs all messages to the console (SYSLOG). ROUTE=LAL directs all messages both to the printer and to the console. The default is ROUTE=LST for a JCL-initiated application, and ROUTE=LOG for a program-invoked application.

ROUTE=nnn directs all messages to the disk device whose logical unit is specified by nnn, which can be any value from 000 to 221. ROUTE=nnn is accepted only when SyncSort is invoked by another program. If ROUTE=nnn is specified for a JCL-initiated sort/merge, it is ignored and the default (ROUTE=LST) is used. Specifying ROUTE=nnn allows the user to separate SyncSort's messages from the report being produced by the invoking program.

When ROUTE=nnn is specified, the following statements must be included in the JCL of the program that is invoking the sort/merge.

```
// DLBL SYSLST,'file-id'...
// EXTENT SYSnnn,...
```

Figure 83. JCL Statements for the ROUTE=nnn Parameter

The 'nnn' specified in the EXTENT statement must correspond to the logical unit number specified in the ROUTE=nnn parameter.

RPS Parameter (Optional)

The RPS parameter determines whether SyncSort for z/VSE uses rotational position sensing, an operating system facility that frees the channel during part of an I/O operation. If RPS is active on the system, RPS=ON causes SyncSort to utilize the feature and RPS=OFF causes SyncSort to ignore the feature. RPS=OFF is the default.

The use of RPS can enhance system performance by improving throughput.

SAMESDS Parameter (Optional)

The SAMESDS parameter is ignored.

SKIPREC Parameter (Optional)

The SKIPREC parameter instructs SyncSort to skip a decimal number of 'n' records before the input file is sorted or copied. The input bypasses the skipped records before E15 and INCLUDE/OMIT processing, if specified, take place.

The SKIPREC parameter can be specified for a merge only when FIELDS=COPY is specified.

For SKIPREC=n, the maximum value of n is 2G-1 (2147483647).

SORTOUT, SORTIN, SORTWK, and JOINWK Parameters (Optional)

The SORTOUT, SORTIN, SORTWK, and JOINWK parameters override the default symbolic unit numbers that SyncSort uses to logically connect the sort/merge program with input, output, and work devices. The symbolic user-supplied numbers assigned in the JCL ASSGN and EXTENT statements must be specified in the SORTIN, SORTOUT, SORTWK, and JOINWK parameters. However, if the standard default numbers are used, these parameters can be omitted.

Symbolic numbers can be selected from SYSnumbers SYS001 to SYS254. Specify only the numeric part of the SYSnumber. Leading zeros can be omitted.

The following example illustrates how the SORTOUT, SORTIN, SORTWK, and JOINWK parameters can be specified.

```
SORTOUT=11,SORTIN=(16,,),SORTWK=(,,10),JOINWK=(12)
```

Figure 84. Sample SORTOUT, SORTIN, SORTWK, and JOINWK Parameters

This example assumes that SyncSort is reading input and writing output (see Table 45 on page 5.4) and that FILES=3, WORK=3, and JOINWORK=2 have been specified. This example illustrates the following:

- SYS011 has been assigned to the output file, overriding the default value of SYS001.
- SYS016 has been assigned to the first input file, overriding the default of SYS002. The second and third input files represented by commas default to SYS003 and SYS004.
- The default values for SORTWK begin with O+I+1 where O is the number of output files and I is the number of input files. The first two sort work files, represented by commas, default to SYS005 and SYS006. SYS010 has been assigned to the third sort work file, overriding the default of SYS007.
- The default values of JOINWK begin with N+I+W+1 where N is the number of output files, I is the number of input files and W is the number of sortwork files. SYS012 has been assigned to the first sort joinwork file, overriding the default SYS008.

If a direct access work file with multi-extents is being used and DA has been coded in the DLBL statement, it is not necessary to supply symbolic unit numbers for all the extents. Simply provide a single number for the first extent and SyncSort will access all the extents.

OPTION

SPANINC Parameter (Optional)

Figure 85. SPANINC Parameter Format

The SPANINC parameter overrides the SPANINC installation option. SPANINC indicates how SyncSort responds when one or more incomplete spanned records are encountered in a variable spanned input data set.

RC0 causes SyncSort to delete incomplete spanned records it encounters and issue message WER217 with return code 0. Processing continues for complete records.

RC4 is identical to RC0 except that SyncSort issues return code 4.

RC16 causes SyncSort to terminate when incomplete spanned records are encountered and issue message WER217 with return code 16.

If more than one return code is generated by SyncSort, the highest is issued.

SyncSort terminates with return code 16 if a spanned record cannot be assembled, as when segment length is less than 4 bytes. In such cases, SPANINC is not relevant.

STOPAFT Parameter (Optional)

The STOPAFT 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.

The STOPAFT parameter can be specified for a merge only when FIELDS=COPY is specified.

For STOPAFT=n, the maximum value of n is 2G-1 (2147483647).

STORAGE Parameter (Optional)

The STORAGE parameter specifies the amount of main storage to be used and indicates whether SyncSort can perform private CCW translation.

STORAGE=n **STORAGE=nK** **STORAGE=nM**	Define the amount of main storage as a decimal number of bytes, K bytes (K=1024), or M bytes (M=1024x1024).
STORAGE=(n,VIRT) **STORAGE=(nK,VIRT)** **STORAGE=(nM,VIRT)**	Instruct SyncSort not to perform private CCW translations.
STORAGE=(n,NOVIRT) **STORAGE=(nK,NOVIRT)** **STORAGE=(nM,NOVIRT)**	Allow SyncSort to perform a private CCW translation according to the real storage allocated to a partition; equivalent to STORAGE=(,REAL).
STORAGE=(,REAL)	Instructs SyncSort to perform a private CCW translation according to the real storage allocated to a partition. The translation will be done based on the user's real storage definition specified by the ALLOCR or SETPFIX command.
STORAGE=(,AUTO)	Instructs SyncSort to perform a private CCW translation according to what SyncSort considers optimal based on both the *operating system* and *sort performance*.

Table 25. Storage Parameter Options

If the SIZE parameter is specified on the EXEC statement and the STORAGE parameter is specified on the OPTION statement, SyncSort uses the lower figure.

If the figure specified for the STORAGE parameter is larger than the partition size in which the sort/merge is to operate, SyncSort will use whatever space is available within the partition.

In addition to the above rules, the VSCORE parameter (after DSMVSE adjustment)—an internal default that can be changed at installation time—specifies an absolute upper bound to the main storage the sort will be allowed to run in. (See "VSCORE Parameter (Optional)" on page 2.137.)

SyncSort can execute from the GETVIS area. If you want to load SyncSort into the GETVIS area, specify the SIZE parameter on the invoking program's EXEC statement, issue the GETVIS macro, *and* use the STORAGE=n or STORAGE=nK parameter to specify an equivalent amount of storage.

SYMNAMES Parameter (Optional)

The presence of the SYMNAMES parameter indicates that the SyncSort dictionary feature is to be used to map symbolic names in control statements to proper position, length, and format values or constants. See “Chapter 3. Using the SyncSort Dictionary Feature” for information on using a SyncSort dictionary.

If this parameter is used, the OPTION control statement must be placed first, before all other SyncSort control statements.

The SYMNAMES parameter specifies where the dictionary statements are located. It may include a complete library name, sublibrary name, member name, and member type for a library member that contains the dictionary statements. Alternatively, SYMNAMES may include just the member name and the member type. In that case, SyncSort will find the library member by using the library search chain supplied by the SYMNLIB parameter.

Dictionary statements may also be supplied along with SyncSort control statements by specifying SYMNAMES=SYSIPT. In this case, place the dictionary statements immediately after the OPTION control statement (but before all other non-dictionary SyncSort control statements). Use a statement that starts with /* in column 1 and 2 to indicate the end of the dictionary statements. SYMNAMES=SYSIPT is not valid when SyncSort is invoked from within a program.

SYMNLIB Parameter (Optional)

The SYMNLIB parameter specifies a set of library.sublibrary pairs that constitute a library search chain used in library searches when the SYMNAMES parameter only specifies a member name and member type. As many as fourteen library/sublibrary pairs may be specified. See “Chapter 3. Using the SyncSort Dictionary Feature” for information on using a SyncSort dictionary.

The SYMNLIB parameter overrides the installation default value of SYMNLIB.

SYMNOUT Parameter (Optional)

The SYMNOUT parameter specifies where the dictionary listing is to be printed. The listing contains the original dictionary statements and the content of the SyncSort dictionary derived from them. The listing should be reviewed and verified during initial setup or after a change was made to a dictionary.

- SYMNOUT=LST directs the listing to the printer (SYSLST).
- SYMNOUT=LOG directs the listing to the console (SYSLOG).
- SYMNOUT=LAL sends the listing to both the printer and the console.
- SYMNOUT=NONE suppresses the listing.

In all cases, critical error messages from SyncSort dictionary processing will also appear at the console. See "Chapter 3. Using the SyncSort Dictionary Feature" for information on using a SyncSort dictionary.

The SYMNOUT parameter overrides the installation default value of SYMNOUT.

TP Parameter (Optional)

The TP parameter is ignored.

VERIFY Parameter (Optional)

The VERIFY parameter is ignored.

VSCORE Parameter (Optional)

The VSCORE parameter is an internal default parameter that establishes the upper bound of virtual storage below the 16-megabyte line in which SyncSort will run. Refer to the *SyncSort for z/VSE Installation Guide* for this default value and for instructions on how to change it permanently.

If you want to change the VSCORE value for a single job, specify the VSCORE parameter on the OPTION statement to override the default setting. Specify the amount of main storage to be used as an upper limit as a decimal number of n, nK (K=1024) bytes, or nM bytes.

VSCORET Parameter (Optional)

VSCORET sets the maximum amount of storage below and above the 16-megabyte virtual storage line that SyncSort can use when it is running in a partition that has above the 16-megabyte line virtual storage.

The optional VSCORET parameter is used to override the installation default value of VSCORET. Refer to the *SyncSort for z/VSE Installation Guide* for instructions on how to change the installation default value.

If the value specified for VSCORET is less than the value of VSCORE, SyncSort will automatically raise it to that of VSCORE.

WORKNM Parameter (Optional)

The WORKNM parameter changes the four-character prefix of all SORTWK names from the default value, SORT. For example, if WORKNM=SYNC is specified, all sort work files will be named SYNCWKx instead of SORTWKx.

Note: For a JOIN application, the WORKNM parameter will also change the four character prefix of all the JOINWORK files in the same way.

OPTION

XDUPLAB Parameter (Optional)

The XDUPLAB parameter specifies the label type for the XDUP output file:

S Standard labels (the default)

U Unlabeled

XDUPNM Parameter (Optional)

The XDUPNM parameter overrides the default filename for the XDUP output file, which contains records deleted by DUPKEYS processing. This alternate filename must also be used in the DLBL/TLBL job control statements.

An alternate filename can consist of up to seven alphanumeric characters and the first character must be a letter. The delivered default filename for XDUP output is SORTXDP.

XDUPOUT Parameter (Optional)

The XDUPOUT parameter overrides the default symbolic logic unit number that SyncSort uses to connect the sort/merge program to the device on which the XDUP output will be written. This alternate symbolic logic unit must also be used in the ASSIGN and EXTENT job control statements for the XDUP output file.

The alternate symbolic logic unit number must be a number from 1 to 254 that represents SYSnumbers SYS001 to SYS254. The available SYSnumbers are limited by the number of logical units generated in your VSE Supervisor. Specify only the numeric part of the SYS-number. Leading zeros can be omitted.

The delivered default symbolic logic unit number for XDUP output is N+I+W+1, where N is the number of output files not including XDUP output, I is the number of input files, and W is the number of sort work files.

For example, if a sort job has one output file, one input file, one sort work file, and specifies the XDUP parameter in the DUPKEYS statement, then the default symbolic logic unit numbers for the output file, input file, sort work file, and the XDUP output file, are SYS001, SYS002, SYS003, and SYS004, respectively.

XSUMLAB Parameter (Optional)

The XSUMLAB parameter specifies the label type for the XSUM output file:

S Standard labels (the default)

U Unlabeled

XSUMNM Parameter (Optional)

The XSUMNM parameter overrides the default filename for the XSUM output file, which contains records deleted by SUM processing. This alternate filename must also be used in the DLBL/TLBL job control statements.

An alternate filename can consist of up to seven alphanumeric characters and the first character must be a letter. The delivered default filename for XSUM output is SORTXSM.

XSUMOUT Parameter (Optional)

The XSUMOUT parameter overrides the default symbolic logic unit number that SyncSort uses to connect the sort/merge program to the device on which the XSUM output will be written. This alternate symbolic logic unit must also be used in the ASSIGN and EXTENT job control statements for the XSUM output file.

The alternate symbolic logic unit number must be a number from 1 to 254 that represents SYSnumbers SYS001 to SYS254. The available SYSnumbers are limited by the number of logical units generated in your VSE Supervisor. Specify only the numeric part of the SYSnumber. Leading zeros can be omitted.

The delivered default symbolic logic unit number for XSUM output is N+I+W+1, where N is the number of output files not including XSUM output, I is the number of input files, and W is the number of sort work files.

For example, if a sort job has one output file, one input file, one sort work file, and specifies the XSUM parameter in the SUM statement, then the default symbolic logic unit numbers for the output file, input file, sort work file, and the XSUM output file, are SYS001, SYS002, SYS003, and SYS004, respectively.

ZDPRINT/NZDPRINT Parameter (Optional)

Figure 86. ZDPRINT/NZDPRINT Parameter Format

This option specifies if positive ZD summation results from the SUM or DUPKEYS control statement are to be converted to printable numbers. ZDPRINT enables conversion to printable format. NZDPRINT, the default, prevents the conversion.

This option determines whether the sign byte of a positive ZD field summed by the SUM or DUPKEYS control statement 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.

Sample OPTION Control Statement

```
OPTION PRINT=ALL,ROUTE=LST,STORAGE=512K,STOPAFT=1000
```

Figure 87. Sample OPTION Control Statement

This sample OPTION statement specifies the following:

- All sort messages are to be printed.
- Messages are to be routed to the printer. If a console has been assigned to the application, critical messages will also appear on the console.
- The amount of memory to be used by SyncSort for z/VSE is 512K.
- Only 1000 records are to be sorted or merged.

OUTFIL Control Statement

The OUTFIL control statement describes the output file or files. It is required to accomplish the following tasks:

- Create multiple output files
- Use the SortWriter facility
- Reformat records after E35 processing
- Convert variable-length records to fixed-length, or fixed-length records to variable-length.

The OUTFIL parameters associated with each of these tasks are listed below:

- Creating multiple output files: FILES, INCLUDE/OMIT, OUTREC
- Using the SortWriter facility: HEADER1, HEADER2, LINES, TRAILER1, TRAILER2, SECTIONS, NODETAIL
- Reformatting records: OUTREC, CONVERT, FTOV, VTOF, BUILD, IFTHEN, IFOUTLEN, OVERLAY

In addition, the OUTFIL statement is required with VSAM output, in which case the ESDS, KSDS or RRDS parameter must be specified.

The format for the OUTFIL statement is illustrated below.

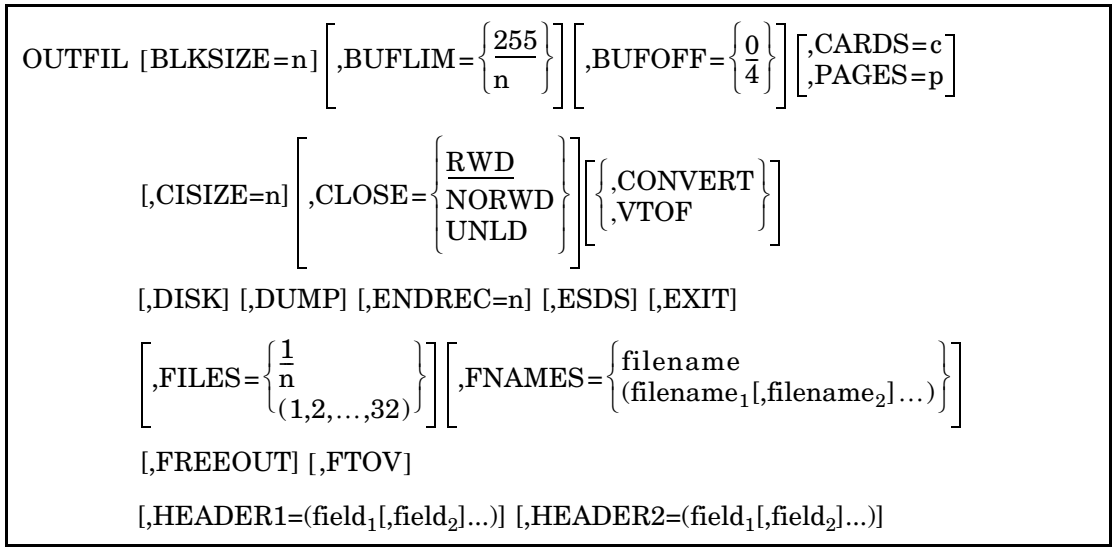


Figure 88. (Page 1 of 2) OUTFIL Control Statement Format

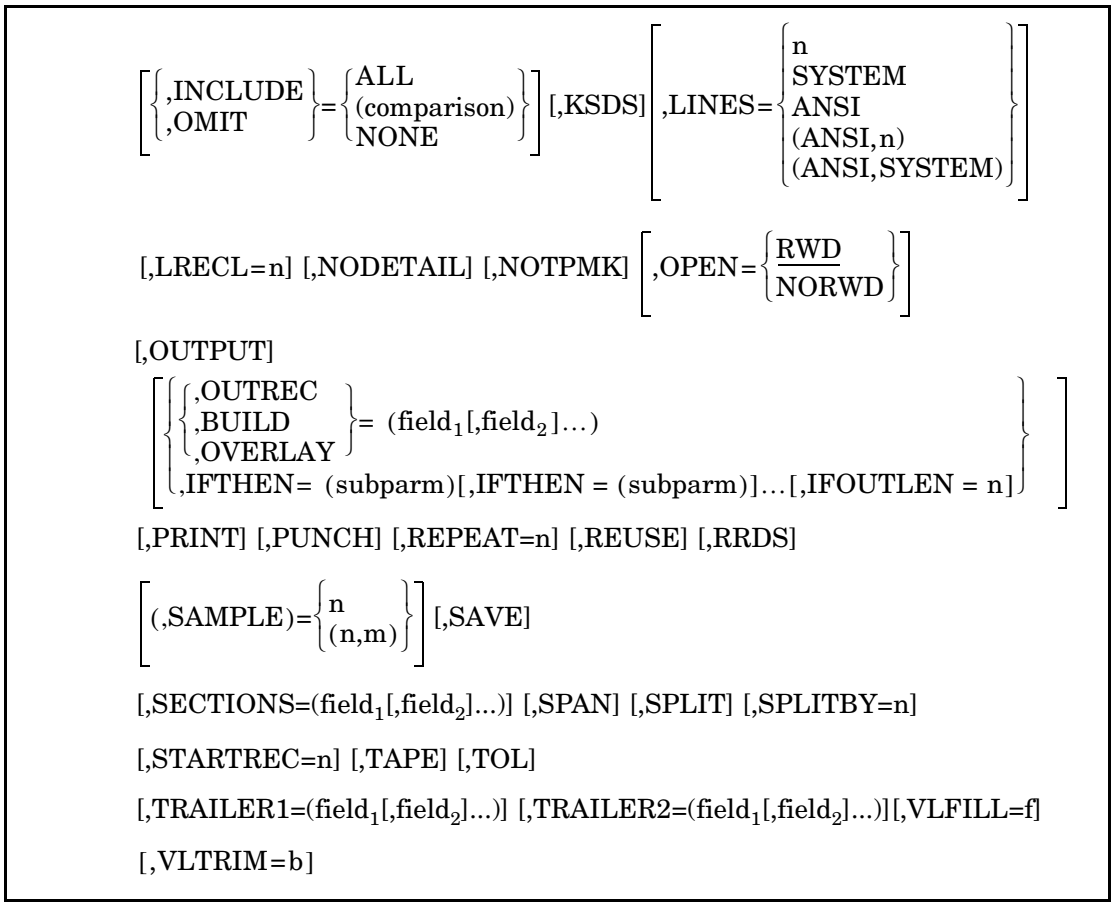


Figure 88. (Page 2 of 2) OUTFIL Control Statement Format

For more information on the OUTFIL processing sequence, see "OUTFIL Processing Sequence" on page 2.21.

The Multiple Output Capability

Use the OUTFIL control statement to create multiple files without making multiple passes through the input data.

- Each output file can be directed to the same output device, or each file can be directed to a different output device.
- Each output file can contain the same records, or each file can contain different records.
- The records in each output file can be identically formatted, or the records in each file can be uniquely formatted.

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.
- 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.

BLKSIZE Parameter (Optional)

The BLKSIZE parameter specifies the blocksize of the output records.

If the BLKSIZE parameter is omitted, either by leaving it out of the OUTFIL statement or omitting the OUTFIL statement itself, SyncSort will generate unblocked records.

Do not specify the BLKSIZE parameter for VSAM output.

Follow these guidelines in specifying the BLKSIZE value:

fixed-length EBCDIC and ASCII records:
BLKSIZE must be a multiple of the fixed record length (l_3).

variable-length EBCDIC records:
BLKSIZE must be greater than or equal to the length of the longest record (l_3, which includes the Record Descriptor Word) *plus* 4 bytes (for the Block Descriptor Word).

variable-length ASCII records:
BLKSIZE must be greater than or equal to the length of the longest record (l_3, which includes the Record Descriptor Word) *plus* the BUFOFF value, if specified.

The l_3 value represents the maximum output record length after E35 and OUTREC processing.

See Table 2 on page 2.23 for the maximum BLKSIZE values allowed for different output devices.

BUFLIM Parameter (Optional)

The BUFLIM parameter places an upper limit of 'n' on the number of output buffers SyncSort can create. Specify n as a decimal number between 1 and 255. The delivered default is 255.

In the vast majority of cases, SyncSort for z/VSE needs no external assistance in choosing the number of buffers. Therefore, avoid specifying this parameter because using it inappropriately may seriously degrade performance.

BUFOFF Parameter (Optional)

The BUFOFF parameter can be specified when TYPE=D is specified on the RECORD control statement to indicate variable-length ASCII data. For variable-length ASCII data, the 'n' value can be 0 *or* 4. If the BUFOFF parameter is not specified, it defaults to 0. For fixed-length ASCII data, this parameter should either be omitted or, if specified, the n value must be 0.

For variable-length data only, when BUFOFF=4 is specified, the 4-digit ASCII blocksize is inserted at the beginning of each block.

When BUFOFF=n is specified, the BLKSIZE value must include the BUFOFF value.

BUILD Parameter (Optional)

The BUILD parameter is identical to the OUTREC parameter of the OUTFIL Control Statement. See “OUTREC Parameter (Optional)” on page 2.158 for a description of this parameter.

CARDS/PAGES Parameter (Optional)

The CARDS parameter sets an upper bound, 'c', on the number of cards to be punched. The PAGES parameter sets an upper bound, 'p', on the number of *logical* pages (not physical pages) to be printed. Specify only one of these two parameters. The choice must reflect the type of device to which the output file has been assigned.

The value specified for CARDS cannot exceed 16,777,215. The delivered default values of 32,767 are assigned only if the PRINT or PUNCH parameters are explicitly specified in the OUTFIL control statement. Specifying 0 for either the CARDS or PAGES parameter causes the sort/merge to terminate abnormally if there is any output.

To use the PAGES parameter to limit the number of physical pages printed, define the logical page using LINES=n with an 'n' value no greater than the line capacity of the paper used.

CISIZE Parameter (Optional)

The CISIZE parameter can be used to override the default control interval size selected by VSAM for implicitly-defined VSAM-managed SAM output files. For other types of output files, this parameter is ignored.

Specify n as a decimal number between 512 and 32768. The actual control interval size of the file created by VSAM will be rounded up to be a multiple of 512 (or 2048), which is larger than the record length (for unblocked files) or the blocksize (for blocked files).

CLOSE Parameter (Optional)

Specified for tape output files only, the CLOSE parameter indicates what action should be taken at end of file.

CLOSE=RWD, the default, specifies that all tape output volumes be rewound at end of file. CLOSE=NORWD specifies that output volumes should not be rewound. CLOSE=UNLD specifies that all output volumes be rewound and unloaded.

The CLOSE parameter is ignored if the output files are on direct access devices.

CONVERT/VTOF Parameter (Optional)

The CONVERT parameter is used to convert variable-length records to fixed-length. In order to use CONVERT, OUTREC, BUILD, OVERLAY, or IFTHEN *must* also be specified as an OUTFIL parameter. In addition, the following control parameters may *not* be specified when CONVERT is used:

- OPTION ADDROUT
- RECORD TYPE=VS, VBS, F or D
- OUTFIL SPAN or EXIT
- MERGE FIELDS=COMPARE
- INPFIL DATA=A

The record length and padding characters of the resulting fixed-length output are determined by the OUTREC (or BUILD, OVERLAY, or IFTHEN) parameter and the presence/absence of the LRECL parameter on the OUTFIL control statement. The following rules are followed:

- If LRECL=n is specified, the output record length is n. Otherwise, the output record length is the sum of the field lengths in the OUTREC parameter list.
- Data records that are shorter than the sum of the field lengths in the OUTREC parameter list are padded to that length with binary zeros; report header and trailer records are padded with blanks. If this length is less than the length specified by the LRECL parameter, additional padding of records is done with blank characters.

When CONVERT is used, all OUTREC fields specifying a position value *must* also specify a length value.

Bytes 1 through 4 of the input record contain the RDW of the record. This field does not need to be copied into the OUTRECed output record when using CONVERT.

DISK Parameter (Optional)

The DISK parameter specifies that the output file be stored on a disk device. The DISK parameter is required if report writing parameters are specified and the output file(s) are to be directed to a disk device.

DUMP Parameter (Optional)

The DUMP parameter specifies that SYSLST output be printed in both character and hexadecimal format.

ENDREC Parameter (Optional)

The ENDREC parameter specifies the last record that will be processed for this output file. The "n"th record sorted will be the last record eligible to be written to the output file. For ENDREC=n, the maximum value of n is 2G-1 (2147483647).

The count of sorted records is done before OUTFIL INCLUDE/OMIT processing, if specified, takes place. (For example, if the first 100 sorted records are to be deleted by INCLUDE/OMIT, the next sorted record is still counted as record number 101. If ENDREC=90, then NO records would be written to the output file.)

If STARTREC is also specified for this output file, the ENDREC value must be greater than or equal to the STARTREC value.

ESDS Parameter (Optional)

Specify the ESDS parameter when the VSAM output data set is to be entry-sequenced. (Entry-sequenced data sets are the VSAM equivalent of SAM data sets.) This parameter is required when creating an entry-sequenced data set. Do not specify the BLKSIZE parameter for an ESDS data set.

When the ESDS parameter is specified, all other OUTFIL parameters that do not relate to VSAM are ignored except for the EXIT parameter, which will override the ESDS parameter.

EXIT Parameter (Optional)

The EXIT parameter specifies that an E35 exit will process the output file(s), including label processing, volume positioning, file opening and closing and handling of the output. Specify exit E35 in the MODS control statement.

If the EXIT parameter is specified, all other OUTFIL parameters are ignored. Parameters specified before the EXIT parameter are read, checked and flagged for errors, but no action is taken.

FILES Parameter (Optional)

The FILES parameter connects the OUTFIL statement with one or more output files. This parameter is required when creating multiple output files.

The format of the FILES parameter is illustrated below:

Figure 89. FILES Parameter Format Chart

Use the numbers 1 through 32 to represent the output file(s). The number 1 designates the output file as SORTOUT; the numbers 2 through 9 designate the output files SORTOF2,...,SORTOF9, and 10 to 32 designate the output files as SORTO10,...SORTO32, respectively. If the FILES parameter is not specified, the default is FILES=1 and the entire output is directed to SORTOUT.

The FILES value cannot exceed the FILESOUT specification on the SORT or MERGE control statement.

When two or more output files have identical OUTFIL specifications (e.g., identical HEADER1s, TRAILER2s, etc.), only one OUTFIL statement is required. If, however, the files have different OUTFIL specifications, a separate OUTFIL statement is required for each file.

FNAMES Parameter (Optional)

The FNAMES parameter connects the OUTFIL statement with one or more output files. Note that FNAMES and FILES are mutually exclusive parameters. If both FNAMES and FILES parameters are specified for the same output file, it will cause a syntax error.

The format of the FNAMES parameter is illustrated below:

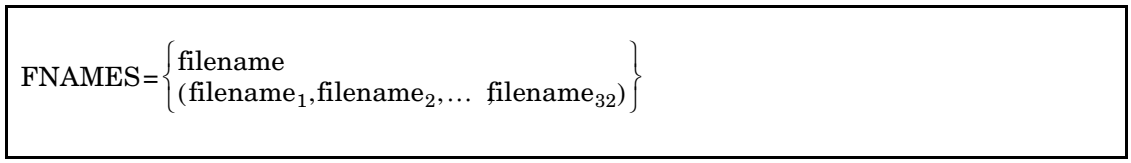


Figure 90. FNAMES Parameter Format Chart

Use FNAMES to specify a filename for an output file. The name can be up to 7 characters long. A DLBL/TLBL statement must be present for this filename. The file names specified with the FNAMES parameter must be either the default filenames (SORTOUT, SORTOF2,...) or a filename specified using the FILNM parameter of the OPTION statement.

FREEOUT Parameter (Optional)

The FREEOUT parameter is ignored.

FTOV Parameter (Optional)

The FTOV parameter converts fixed-length input records to variable-length output records. FTOV can be used either with or 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. The maximum record length is passed from INPFIL.

FTOV can be used with the VLTRIM parameter to delete pad bytes from the end of a record.

The output record format produced by OUTFIL FTOV depends on the BLKSIZE parameter except for VSAM output.

When the BLKSIZE parameter is not specified for the OUTFIL FTOV, SyncSort will generate unblocked output records.

The TYPE for the RECORD control statement must have a fixed-length record format (for example, TYPE=F). In addition, the following control parameters may not be specified when OUTFIL FTOV is used:

- OUTFIL EXIT
- RECORD TYPE=V, D, VS, VBS
- MERGE FIELDS=COMPARE
- OUTFIL VTOF, CONVERT

Note: FTOV cannot be used with CONVERT or VTOF. 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.

FTOV With BLKSIZE Parameter

When the BLKSIZE parameter is specified for the OUTFIL FTOV, SyncSort will generate an output record format of VB or VBS. Make sure the BLKSIZE specified is big enough to contain the largest variable-length output record to be produced.

Note: Do not specify the BLKSIZE parameter for VSAM output.

FTOV With 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. The maximum record length is the reformatted OUTREC record length plus 4.

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 "SPLIT Parameter (Optional)" on page 2.161 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).

Figure 91 illustrates the format for HEADERs. The field entries represent the subparameters that can be specified for each HEADER.

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

Figure 91. HEADER Parameter Format Chart

Figure 92 illustrates and defines the available subparameters. Each subparameter constitutes a separate field of the HEADER.

```
                 [n] X
                 [n] C'literal string'
                 [n] 'literal string'
                 [n] /
                 p,l
                 &DATE=(abcd)
                 DATE=(abcd)
[c]              &DATENS=(abc)
                 DATENS=(abc)
                 &TIME=(nnc)
                 TIME=(nnc)
                 &TIMENS=(nn)
                 TIMENS=(nn)
                 &PAGE
                 PAGE
```

Figure 92. HEADER Subparameters Format Chart

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 (1-255) 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 255 spaces, two or more nX values should be specified.

[C]'literal string' Use the C'literal string' or 'literal string' subparameter to define a literal string. Specify the number of repetitions by coding n immediately before it.

/ Use the / subparameter to indicate the end of a line, force a carriage return and separate text lines of a header. Multiple slashes (coded / / ... / 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,l subparameter 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) must be a character or alphanumeric string or a number in zoned decimal 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 header and blanks will be substituted for the missing bytes.

&DATE=(abcd) The &DATE or DATE subparameter specifies the current generated system date and requires 8 bytes. abcd indicates that you arrange the year, month and day in any order. d is the separator character. Use M to denote month, D to denote day, and Y to denote the last two characters of the year. You can specify 4 as an alternative to Y if you want the year expressed in four digits (1996 as opposed to 96).

For example, if the current date is 4/24/96 and &DATE=(DMY.), the output date is 24.04.96. With the same example, if &DATE=(4MD-), the output date is 1996-04-24.

The date format represented by &DATE may also be set by the DATE parameter on the OPTION statement as follows: ADATE signifies American date format (mm/dd/yy); EDATE signifies European date format (dd/mm/yy); IDATE signifies ISO date format (yy/mm/dd). If omitted, system generated date (STD) is the default.

&DATENS=(abc) Specify &DATENS or DATENS when no separator is desired.

&TIME=(nnc) The &TIME or TIME subparameter specifies the current step start time, which is the time that the current execution of SyncSort began.

nn can be either 12 or 24. 24 specifies twenty-four hour clock with the format hhcmmcss and requires 8 bytes. 12 specifies twelve hour clock with the format hhcmmcss xx. *xx* can be either am or pm. *c* denotes the separation character. When nn=12, &TIME requires 11 bytes.

&TIMENS=(nn) Specify &TIMENS or TIMENS when no separator is desired.

&PAGE The &PAGE or 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 for /, where commas are optional.
- Enclose literals in single quotes.
- Specify blank fields of n bytes as nX.
- Short headings (i.e., 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, specify the LRECL parameter to expand the output record length so that the output record length is at least as long as the longest header. (See "LRECL Parameter (Optional)" on page 2.157.)

IFOUTLEN Parameter (Optional)

The IFOUTLEN parameter overrides the maximum record length, which is automatically set by the IFTHEN parameter, and changes it to a specified value. The IFOUTLEN parameter may only be used in conjunction with the IFTHEN parameter.

The IFOUTLEN parameter automatically makes the following changes to the record to match the new length. A record longer than n is truncated to n. A fixed-length record shorter than n is padded with blanks to reach a length of n. For fixed-length blocked output records, the blocksize must be a multiple of n.

IFTHEN Parameter (Optional)

The IFTHEN parameter uses conditional logic which enables you to reformat records based on specified criteria. Multiple IFTHEN parameters may be specified within the same control statement and are processed sequentially. See "IFTHEN Parameter (Optional)" on page 2.205 for a complete description of this parameter.

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 that 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:

Figure 93. INCLUDE/OMIT Parameter Format

See "INCLUDE/OMIT Control Statement" on page 2.39 for the detailed format of a comparison. The FORMAT=f parameter, which is permitted on the INCLUDE/OMIT control statement, is not permitted on 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 records are to be included in the output file(s), for example, when running a test, 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.

KSDS Parameter (Optional)

The KSDS parameter specifies that the VSAM output data set is to be key-sequenced. The records must be sorted into ascending key sequence on the primary VSAM key. (Key-sequenced data sets are the VSAM equivalent of ISAM data sets.) This parameter is *required* when creating a key-sequenced output data set.

Do not specify the BLKSIZE parameter for a KSDS data set.

When the KSDS parameter is specified, all other OUTFIL parameters that do not relate to VSAM are ignored except for the EXIT parameter, which will override the KSDS parameter.

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/VSE.
- 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.

The format of the LINES parameter is illustrated below:

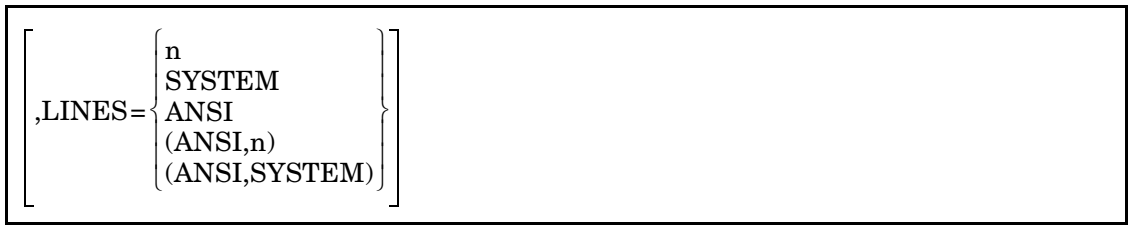


Figure 94. LINES Parameter Format Chart

LINES=n

If LINES=n is specified, paging is automatic and carriage control characters are added to the beginning of each record by SyncSort. Specify 'n' as a value from 0 to 255. If the output file is assigned to a printer, the default is LINES=60. If the output file is assigned to a card punch, the default is LINES=0. LINES=0 is allowed only if the output file is assigned to a card punch. This specification generates all HEADER1 and HEADER2 cards exactly once before all output records.

When LINES=n is specified, the output record length is increased by 1.

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 will occur after HEADER1. Every nth line after the completion of HEADER1 will signal the start of a new page.
- A HEADER2 entry, if present, will be 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 will appear in major to minor order according to the order in which they are specified.
- If PAGEHEAD is specified, HEADER3 will print 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 in which they are specified.

- 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=SYSTEM

LINES=SYSTEM is similar to LINES=n. If LINES=SYSTEM is specified, the VSE system defined default for the number of lines per page for SYSLST will be used as 'n'.

LINES=ANSI

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

The ANSI control characters that can be used with the LINES=ANSI specification are summarized in Table 26 on page 2.157.

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 is encountered. Because '1' is not a valid control character for a card punch, output punched using LINES=ANSI always consists of exactly one logical page and always punches HEADER2 only once before all data records.

- If the output file is assigned to a printer, 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. To prevent the data record from printing over a text line of HEADER2, the HEADER2 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 that 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.

LINES=(ANSI,SYSTEM)

LINES=(ANSI,SYSTEM) is similar to LINES=(ANSI,n). If LINES=(ANSI,SYSTEM) is specified, the VSE system defined default for the number of lines per page for SYSLST will be used as 'n'.

Valid ANSI Control Characters

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

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

Table 26. ANSI Control Character Table

LRECL Parameter (Optional)

The LRECL parameter lets you specify an output record length different from the length specified in the LENGTH parameter on the RECORD statement. It is, therefore, possible to have one record length for the sort/merge and a different record length for the output record. SyncSort automatically pads the output record with blanks or truncates records on the right to achieve the desired length.

The LRECL parameter must be specified when a header or trailer is longer than the output record (l_3 value). In that case, the LRECL value should equal or exceed the length of the *longest* header or trailer.

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.

NOTPMK Parameter (Optional)

The NOTPMK parameter specifies that a tape mark *not* be written before the first record of each volume of an unlabeled tape output file. The default is to write tape marks on each volume of an unlabeled tape output file. The NOTPMK parameter can also be specified on the OPTION control statement.

This parameter is ignored for disk files or labeled tape files.

OPEN Parameter (Optional)

Specified for tape output files only, the OPEN parameter indicates what action should be taken when the file is opened.

OPEN=RWD, the default, specifies that the first volume of the output file be rewound when the file is opened. OPEN=NORWD specifies that the first volume should not be rewound before data is written. The default is OPEN=RWD.

The OPEN parameter is ignored if the output files are on direct access devices.

OUTPUT Parameter (Optional)

The OUTPUT parameter is ignored by SyncSort for z/VSE.

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 that contain differently formatted records.

When the records in 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 SUM control statement, the OUTREC control statement and an E35 exit but before ANSI control character processing.

The format of the OUTREC parameter is illustrated below.

OUTREC=(field$_1$[,field$_2$]...)

Figure 95. OUTREC Parameter Format

The format of the OUTREC parameter is identical to the format of the OUTREC control statement except that the FIELDS= specification is omitted. Refer to "OUTREC Control Statement" on page 2.174 for a detailed explanation of how to specify this parameter.

In addition, OUTFIL OUTREC accepts one subparameter that cannot be specified on the OUTREC control statement:

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

The / subparameter is most useful for its ability to accommodate records whose lengths exceed the width of the physical page. For an example use of the / subparameter, see "Printing Input Records on Multiple Output Lines" on page 4.31.

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

OVERLAY Parameter (Optional)

The OVERLAY parameter enables you to change particular columns within a record, and add fields to the end of a record, without rebuilding the entire record. When using the OVERLAY parameter you only need to specify the columns you want to change; the rest of the input record remains unchanged. See "OVERLAY Parameter (Optional)" on page 2.209 for a complete description of this parameter.

PRINT Parameter (Optional)

Use the PRINT parameter to specify that the output file be written to a printer. If report writing parameters are specified and another output device type is *not* specified, PRINT is the default. If report writing parameters are *not* specified and the output is to be printed, the PRINT parameter must be specified.

When the output file is to be printed, the SYSnumber for this file (usually SYS001 for SORTOUT) must be assigned to a printer. Do not specify a DLBL or TLBL statement for the printer output file.

PUNCH Parameter (Optional)

Use the PUNCH parameter to specify that the output file be written to a card punch. This parameter is required if the output is to be directed to a card punch.

The SYSnumber for this file (usually SYS001 for SORTOUT) must be assigned to a punch. Do not specify a DLBL or TLBL statement for the punch output file.

REPEAT Parameter (Optional)

The REPEAT=n parameter writes each output record multiple times. n specifies the number of times each OUTFIL output record is written. The minimum value for n is 2.

REPEAT can be used with the OUTFIL OUTREC multiline feature (designated by a / in the OUTREC specification). When this is done, each line will be written n times defined by the OUTREC specifications. All occurrences of the first line will be written followed by all the occurrences of the second, and so on.

The REPEAT parameter cannot be used with LINES, HEADER1, TRAILER1, HEADER2, TRAILER2, SECTIONS, and NODETAIL.

REUSE Parameter (Optional)

Use the REUSE parameter to specify that the output be written over an existing VSAM data set. The VSAM cluster must have been defined (DEFINE CLUSTER) with the REUSE option; otherwise, an error will occur.

The REUSE parameter is ignored for non-VSAM files.

RRDS Parameter (Optional)

Specify the RRDS parameter when the VSAM output data set is to be a relative record data set. (A relative record VSAM data set is essentially a string of fixed-length slots, each of which has a relative record number. This number starts at one and continues up to the maximum number of records that can be stored in the file.) This parameter is required when creating a relative record data set.

Do not specify the BLKSIZE parameter for an RRDS data set.

When the RRDS parameter is specified, all other OUTFIL parameters that do not relate to VSAM are ignored except for the EXIT parameter, which will override the RRDS parameter.

SAMPLE Parameter (Optional)

The SAMPLE=n and SAMPLE=(n,m) parameters allow the selection of a sample of records from an OUTFIL group. A specific interval and number of records in that interval can be specified. The sample process will take place within the range of records specified by STARTREC or ENDREC if they are specified. SAMPLE=n and SAMPLE=(*n*,*m*) are mutually exclusive.

The sample consists of the first m records in every nth interval. *n* specifies the interval size. The minimum value for *n* is 2 (sample every other record).

m specifies the number of records to be processed in each interval. The minimum value for *m* is 1(process the first record in each interval). If m is not specified, 1 is used for *m*. If *m* is specified, it must be less than *n*.

SAVE Parameter (Optional)

Use the SAVE parameter to include all the records that are not included in another OUTFIL group in the data sets for this group. SAVE retains all the input records that are not included in the other groups and would otherwise be lost.

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 record 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 96. Sample OUTFIL Control Statement with SPLIT

For the above example, the first record will be written to the SORTOF1 data set, the second, to SORTOF2, the third, to SORTOF3. The fourth record will be written to SORTOF1 again, and so on in round-robin fashion.

OUTFIL

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 (HEADERn, TRAILERn, SECTIONS, LINES, NODETAIL) cannot be specified on the OUTFIL control statement that defines the output group.

SPLIT and SPLITBY=n are mutually exclusive. SPLITBY=1 is equivalent to SPLIT.

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=(field$_1$[,field$_2$]...)
  Each field is specified as follows:
    p,l[,p,l],subparameter[,subparameter]

Figure 97. SECTIONS Parameter Format Chart

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 statement, the OUTREC statement and an E35 exit, if specified, but before processing by the OUTREC parameter and other report writing parameters of the OUTFIL statement, if specified.

l The length value indicates the length of the field. The length must be an integral number of bytes, with a maximum value of 256.

Multiple p,l pairs of non-contiguous fields can be specified to form a control field for a SECTIONS break.

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

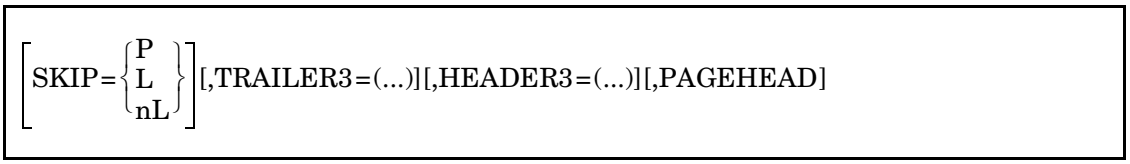


Figure 98. SECTIONS Subparameters Format Chart

SKIP={ **P** | **L** | **nL** }

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. SKIP=P specifies a page break following the completion of a section. SKIP=L means skip one line.

HEADER3=(. . .)

The HEADER3 subparameter specifies a section header or title that will appear at the start of each new section. The formats of the HEADER3 and HEADER1/HEADER2 parameters are identical. (See "HEADER1/HEADER2 Parameters (Optional)" on page 2.149 for details.)

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 (Optional)" on page 2.164 for details.)

PAGEHEAD

The PAGEHEAD subparameter can only be specified in conjunction with the HEADER3 subparameter. The PAGEHEAD subparameter specifies that the HEADER3 appear at the top of each page as well as at the start of each new section.

SPAN Parameter (Optional)

The SPAN parameter indicates that the output files are spanned.

If TYPE=VS or TYPE=VBS is specified on the RECORD control statement, the SPAN parameter does not have to be specified.

SPLITBY Parameter (Optional)

The SPLITBY=n parameter writes groups of records in rotation among multiple output data sets and distributes multiple records at a time among the OUTFIL data sets. *n* specifies the number of records to split by. The minimum value for *n* is 1.

The SPLITBY parameter is similar to SPLIT, but SPLITBY can be used to rotate by a specified number of records rather than by one record, for example, records 1-10 to the first OUTFIL data set, records 11-20 to the second OUTFIL data set, and so on.

For example, if SPLITBY=10 is specified for an OUTFIL group with three data sets:

- The first OUTFIL data set in the group receives records 1-10, 31-40, and so on.
- The second OUTFIL data set in the group receives records 11-20, 41-50, and so on.
- The third OUTFIL data set in the group receives records 21-30, 51-60, and so on.

SPLIT and SPLITBY=n are mutually exclusive. SPLITBY=1 is equivalent to SPLIT.

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

STARTREC Parameter (Optional)

The STARTREC parameter specifies the first record that will be processed for this output file. The "n"th sorted record will be the first record eligible to be written to the output file; the first n-1 records are skipped. For STARTREC=n, the maximum value of n is 2G-1 (2147483647).

The count of sorted records is done before OUTFIL INCLUDE/OMIT processing, if specified, takes place.

TAPE Parameter (Optional)

Use the TAPE parameter to specify that the output file be stored on a tape device. The TAPE parameter is required if report writing parameters are specified and the output file(s) are to be directed to a tape device.

TOL Parameter (Optional)

The TOL parameter specifies that a warning code be accepted when a VSAM output file is opened. If the TOL parameter is not specified, the warning return code is recognized as an error, an error message is generated, and SyncSort for z/VSE terminates.

TRAILER1/TRAILER2 Parameters (Optional)

The SortWriter facility provides three types of trailers:

- TRAILER1, the report trailer

- TRAlLER2, 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 2.162 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 bottom or 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 and SUBCOUNT capabilities at report, page and section levels.

Figure 99 below illustrates the format for TRAILERs. Its field entries represent the subparameters that can be specified for each TRAILER.

Figure 99. TRAILER Parameter Format

Figure 100 illustrates the available TRAILER subparameters. Each subparameter constitutes a separate field of the TRAILER.

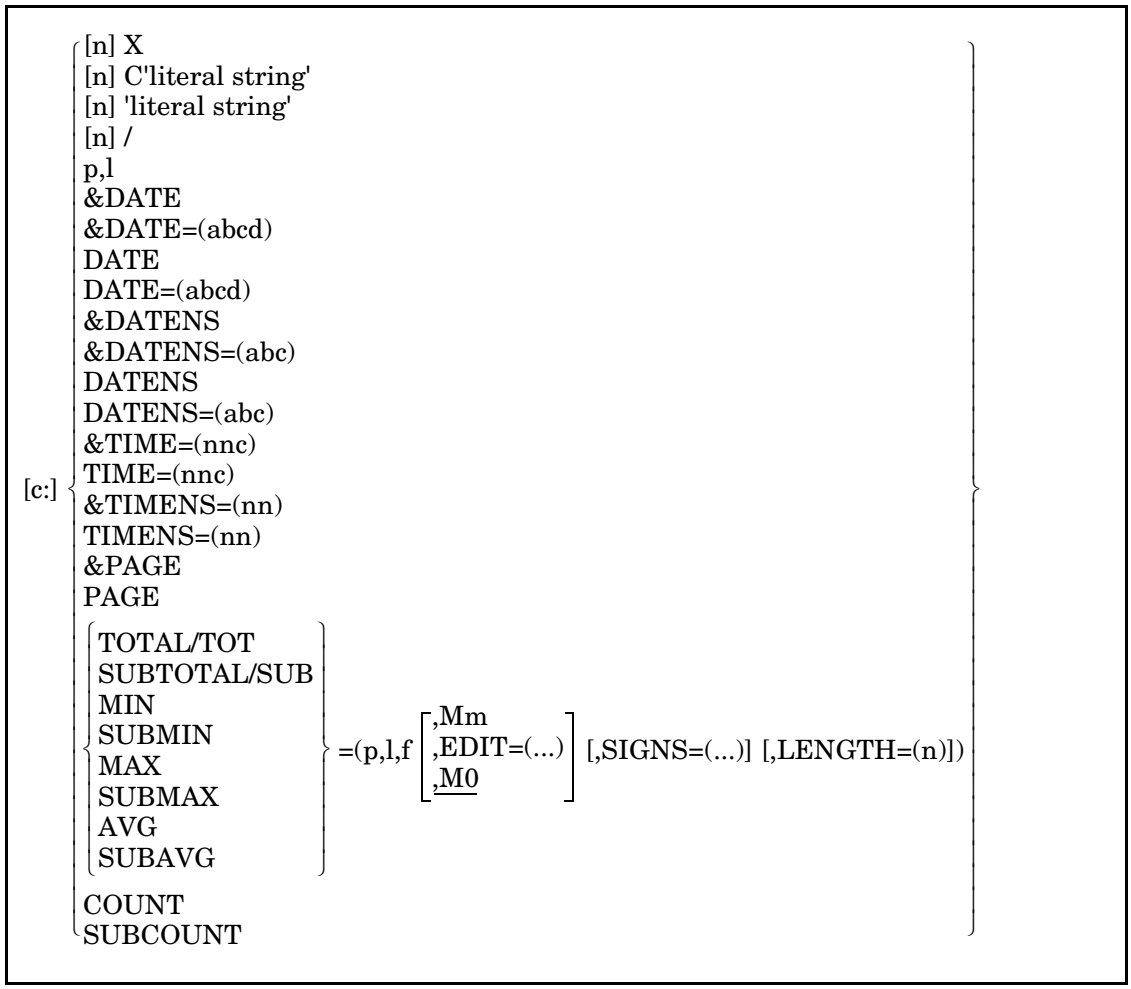


Figure 100. TRAILER Subparameters Format Chart

c: Use the c: subparameter to define the column in which the specified field should begin. The c: subparameter must be specified in ascending order and not cause overlapping fields.

n Used in conjunction with the X, 'literal string', and / subparameters, the n value defines the number (1-255) 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 255 spaces, two or more nX values should be specified.

[C]'literal string'

Use the C'literal string' or 'literal string' subparameter to define a literal string. Specify the number of repetitions by specifying n immediately before it.

/

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,l subparameter to include a field or field(s) 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) must be a character or alphanumeric string or a number in zoned decimal 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=(abcd)

The &DATE or DATE subparameter specifies the current generated system date and requires 8 bytes. abcd indicates that you arrange the year, month and day in any order. d is the separator character. Use M to denote month, D to denote day, and Y to denote the last two characters of the year. You can specify 4 as an alternative to Y if you want the year expressed in four digits (1996 as opposed to 96).

For example, if the current date is 4/24/96 and &DATE=(DMY.), the output date is 24.04.96. With the same example, if &DATE=(4MD-), the output date is 1996-04-24.

The date format represented by &DATE may also be set by the DATE parameter on the OPTION statement as follows: ADATE signifies American date format (mm/dd/yy); EDATE signifies European date format (dd/mm/yy); IDATE signifies ISO date format (yy/mm/dd). If omitted, system generated date (STD) is the default.

&DATENS=(abc) Specify &DATENS or DATENS when no separator is desired.

&TIME=(nnc) The &TIME or TIME subparameter specifies the current step start time, which is the time that the current execution of SyncSort began. The default time format is hh:mm:ss.

nn can be either 12 or 24. 24 specifies twenty-four hour clock with the format hhcmmcss and requires 8 bytes. 12 specifies twelve hour clock with the format hhcmmcss xx and requires 11 bytes. xx can be either am or pm. c denotes the separation character. For example, &TIME=(24-) expresses 7:31 pm as 19-31-00.

&TIMENS=(nn) Specify &TIMENS or TIMENS when no separator is desired.

&PAGE The &PAGE or 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 is 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 (ZD) format.

SUBTOTAL/SUB Use the SUBTOTAL (or SUBTOT or SUB) 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 (ZD) format.

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 field to be totaled.

l Use the l subparameter to indicate the length of the field to be totaled.

f Use the f subparameter to indicate the format of the field to be totaled. Replace f with BI, FI, PD or ZD.

Mm Use the Mm subparameter to indicate that one of the 26 SyncSort-supplied masks (M0-M25) be used to format the totaled field. Replace m with the mask number. (For details, refer to “Mm Subparameter (Editing Masks)” on page 2.202.)

EDIT=(pattern) Use the EDIT=(pattern) subparameter to indicate that a user-provided editing mask be used to format the totaled field. (For details, refer to “EDIT Subparameter” on page 2.200.)

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 “SIGNS Subparameter” on page 2.204.)

LENGTH=(n) Use the LENGTH subparameter to alter the length of a totaled field determined by the edit pattern and the internal field format. (For details, see “LENGTH=n Subparameter” on page 2.201.)

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. This number will be a right-justified, 8-digit field with leading zeros suppressed. The maximum value is 99999999.

SUBCOUNT Use the SUBCOUNT subparameter to obtain a running 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 running count of the total number of data records in the report. In a TRAILER2, it will contain a running count of the number of data records on each page. In a TRAILER3, it will contain a running count of the number of data records in each section. This number will be a right-justified, 8 digit field with leading zeros suppressed. The maximum value is 99999999.

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.
- Short trailers (i.e., 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, specify the LRECL parameter to expand the output record length so that the output record length is at least as long as the longest trailer. (See “LRECL Parameter (Optional)” on page 2.157.)

VLFILL Parameter (Optional)

The VLFILL parameter is used in conjunction with VTOF to specify a fill byte to be used for any short record. By default, spaces will be used. In order to use the VLFILL parameter, OUTREC, BUILD, OVERLAY, or IFTHEN *must* also be specified as an OUTFIL parameter.

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 Record Descriptor Word (RDW), or the ANSI carriage control character.

For the VLTRIM parameter, specify a byte *b* 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.

VTOF Parameter (Optional)

The VTOF parameter is identical to the CONVERT parameter. See "CONVERT/VTOF Parameter (Optional)" on page 2.145 for a description of these parameters.

Sample OUTFIL Control Statements

The following example uses the IFOUTLEN and OVERLAY parameters to lengthen the records.

```
OUTFIL BLKSIZE=800,LRECL=80,IFOUTLEN=100,
     OVERLAY=(9:9,4,PD,SUB,13,4,PD,PD,LENGTH=4,81:9,4,PD)
```

Figure 101. Sample IFOUTLEN and OVERLAY parameters

This OUTFIL control statement refers to an 80-byte record. The OVERLAY parameter subtracts the Payments field (13,4,PD) from the Balance Due field (9,4,PD). This updated Balance Due amount is entered in a new, displayable field at the end of the record. Blanks are filled in after the Balance Due amount to increase the output LRECL to 100 bytes.

The following example uses the IFOUTLEN and OVERLAY parameters to shorten the records.

```
OUTFIL BLKSIZE=800,LRECL=100,IFOUTLEN=80,
     OVERLAY=(9:9,4,PD,SUB,13,4,PD,PD,LENGTH=4,51:9,4,PD)
```

Figure 102. Sample IFOUTLEN and OVERLAY parameters

This OUTFIL control statement refers to an 100-byte record. The OVERLAY parameter subtracts the Payments field (13,4,PD) from the Balance Due field (9,4,PD). This updated Balance Due amount is entered at columns 51-54. The output LRECL is truncated to 80 bytes.

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

```
OUTFIL FILES=1,DISK,OUTREC=(1,20,45,5,60,8),INCLUDE=(21,2,CH,EQ,C'NY')
OUTFIL FILES=2,TAPE,OUTREC=(1,20,60,8),INCLUDE=(21,2,CH,EQ,C'MA')
```

Figure 103. Sample Multiple OUTFIL Control Statements

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

- The first file (SORTOUT) will be written to disk. Its output records contain three fields: the first field begins in byte 1 and is 20 bytes long, the second field begins in byte 45 and is 5 bytes long, and the third field begins in byte 60 and is 8 bytes long. This file will include only those records with 'NY' in bytes 21 and 22.

- The second file (SORTOF2) will be written to tape. Its output records contain two fields: the first field begins in byte 1 and is 20 bytes long, and the second field begins in byte 60 and is 8 bytes long. This file will include only those records with 'MA' in bytes 21 and 22.

In the following example, OUTFIL parameters specify information about a tape output file.

```
OUTFIL TAPE,BLKSIZE=800,CLOSE=UNLD
```

Figure 104. Sample OUTFIL Control Statement

This OUTFIL statement specifies the following:

- The output file is to be written to tape.

- Its blocksize is 800.

- All tape output volumes are to be rewound and unloaded at end of file.

Comprehensive examples illustrating the SortWriter facility and the multiple output capability of the OUTFIL statement are provided in “Chapter 4. How to Use SyncSort Data Utility Features”.

```
INPFIL BLKSIZE=800,LRECL=80
OUTFIL FTOV,VLTRIM=C''
```

Figure 105. Sample OUTFIL Control Statement using FTOV Parameter

The control statements above will cause 80-byte fixed-length input records to be converted to variable-length records, removing trailing blanks from each record and adding an RDW.

```
INPFIL BLKSIZE=2400,LRECL=240
OUTFIL OUTREC=(1,50,200,10),FTOV,VLTRIM=C''
```

Figure 106. Sample Multiple OUTFIL Control Statement using FTOV With OUTREC Parameter

The control statements above will cause the 240-byte fixed-length input records to be converted to variable-length records by using 60 bytes from each input record, removing trailing blanks, and adding an RDW.

OUTREC Control Statement

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

- Delete or repeat segments of the input records.
- Insert character strings between data fields.
- Insert binary zeros.
- Insert record sequence number or date/time of job run.
- Convert numeric data to printable format or to another numeric 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 binary data to printable hexadecimal format, or vice versa.
- Change the order of, or completely redesign, the original input records.
- Select, realign, and reorder data fields.
- Translate the case (upper, lower) of EBCDIC letters.
- Translate characters based on a translation table.
- The BUILD, IFTHEN, IFOUTLEN, and OVERLAY parameters can be used to *conditionally* reformat records or reformat only *selected portions* of records.

The OUTREC parameter of the OUTFIL control statement can also be used to accomplish any of the above tasks.

OUTREC control statement processing takes place after the records have been sorted or merged but before E35 processing, if specified. OUTREC parameter processing takes place after E35 processing.

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

- Use the INREC 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 statement or the OUTREC parameter of the OUTFIL 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 statement when you are creating multiple output files with the same output record.
- Use the OUTREC parameter of the OUTFIL statement when you are creating multiple output files with different output records.
- Use the OUTREC parameter of the OUTFIL statement when an E35 exit processes the records.
- Use the OUTREC parameter of the OUTFIL statement when you specify the TOTAL and/or SUBTOTAL subparameters of the TRAILER parameter so that the data field being totaled can easily be aligned with the total in the trailer.

OUTREC Control Statement Format

The format for the OUTREC control statement is illustrated below.

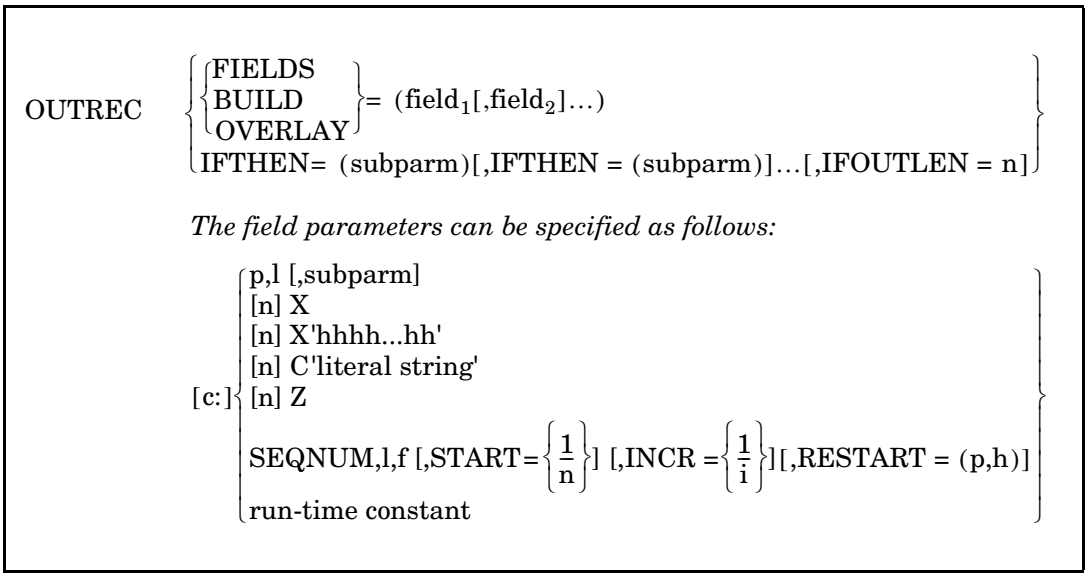


Figure 107. OUTREC Control Statement Format

FIELDS/BUILD Parameter

The FIELDS parameter specifies fields that are to be included in the output record. BUILD is an alias for FIELDS. Fields can be data fields or spaces, hexadecimal digits, literal strings or binary zeros.

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

c: For the description of this term, see "FIELDS Subparameters" on page 2.180.

p The position value indicates the first byte of the field relative to the beginning of the input record after E15, INREC and SUM processing, if specified, have completed. If the OUTREC parameter of the OUTFIL statement is used, the position value refers to the record after E35 processing as well. The field must begin on a byte boundary and can be anywhere in the record.

l The length value indicates the length of the field. The length must be an integral number of bytes.

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 be inserted in the output record.

[n]X Use the nX entry to define a specified number, n, of spaces. The n value may be any number between 1 and 4095 inclusive. The X entry represents spaces 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. For example, 4095X,4095X,63X will provide 8253 spaces.

[n]X'hhhh...hh' Use the nx'hhhh...hh' entry to specify that n copies of hex-digits or hex-digit strings be inserted in the output record. The repetition factor, n, may be any number between 1 and 4095 inclusive.

[n]C'literal string' Use the nC'literal string' entry to specify that n copies of literal characters be inserted in the output record. The repetition factor, n, may be any number between 1 and 4095 inclusive.

[n]Z Use the nZ entry to define a specified number, n, of binary zeros to 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 the number specified for 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, CSF, FS, PD or ZD formats. A starting value and an increment can be specified for the field. In addition, the sequence numbering can be restarted when the value in a specified field changes.

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, CSF, FS, PD or ZD can be specified to create either an unsigned binary field, a floating signed field, a packed decimal field or zoned decimal field.

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.

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.

RESTART Optionally specifies that the sequence numbering is restarted when the value in the field defined by the p,h specification changes. The value of n specified in the START parameter is used to restart the numbering sequence. p represents the position of the first byte of the field. h is the length of the field and can be from 1 to 256 bytes. A binary comparison is performed on the field.

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.

run-time constant Use any of the run-time constant parameters in Table 27 on page 2.179 to insert the date and/or time of the current SyncSort run. The table shows each parameter along with its output format and length.

The leading '&' of each parameter may be omitted.

A 'C' in the output format denotes a character constant output, while a 'P' denotes a packed decimal constant output. Packed

decimal constants contain a positive sign and a leading zero where padding is necessary.

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

The 'xyz' in the **DATE=** and **DATENS=** parameters represent the desired layout of year, month, and day. Use **M** for month (01-12), **D** for day (01-31), and either **Y** for 2-digit year (00-99) or **4** for 4-digit year, replace 'xyz' with any sequence of day, month, and year. The 'c' in **DATE=**parameter is the separator character.

The 'tt' in **TIME=** and **TIMENS=** parameters can be **12** or **24**, indicating a 12-hour or 24-hour time format of output. The 24-hour time format ranges from 00 hour 00 minute 00 second to 23 hour 59 minute 59 second. The 12-hour time format ranges from 12 hour 00 minute 00 second AM to 11 hour 59 minute 59 second PM. The last three characters of the 12-hour time format output are either ' **am**' or ' **pm**'. The 'c' in the **TIME=** parameter is the separator character.

Parameter	Output	Length (Bytes)
&DATE	C'mm/dd/yy'	8
&DATE1	C'yyyymmdd'	8
&DATE1(c)	C'yyyycmmcdd'	10
&DATE1P	P'yyyymmdd'	5
&DATE2	C'yyyymm'	6
&DATE2(c)	C'yyyycmm'	7
&DATE2P	P'yyyymm'	4
&DATE3	C'yyyyddd'	7
&DATE3(c)	C'yyyycddd'	8
&DATE3P	P'yyyyddd'	4
&DATE4	C'yyyy-mm-dd-hh.mm.ss'	19
&DATE=(xyzc)	(see description above table)	8 or 10
&DATENS=(xyz)	(see description above table)	6 or 8
&TIME	C'hh:mm:ss'	8
&TIME1	C'hhmmss'	6
&TIME1(c)	C'hhcmmcss'	8
&TIME1P	P'hhmmss'	4
&TIME2	C'hhmm'	4
&TIME2(c)	C'hhcmm'	5
&TIME2P	P'hhmm'	3
&TIME3	C'hh'	2
&TIME3P	P'hh'	2
&TIME=(ttc)	C'hhcmmcss am'	8 or 11
&TIMENS=(tt)	C'hhmmss am'	6 or 9

Table 27. Run-Time Constants

Specifying the FIELDS Parameter for Variable-Length Records

Observe these rules when you specify the FIELDS parameter for variable-length records:

- Remember to specify 4 bytes for the Record Descriptor Word. 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.
- At least one byte from the fixed portion of the input records must be 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. Make sure the position value does not exceed the minimum input record length. Do not specify any other subparameters after the position value *unless* HEX conversion is also specified. (Refer to "FIELDS Subparameters" on page 2.180 for an explanation of HEX conversion.)
- The nX entry cannot be used to insert spaces before the first field if variable-length records are being used because the Record Descriptor Word must be accounted for.
- The contents of the Record Descriptor Word will be automatically revised by the sort.

FIELDS 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.
- Convert a field to its printable hexadecimal representation.
- Change an input field to a replacement constant if the input field equals a specified search constant.
- Provide minimum and maximum functions and arithmetic operations (add, subtract, multiply, divide) with numeric fields and constants.

Figure 108 illustrates how the FIELDS subparameters should be specified and describes their functions. More detailed descriptions of the EDIT, LENGTH, Mm, and SIGNS subparameters follow this figure in the section "How to Convert Numeric Data" on page 2.192.

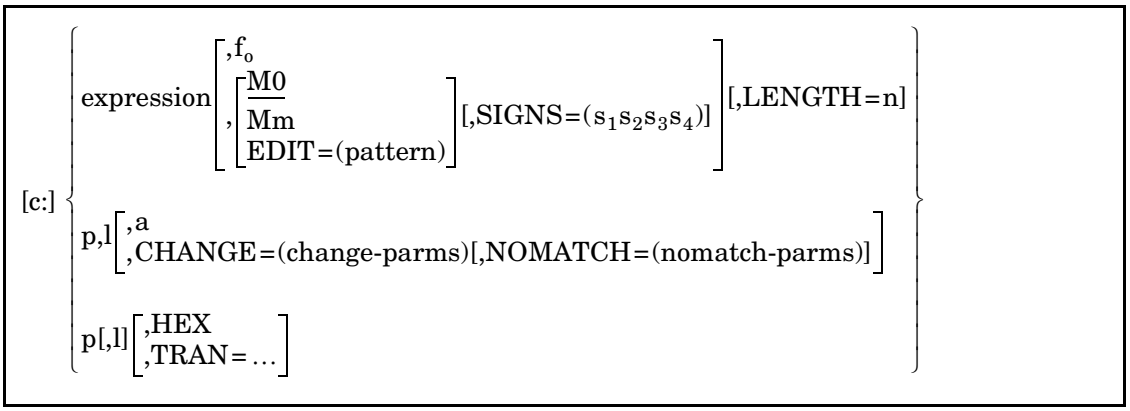


Figure 108. Fields Subparameters Format

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. The c: subparameter must be specified in ascending order and not cause overlapping fields. This subparameter can be specified for *all* types of fields.

The term *expression* represents the following syntax:

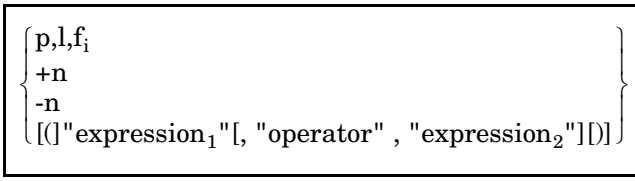


Figure 109. Syntax for expression

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 chart. 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 divide 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. 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

MOD modulus

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, MOD, ADD, SUB). Within the arithmetic operators, multiplication (MUL), division (DIV), and modulus (MOD) are performed before addition (ADD) and subtraction (SUB). Operations within the same precedence level are performed from left to right.

The result of the DIV operation is truncated (rounded down) to an integer. The MOD operation produces an integer remainder with the sign of the dividend.

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, PD0, one of the SMF formats (DT1, DT2, DT3, TM1, TM2, TM3, and TM4), 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. Y2DP and Y2PP can be used as synonyms for Y2ID and Y2IP respectively.

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/FS ... 1-16 inclusive (15 digit limit)

for FI ... 1-4 inclusive
for PD ... 1-8 inclusive
for PD0 ... 2-8 inclusive
for SFF ... 1-44 inclusive (15 digit limit)
for UFF ... 1-44 inclusive (15 digit limit)
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

Note that the above list of date formats (Y2x) does not include full-date formats. For full-date formats, see Table 28 on page 2.185 and "Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC or OUTFIL OUTREC: Full-Date Data" on page 2.196.

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 is to 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 be to 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.

The following table shows the full-date formats:

Full-Date Format	Date 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	dddyy	5
		xxxxyy	mmddyy	6
Y2X	PD	xyy (X'xyys')	qyy	2
		xxxyy (X'xxxyys')	dddyy	3
Y2Y	PD	xxyy (X'0xxyys')	mmyy	3
		xxxxyy (X'0xxxxyys')	mmddyy	4

Note: The following symbols are used in the table:

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

Table 28. Full-Date Formats

f_o Use this parameter 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.

Mm Use the Mm subparameter to indicate that one of the 26 SyncSort-provided editing masks, M0-M25, is to be used. Replace 'm' with the mask number. (See “Mm Subparameter (Editing Masks)” on page 2.202.)

EDIT=(pattern) Use the EDIT subparameter to specify that a user-provided editing mask be used to format the output field. (See “EDIT Subparameter” on page 2.200.)

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. (See “SIGNS Subparameter” on page 2.204.)

LENGTH=n Use the LENGTH subparameter to alter the length determined by the edit pattern and the format of the edited field. (See “LENGTH=n Subparameter” on page 2.201.)

a Use this subparameter to tell SyncSort for z/VSE 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.

HEX Use the HEX subparameter to convert a record field to its printable 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 hex digit will be converted to its printable representation.

Note: In the reformatted record, the converted field will double in length.

TRAN Use this subparameter to translate each byte of the field. 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 specified. The format of the TRAN subparameter is illustrated below:

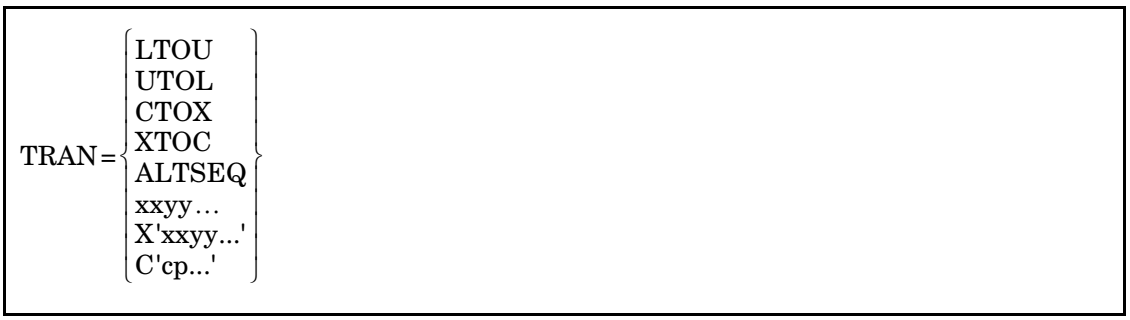


Figure 110. TRAN Subparameters format

LTOU Instruct SyncSort to translate EBCDIC letters in the specified field from lower case to upper case.

UTOL Instruct SyncSort to translate EBCDIC letters in the specified field from upper case to lower case.

CTOX Instruct SyncSort to convert each byte in the specified field into 2-byte printable HEX digits. (p,l,TRAN=CTOX is equivalent to p,l,HEX)

XTOC Instruct SyncSort to convert each 2-byte HEX digits in the specified field into 1-byte binary value. The field length must be an even number of bytes long. HEX digits are EBCDIC characters '0' through '9', 'A' through 'F', or 'a' through 'f'. Non-HEX characters are treated as '0'.

ALTSEQ Instruct SyncSort to translate each byte in the specified field based on the ALTSEQ table in effect.

xxyy... Instruct SyncSort to translate each byte in the specified field that matches xx into yy. xx and yy are both HEX representations of a byte value. Up to 256 pairs of xxyy may be specified. Do not specify a comma (,) between pairs of xxyy.

C'cp...' Instruct SyncSort to translate each byte in the specified field that matches character c into character p. Up to 256 pairs of cp may be specified. Do not specify a comma (,) between pairs of cp.

The following example illustrates the use of the TRAN subparameter:

```
OUTREC FIELDS=(1,4,5,9,TRAN=LTOU
                  21,16,TRAN=C'(/)/-/#/',
                  37,20,TRAN=(X'005CFF5C',C'$_'))
```

Figure 111. Sample Use of the TRAN Subparameter

In the above example, the OUTREC control statement reformats the output record to contain the first four bytes of the input record, followed by nine bytes of the input record with lower case characters converted to upper case, followed by 16 bytes of input at column 21 with any '(' , ')', '-', or '#' characters converted to slashes ('/'), and followed by 20 bytes of input at column 37 with bytes of value X'00' and X'FF' changed to character '*' (X'5C') and all '$' signs changed to character '_'.

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

Multiple search-replacement paired values, 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.

- Replacement values are either character or hexadecimal string constants or a field from the input record. Multiple replacement constants on a CHANGE subparameter can be a mixture of character and hexadecimal string constants and fields from the input record.

- 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 subparameters for CHANGE are specified by the following syntax:

o,$srch_1$,$repl_1$[,$srch_2$,$repl_2$...,$srch_n$,$repl_n$]

Figure 112. Syntax for change-parms

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'bbbbbbbb'). 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 2.39.)

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', or X'20' if INPFIL DATA=A was specified). 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 value to which the input field is changed in the reformatted output record when the input field matches a search constant. The replacement value can either be a constant or a field from the input record.

The term repl represents the following syntax:

Figure 113. Syntax of repl

mrepl A replacement constant to which the input field is changed. The replacement formats that are permissible for constants 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', or X'20' if INPFIL DATA=A was specified). Hexadecimal strings are padded with zeros (X'00').

j,k The position j and the length k of an input field that will be inserted in the output record. k must be at least 1 and cannot be greater than the length *o* specified for the output replacement field. If *k* is less than *o*, the field j,k will be padded on the right with blanks (X'40', or X'20' if INPFIL DATA=A was specified) to the length *o*.

NOMATCH The NOMATCH subparameter 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.

The subparameters for NOMATCH are specified by the following syntax:

Figure 114. Syntax for nomatch-parms

The following describes the elements of the NOMATCH subparameter:

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', or X'20' if INPFIL DATA=A was specified) 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',
                            C'XX',50,13),
                        NOMATCH=(C'NOT SUPPORTED'),
               8X,
               24,1,
                 CHANGE=(10,B'1.......',C'EAST COAST',
                            B'0.......',C'WEST COAST'))
```

Figure 115. Sample OUTREC Parameter with CHANGE Subparameter

In the above example, the OUTREC 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 the state code is XX, posi-

tions 50 through 62 of the input record will be placed in the output record. 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 FIELDS=(24,1,
                  CHANGE=(6,B'.....11.',C'SHARE',
                            B'......1.',C'UNIQUE'))
```

Figure 116. 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'.

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 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, INPFIL INREC or OUTFIL OUTREC: 2-Digit Year Data” on page 2.194.

When a single numeric field defined by p,l,f_i is to be converted, the format and length of the field determines the length of the output field, as illustrated in the chart below.

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)
PSI	n	2n-1
PD0	n	2n-2 digits
SFF, UFF	n	n (to maximum of 15, then truncated)
Y2C, Y2P, Y2S, Y2Z	2	4 digits
Y2B, Y2D	1	4 digits
Y2ID	1	2 bytes
Y2IP	2	3 bytes
ZSI	1	1 byte

Table 29. Data Conversion Table

Figure 29 does not include full-date formats. For full-date formats, see "Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC or OUTFIL OUTREC: Full-Date Data" on page 2.196.

When any other type of expression is to be converted, the number of resulting digits (d) will be 15. d will be used to determine the default length of the output field, which may be a numeric data field or printable output.

If you specify no other FIELDS subparameters, the result will be converted to printable output according to the default editing mask, M0. Refer to "Mm Subparameter (Editing Masks)" on page 2.202. 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 pat-

terns, or by using one of the 26 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 second column represents the number of digits in the input.

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

Table 30. Output Length of Output Formats

These lengths can be overridden by specifying the LENGTH parameter.

The following six sections describe the data conversion capabilities:

- Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC, or OUTFIL OUTREC: 2-Digit Year Data
- Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC, or OUTFIL OUTREC: Full-Date Data
- 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, INPFIL INREC or OUTFIL OUTREC: 2-Digit Year Data

This section describes 2-digit year data. For information on converting full-date year data, see “Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC or OUTFIL OUTREC: Full-Date Data” on page 2.196.

A 2-digit year field, (as specified by the Y2B, Y2C, Y2D, Y2ID, Y2IP, Y2P, Y2S, and Y2Z formats) can be converted on output to a 4-digit year:

- The Y2C and Y2Z formats specify 2-digit year data that is in displayable (zoned decimal) format. The 2-digit year data will be expanded to a 4-digit field containing the appropriate century value.

- 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 18 on page 2.103.

- 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. (For a description of Y2D and Y2P formats, see Table 18 on page 2.103.)

- The Y2ID and Y2IP formats take as input the same 2-digit packed decimal year data as Y2D and Y2P formats but produce a 4-digit year output that remains in packed decimal format.

- The Y2S format is equivalent to Y2C and Y2Z for valid numeric year data. All three formats will convert such data to a 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 two 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 of the three types of data:

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

- The date data formats Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z are changed into 4 digit character (EBCDIC) format by OUTREC data conversion processing. If DATA=A has been specified on the INPFIL statement then the above date data formats will be converted into 4 digit ASCII character formats.

For information on using the date formats for SORT/MERGE field specifications, see “CENTWIN (Century Window) Processing with SORT: 2-Digit Year Formats” on page 2.235.

There are other data formats (PD0, PSI and ZSI) that do not convert to 2-digit format:

- The PD0 format 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.

Note: On an OUTREC statement, PZ is an acceptable synonym for PD0.

- The PSI format is used to process 1-8 byte packed decimal data. PSI ignores the trailing sign during processing. PSI is normally used in conjunction with the Y2D data format. The Y2D format is used to process the 2-digit year portion of a packed decimal data field, while the PSI format is used to process the day portion of the field.

  Although PSI is typically used with Y2D, the PSI format itself is not affected by CENTWIN processing.

  Consider the packed decimal date field:

  ```
  yyddd=X'yydddC'
  ```

  where the trailing C (sometimes F) is a positive sign. This field has the year (yy) in the first byte and the day (ddd) in bytes 2 and 3. The packed decimal sign would be in the last digit (half byte) of the third byte.

  The date can be processed as follows: Y2D processes the year component (X'yy'), while PSI processes the day components (X'dddC').

- The ZSI format is used to process 1-15 byte zoned decimal data. ZSI ignores the trailing sign during processing. ZSI is normally used in conjunction with the Y2Z data format. The Y2Z format is used to process the 2-digit year portion of a zoned decimal data field, while the ZSI format is used to process the day portion of the field.

  Although ZSI is typically used with Y2Z, the ZSI format itself is not affected by CENTWIN processing.

  Consider the zoned decimal date field:

  ```
  yymmdd=X'xyxyxmxmxdxd'
  ```

  where y, m and d are hexadecimal digits 0-9, and x is hexadecimal 0-F.

  The date can be processed as follows: Y2Z processes the year component (X'xyxy'). ZSI processes the day components (X'xmxmxdxd)'.

For more information on using the date formats for OUTREC processing, see “Sample OUTREC Control Statements with CENTWIN Processing: 2-Digit Year Formats” on page 2.213.

Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC or OUTFIL OUTREC: Full-Date Data

You can use full-date formats with INREC/OUTREC, INPFIL INREC, and OUTFIL OUTREC to simplify conversion of date fields to printable or packed decimal format. Such con-

version is often required to create output data with 4-digit years or to generate reports displaying 4-digit year dates.

The full-date formats will convert CH, ZD, and PD date fields of the form yyx...x or x...xyy. You can convert 2-digit year dates to character or packed decimal 4-digit year dates. You can also convert PD 2-digit year to 4-digit years without unpacking. The following table indicates the Y2x formats to use for different conversions to 4-digit year dates:

From 2-Digit Format	**To 4-Digit Format**	**Y2x Format to Use**	**Comment**
CH, ZD, PD	CH	Y2T, Y2U, Y2V, Y2W, Y2X, Y2Y	Example Z'mmddyy' to C'mmddyyyy'
CH, ZD, PD	PD	Y2TP, Y2UP, Y2VP, Y2WP, Y2XP, Y2YP	Example P'yyddd' to P'yyyyddd'
PD	PD	Y2ID, Y2IP	Y2DP and Y2PP are synonymous with Y2ID and Y2IP

Table 31. Y2x Formats to Use to Convert 2-Digit to 4-Digit Year Dates

For date formats, Y2x and Y2x(c), if DATA=A is specified in the INPFIL statement, the input field is translated appropriately to ASCII character formats in the output record.

Note that full-date formats expand non-date characters but do not apply century window processing. For example, if you are converting a field of the form P'yymm' to C'yyyy/mm', a P'9999' date will be expanded to C'9999/99'.

Table 28 on page 2.185 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)

CH and ZD formats are represented by X'FdFd...sd', and PD formats are represented by X'dd...ds', where “d” is a decimal digit (0-9) and “s” is a sign (0-F). For dates, the sign is not

relevant and can be ignored. See Table 28 on page 2.185 for examples of date fields in CH, ZD, and PD formats.

You can convert CH, ZD, and PD 2-digit year dates to any of three 4-digit year forms: a date without date separators, a date with date separators, a 4-digit packed decimal date.

The following table describes the three date forms:

Date Form	**Description**	**Field Specification Form**	**Example**
Y2x	4-digit year without date separators. Output data format is always CH.	p,l,Y2x	20,6,Y2W
Y2x(c)	4-digit year with date separators Output date format is always CH.	p,l,Y2x(c) A date separator can be any character except a blank.	20,5,Y2W(/) inserts slashes as date separators, producing output such as 137/2002.
Y2xP	4-digit year in packed decimal format. Output date format is always PD, even though input data format can be CH (for Y2T and Y2W) or PD (for Y2U, Y2V, Y2X, and Y2Y).	p,l,Y2xP	20,6,Y2TP converts C'yymmdd' to P'yyyymmdd'.

Table 32. 4-Digit Date Forms

For date formats Y2x and Y2x(c), if DATA=A is specified in the INPFIL statement, the input field is translated appropriately to ASCII character formats in the output record.

The following table summarizes the three types of data conversion with the full-date formats:

Date Form	Full-Date Format	Length (bytes)	Output Y2x	Output Y2x(/)	Output Y2xP
yyx	Y2T Y2U	3 2	C'yyyyx'	C'yyyy/x'	P'yyyyx'
yyxx	Y2T Y2V	4 3	C'yyyyxx'	C'yyyy/xx'	P'yyyyxx'
yyxxx	Y2T Y2U	5 3	C'yyyyxxx'	C'yyyy/xxx'	P'yyyyxxx'
yyxxxx	Y2T Y2V	6 4	C'yyyyxxxx'	C'yyyy/xx/xx'	P'yyyyxxxx'
xyy	Y2W Y2X	3 2	C'xyyyy'	C'x/yyyy'	P'xyyyy'
xxyy	Y2W Y2Y	4 3	C'xxyyyy'	C'xx/yyyy'	P'xxyyyy'
xxxyy	Y2W Y2X	5 3	C'xxxyyyy'	C'xxx/yyyy'	P'xxxyyyy'
xxxxyy	Y2W Y2Y	6 4	C'xxxxyyyy'	C'xx/xx/yyyy'	P'xxxxyyyy'

Table 33. Three Types of Data Conversion

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'yyyyddd'
TM1	Z'hhmmss'
TM2	Z'hhmm'
TM3	Z'hh'
TM4	Z'hhmmssxx'

Table 34. SMF Formats and Converted Output

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

The c in the date source P'cyyddd' 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 predefined edit masks (M0-M26) or specified edit patterns can be used to edit the converted date and time. The default mask is M11.

Note: A data exception 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.

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).

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 that encompasses all printable characters. Figure 117 illustrates the concept of replacing the insignificant and significant digit selectors I and T with other characters.

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

Figure 117. Replacing Digit Selector Characters

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

LENGTH=n Subparameter

Use the LENGTH=n subparameter to alter the default length 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 or padded with blanks on the left so that the length of the pattern equals the 'n' value.

The maximum value that can be specified for 'n' 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 35. Maximum Values of LENGTH=n for Output Data

Mm Subparameter (Editing Masks)

SyncSort for z/VSE provides editing masks to simplify the more common editing operations. If neither Mm nor EDIT is specified in the OUTREC control statement, the default mask, M0, is used.

Figure 118. EMASK Format

Where applicable (in most European countries), the default may be set via the SYNCMAC parameter at installation time, to interchange decimal commas and periods in the SyncSort supplied editing masks: M2, M3, M4, and M5.

Mask	Pattern	Signs	Length
Default Mask (M0)	IIII...IITS	(,,' ',-)	d+1
M1	TTTTT...TTTS	(,,' ',-)	d+1
M2	I,III,...,IIT.TTS	(,,' ',-)	d+1 + [d/3]
M3	I,III,...IIT.TTCR		d+2 + [d/3]
M4	SI,III,...,IIT.TT	(+,-)	d+1 + [d/3]
M5	SIII,...,IIT.TTS	(' ',(,' ',))	d+2 + [d/3]
M6	III-TTT-TTTT		12
M7	TTT-TT-TTTT		11
M8	IT:TT:TT		8
M9	TT/TT/TT		8
M10	IIIIIIIIIIIIT		d
M11	TTTTTTTTTTTTTTT		d
M12	SIII,..,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]
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	SIIIIIIIIIIIIT	(' ',-)	d+1
M26	STTTTTTTTTTTTTT	(+,-)	d+1

Table 36. Editing Masks

Note: The letter 'd' represents the number of resulting digits after data conversion. The brackets indicate that only the integer part of this division should be retained.

Table 36, above, illustrates the following:

- Edit pattern
- Leading or trailing signs, where appropriate
- Length

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 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).

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.

Figure 119. SIGNS Subparameter Format

where:

s_1=leading positive sign indicator

s_2=leading negative sign indicator

s_3=trailing positive sign indicator

s_4=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 single quotation marks. A single quotation mark used as a SIGNS character must be specified as a double quotation mark and enclosed within single quotation marks.

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.

IFOUTLEN Parameter (Optional)

The IFOUTLEN parameter overrides the maximum record length, which is automatically set by the IFTHEN parameter, and changes it to a specified value. The IFOUTLEN parameter may only be used in conjunction with the IFTHEN parameter.

The IFOUTLEN parameter automatically makes the following changes to the record to match the new length. A record longer than n is truncated to n. A fixed-length record shorter than n is padded with blanks to reach a length of n.

IFTHEN Parameter (Optional)

The IFTHEN parameter uses conditional logic which enables you to reformat records based on specified criteria. Multiple IFTHEN parameters may be specified within the same control statement and are processed sequentially.

The format of the IFTHEN parameter is illustrated below.

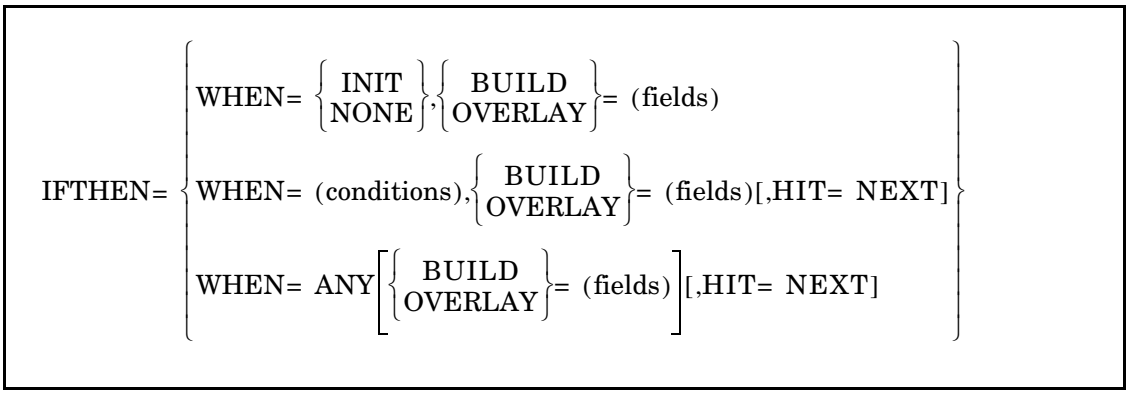


Figure 120. IFTHEN Parameter Format

At the beginning of IFTHEN processing, a temporary record is created from each of the input records.

The IFTHEN parameter automatically makes the following changes to the temporary record to accommodate any adjustments in length. In a variable-length record, the RDW length is adjusted accordingly. In a fixed-length record, the record is padded with blanks when necessary. Blanks also replace missing bytes in input fields.

The IFTHEN parameter has two main parts: the WHEN subparameter and a second subparameter. As shown in "Figure 119. IFTHEN Parameter Format", the WHEN subparameter may be WHEN=INIT, WHEN=(conditions), WHEN=ANY, or WHEN=NONE; and the second subparameter may be BUILD or OVERLAY. The WHEN subparameter defines a condition that must be satisfied before the second subparameter is applied to the temporary records. If the WHEN subparameter condition is not satisfied, then the second subparameter is not applied to the temporary records.

Since IFTHEN parameters refer to the temporary records instead of the input records, all subsequent IFTHEN parameters within the same control statement will take previous BUILD or OVERLAY changes into account. Once IFTHEN processing for each record stops, the temporary record becomes the output record.

The following describes the IFTHEN subparameters:

WHEN=INIT The WHEN=INIT subparameter condition is automatically satisfied. It applies the specified second subparameter to each temporary record as an initialization step.

A second subparameter is required.

WHEN=(conditions) The WHEN=(conditions) subparameter condition is satisfied if a temporary record meets the specified conditions. It applies the specified second subparameter to each temporary record that meets the specified conditions.

conditions The conditions must be formulated into a comparison or logical expression that can be evaluated to *true* or *false*.

For a complete description of comparisons and logical expressions, see "COND Parameter (Required)" on page 2.41. However, the following cannot be used in WHEN=(conditions):

- FORMAT=f

A second subparameter is required. The HIT=NEXT subparameter is optional.

WHEN=ANY The WHEN=ANY subparameter condition is satisfied if one or more of its associated WHEN=(conditions) subparameter conditions have been satisfied. Its associated WHEN=(conditions) subparameters are those that precede it but no other WHEN=ANY subparameter. If the WHEN=ANY subparameter condition is satisfied, it applies the specified second subparameter to the temporary record.

A second subparameter is optional. If it is not used, IFTHEN processing simply stops if the WHEN=ANY subparameter condition is satisfied unless the optional HIT=NEXT subparameter has been specified.

WHEN=NONE The WHEN=NONE subparameter condition is satisfied if none of the preceding WHEN=(conditions) subparameter conditions is satisfied or if there are no WHEN=(conditions) subparameters. If the WHEN=NONE subparameter condition is satisfied, it applies the specified second subparameter to the temporary record.

A second subparameter is required.

The IFTHEN parameters must be specified such that the WHEN subparameters are in the following order:

- WHEN=INIT
- WHEN=(conditions) and WHEN=ANY

- WHEN=NONE

BUILD The IFTHEN parameter will accept BUILD as a second subparameter. See "FIELDS/BUILD Parameter" on page 2.175 for a complete description of this subparameter.

OVERLAY The IFTHEN parameter will accept OVERLAY as a second subparameter. See "OVERLAY Parameter (Optional)" on page 2.209 for a complete description of this subparameter.

HIT=NEXT The HIT=NEXT subparameter is optional, but can only be used in conjunction with the WHEN=(conditions) or WHEN=ANY subparameter. IFTHEN processing stops by default once a WHEN=(conditions) subparameter condition or WHEN=ANY subparameter condition is satisfied. Including the HIT=NEXT subparameter will continue IFTHEN processing regardless of whether or not the WHEN subparameter condition is satisfied.

IFTHEN processing continues until:

- All IFTHEN parameters have been processed; or
- A WHEN=(conditions) or WHEN=ANY subparameter condition is satisfied and the HIT=NEXT subparameter is not included.

The following is an example of the IFTHEN parameter.

```
OUTREC IFTHEN=(WHEN=INIT,
                BUILD=(1,80,1,8,ZD,MUL,+107,DIV,+100,ZD)),
       IFTHEN=(WHEN=(81,15,ZD,GT,+10000),
                OVERLAY=(81:81,15,ZD,ADD,+0500,ZD),HIT=NEXT),
       IFTHEN=(WHEN=(81,15,ZD,GT,+20000),
                OVERLAY=(81:81,15,ZD,ADD,+2000,ZD),HIT=NEXT),
       IFTHEN=(WHEN=ANY,
                OVERLAY=(96:C'*',97:81,15,ZD,MUL,+15,DIV,+100)),
       IFTHEN=(WHEN=NONE,
                 OVERLAY=(97:81,15,ZD,MUL,+12,DIV,+100))
```

Figure 121. Sample IFTHEN parameter

This OUTREC control statement refers to 80-byte input records containing a salesperson's weekly sales in dollars in the first field (1,8,ZD) of each record. There are five IFTHEN parameters in the example and they reformat each input record as follows:

- The first IFTHEN parameter uses WHEN=INIT to take the sales total in the first field (1,8,ZD), increase it by 7%, and enter the result in a new 15-byte ZD field in column 81.

- The second IFTHEN parameter uses WHEN=(conditions) to test if the newly adjusted sales total in the field (81,15,ZD) is over $10,000. If it is, its OVERLAY subparameter increases the total by $500 and replaces the result in that field.

- The third IFTHEN parameter uses WHEN=(conditions) to test if the newly adjusted sales total in the field (81,15,ZD) is over $20,000. If it is, its OVERLAY subparameter increases the total by $2,000 and replaces the result in that field.

- The fourth IFTHEN parameter uses WHEN=ANY to test the second and third IFTHEN parameters. If one or both WHEN subparameter conditions are satisfied, indicating that the salesperson is getting a bonus, its OVERLAY subparameter inserts an "*" in column 96 to denote a bonus, calculates a commission rate of 15%, and enters the result in column 97.

- The fifth IFTHEN parameter uses WHEN=NONE to test the second and third IFTHEN parameters. If neither WHEN subparameter condition is satisfied, indicating that the salesperson is not getting a bonus, its OVERLAY subparameter calculates a commission rate of 12% and enters the result in column 97.

OVERLAY Parameter (Optional)

The OVERLAY parameter enables you to change particular columns within a record, and add fields to the end of a record, without rebuilding the entire record. When using the OVERLAY parameter you only need to specify the columns you want to change; the rest of the input record remains unchanged.

The format of the OVERLAY parameter is similar to that of the FIELDS parameter of the OUTREC control statement. See “FIELDS/BUILD Parameter” on page 2.175 for the format of this parameter. The following exceptions apply:

- In the OVERLAY parameter the length l is always required, unlike one case of the FIELDS parameter in which l is optional after p.

- In the OVERLAY parameter for a variable-length record, the column value c: is always required and must be set at 5 or greater, since c: is set to 1 by default and positions 1 through 4 comprise the RDW. The RDW cannot be overlaid.

The OVERLAY parameter automatically makes the following changes to the output record to accommodate any adjustments in length. In a variable-length record, the RDW length is adjusted accordingly. In a fixed-length record, the record is padded with blanks or user-

specified VLFILL characters when necessary. Blanks also replace missing bytes in input fields.

The following is an example of the OVERLAY parameter:

```
OUTREC OVERLAY=(9:9,4,PD,SUB,13,4,PD,PD,LENGTH=4,81:9,4,PD)
```

Figure 122. Sample OVERLAY parameter

Sample OUTREC Control Statements without Date Data

Example 1

The following example illustrates how the OUTREC 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 123. Sample OUTREC Control Statement

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

- Four binary zeros are inserted in the first four 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 begins in position 23. SyncSort for z/VSE 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 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 124. Sample OUTREC Control Statement

This OUTREC statement defines a 69-byte output record as follows:

- The first field begins in position 1. This field began in position 1 before OUTREC processing and is 50 bytes long. (1,50)

- The next field 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, the resulting field will be 9 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. (64,4,PD,M2)

- The last field begins in position 60. This zoned decimal field began in position 68 before OUTREC processing and is 6 bytes long. The EDIT and SIGNS subparameters 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. (68,6,ZD,EDIT=($I,IIT.TTS),SIGNS=(,,+, -))

Example 3

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

```
OUTREC FIELDS=(20,10,ZD,PD,
               11,4,FI,ZD,LENGTH=8)
```

Figure 125. Sample OUTREC Control Statement

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

- The first field (20,10,ZD,PD) begins in position 1. This field was a 10-byte ZD field that began in position 20 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 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 126. Example 4, Sample OUTREC Control Statement

This OUTREC 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. The largest of the values in Field A, Field B, and Field C is output as an 8-byte PD field.

- The fourth output field begins in position 60. 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.

Comprehensive examples illustrating the OUTREC control statement and the OUTREC parameter of the OUTFIL statement are provided in "Chapter 4. How to Use SyncSort Data Utility Features".

Sample OUTREC Control Statements with CENTWIN Processing: 2-Digit Year Formats

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 using 2-digit year formats. For full-date formats, see "Sample OUTREC Control Statements with CENTWIN Processing: Full-Date Formats" on page 2.217.

Example 1 (Y2C):

To expand a 2-digit year field in character format at position 20, use a specification similar to the following:

```
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 127. Sample OUTREC Control Statement to Expand a 2-Digit Year Field

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 (PD0 and Y2P):

Consider the following packed decimal date field:

yymmdd=X'0yymmddC'

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,                * copy first portion of record
               21,2,PD0,EDIT=(TT), * convert X'ymmd' to X'mm' then C'mm'
               C'/',               * insert slash
               22,2,PD0,EDIT=(TT), * convert X'mddC' to X'dd'then C'dd'
               C'/',               * insert slash
               20,2,Y2P,           * convert X'0yym' to X'yy' then C'yyyy'
               24,76)              * copy rest of record
```

Figure 128. Sample OUTREC Control Statement for Displaying a 4-Digit Year

The 4-digit year output from the input year field (20,2,Y2P) depends on the CENTWIN setting. The following sample input and output data shows the case for CENTWIN=1980:

SORTIN Input Date Field	OUTREC Output Date Field
0800329C	03/29/1980
0790603C	06/03/2079

Example 3 (Y2ID):

To expand a 3-byte packed decimal date field of the form X'yyddds' to a 4-byte packed field of the form X'yyyyddds' that contains a prefixed century value, specify an OUTREC control statement such as the following:

```
OUTREC FIELDS=(1,19,                * copy first portion of record
               20,1,Y2ID,           * convert X'yy' to X'yyyy'
               21,60)               * copy rest of record starting with
                                    * the X'ddds' of the date field
```

Figure 129. Sample OUTREC Control Statement to Expand a 3-Byte Field

Note that in the above example the output record length will be 1 byte larger than the input record length. The following sample 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 (Y2IP):

To expand a 4-byte packed decimal date field of the form X'0yymmdds' to a 5-byte field of the form X'0yyyymmdds' that contains a prefixed century value, specify an OUTREC control statement such as the following:

```
OUTREC FIELDS=(1,19,              * copy first portion of record
               20,2,Y2IP,         * convert X'0yym' to X'0yyyym'
               22,59)             * copy rest of record starting with
                                  * the X'mdds' of the date field
```

Figure 130. Sample OUTREC Control Statement to Expand a 4-Byte Field

As with Y2ID conversion, the output record length will be 1 byte larger than the input length. The following sample input and output data shows the effect for CENTWIN=1980:

SORTIN Input Date Field	OUTREC Output Date Field
X'0791'	X'020791'
X'0800'	X'019800'

Example 5 (Y2D and PSI):

To convert a 3-byte packed decimal date field of the form X'yyddds' to a 8-byte character field of the form C'yyyy/ddd' that contains a prefixed century value, specify an OUTREC control statement such as the following:

```
OUTREC FIELDS=(1,19,              * copy first portion of record
               20,1,Y2D,          * convert X'yy' to C'yyyy'
               C'/',              * insert slash (/)
               21,2,PSI,          * convert X'ddds' to C'ddd'
               23,58)             * copy rest of record
```

Figure 131. Sample OUTREC Control Statement to Convert a 3-Byte Field

The output record length will be 5 bytes larger than the input length. The following sample input and output data shows the effect for CENTWIN=1980:

SORTIN Input Date Field	OUTREC Output Date Field

```
X'79124C'          2079/124
X'80022C'          1980/022
```

Example 6 (Y2Z and ZSI):

To convert a 6-byte zoned decimal date field of the form X'FyFyFmFmFdFd' to a 10-byte character field of the form C'yyyy/mm/dd' that contains a prefixed century value, specify an OUTREC control statement such as the following:

```
OUTREC FIELDS=(1,19,               * copy first portion of record
               20,2,Y2Z,           * convert X'FyFy to C'yyyy'
               C'/',               * insert slash (/)
               22,2,ZSI,           * convert X'FmFm' to C'mm'
               C'/',               * insert slash (/)
               24,2,ZSI,           * convert X'FdFd' to C'dd'
               26,55)              * copy rest of record
```

Figure 132. Sample OUTREC Control Statement to Convert a 6-Byte Field

The output record length will be 4 bytes larger than the input length. The following sample input and output data shows the effect for CENTWIN=1980:

SORTIN Input Date Field	OUTREC Output Date Field
X'F7F9F1F1F1F7'	2079/11/17
X'F8F0F0F2F0F5'	1980/02/05

Example 7 (Y2S)

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 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 133. Sample OUTREC Control Statement to Convert Header/Trailer Fields

As with Y2C or Y2Z, the output record length will be 2 bytes larger than the input record length.

For CENTWIN=1990 and an ascending sort, the Y2S field would be sorted and converted as follows:

SORTIN Input Date Field	OUTREC Output Date Field
X'4001'	X'00000000'
X'F9F8'	X'40404001'
X'F0F3'	X'F1F9F9F8'
X'0000'	X'F2F0F0F3'
X'FFFF'	X'FFFFFFFF'

Sample OUTREC Control Statements with CENTWIN Processing: Full-Date Formats

The following provides examples of data conversion with CENTWIN using full-date formats.

Example 1

The following OUTREC control statement converts a 6-byte Z'yymmdd' input date field to a 10-byte C'yyyy/mm/dd ' output date field. As indicated in Table 28 on page 2.185 the 6-byte form of Y2T is the appropriate format to use for this conversion. The previously set century window is 1950-2049.

```
SORT FIELDS=COPY
OUTREC=(1,30,          * Copies field 1,30.
        31,6,Y2T(/))   * Converts Z'yymmdd' to C'yyyy/mm/dd'.
```

Figure 134. Sample OUTREC Control Statement for 6-Byte to 10-Byte Conversion

The OUTREC control statement converts a 6-byte Z'mmddyy' date field in position 31 to a 10-character date field. Thus, the date field will expand by 4 bytes. The expansion is shown in the following table. Also shown in the table is the conversion of non-date data (zeros). Such data is converted but century window processing is not applied to it.

SORTIN Input (Z'yymmdd')	SORTOUT Output (C'yyyy/mm/dd')
740514	1974/05/14
101121	2010/11/21
500803	1950/08/03
000000	0000/00/00

Example 2

The following OUTREC control statement converts a 3-byte P'yyddd' packed decimal date field to a 4-byte P'yyyyddd' packed decimal date field. As indicated in Table 28 on page 2.185 the Y2UP format is appropriate for this conversion. The previously set century window is 1970-2069.

```
SORT FIELDS=COPY
OUTREC=(20,3,Y2UP)   * Converts 3-byte P'yyddd' to 4-byte
                     * P'yyyyddd'.
```

Figure 135. Sample OUTREC Control Statement for 3-Byte to 4-Byte Conversion

The packed decimal input format is retained in the output data, as indicated in the following table:

SORTIN Input (P'yyddd')	SORTOUT Output (P'yyyyddd')
P'60347'	P'2060347'
P'99002'	P'1999002'
P'70143'	P'1970143'
P'99999'	P'9999999'
P'69113'	P'2069113'
P'84175'	P'1984175'

RECORD Control Statement

The RECORD control statement provides record length and format information. The RECORD statement can also be used to delete certain records before sort/merge processing The RECORD control statement is required unless the JOINKEYS control statement is used.

The format of the RECORD control statement is illustrated below.

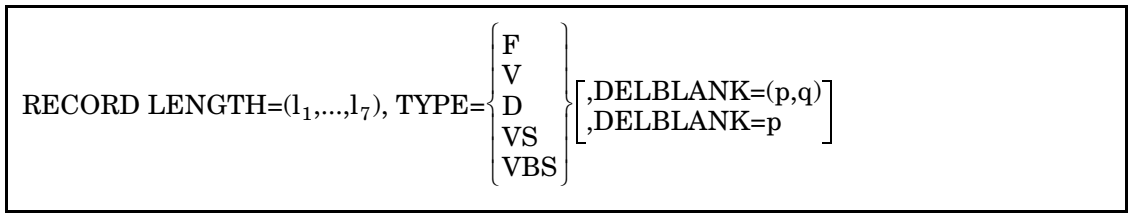


Figure 136. RECORD Control Statement Format

LENGTH Parameter (Required)

The LENGTH parameter specifies the length of the record at various points during the processing of the application. These points are illustrated in Figure 3 on page 2.18.

The number of length values can vary from 1 to 7. Only the l_1 value is required. If l_1 is the only value used, parentheses are optional. If l_1 and additional length values are used, they all must be enclosed in parentheses.

The length values are positionally dependent. An extra comma must indicate a missing l value between any two that are specified. Commas need not follow the final l value specified. For example, if LENGTH=(l_1,,,l_4) is specified, the omitted values are understood to be l_2 ,l_3, l_5, l_6, and l_7.

If the KEYLEN or ADDROUT parameter is specified on the OPTION control statement, refer to Table 22 on page 2.118 for information on calculating RECORD length values.

The l_1,...,l_7 variables specify the following:

l_1 The maximum record length of the logical records at input to the sort/merge, **after** any length change from INPFIL INREC control statement(s) but **before** E15 processing. (See Figure 3 on page 2.18.) 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. For tape input, 18 bytes is the minimum record length that is permitted.

l_2 The maximum length of the logical records **after** INPFIL INREC E15 processing. Do not specify an l_2 value to reflect a

length change due to the INREC control statement; the post-INREC length is calculated automatically. However, it *is* necessary to specify an l_2 value if an E15 exit has altered the record length. If an l_2 value is specified, it must be the length after INPFIL INREC and/or E15 have altered the record length.

l_3 The maximum length of the logical records **after** OUTREC or E35 processing. If the record length has not changed from the l_2 value, either as specified or as defaulted, you may omit the l_3 value. It is not necessary to specify an l_3 value to reflect a length change due to the OUTREC statement or the OUTREC parameter on the OUTFIL statement; the post-OUTREC length is calculated automatically. However, it is necessary to specify an l_3 value if an E35 exit has altered the record length and the sort is writing the output file.

l_4 The minimum length of the variable-length logical records plus the 4-byte Record Descriptor Word. This length should take into account any modifications made by an E15 exit program and/or an INPFIL INREC statement, but not an INREC control statement; the post-INREC length is calculated automatically. An omitted l_4 value defaults to the number of bytes needed to contain all of the control fields specified on the INCLUDE/OMIT, INREC, MERGE, OUTREC, SORT, SUM and/or OUTFIL statements.

The l_4 value is not used for a merge, but if it is specified, SyncSort will check record lengths.

l_5 The most frequent record length of the variable-length records. This length should take into account any modifications made by an E15 exit program and/or an INPFIL INREC statement, but not an INREC control statement; the post-INREC length is calculated automatically. Specify this length value to optimize the size of the segment, i.e., the fixed-length block reserved to contain variable-length records when they are placed on intermediate storage devices.

The l_5 value is not used for a merge. If specified, it will be ignored.

l_6 The average work space required by each variable-length record. The VL5L6L7 PARM or the SYNCHSTO utility program can be used to calculate this value.

l_7 The optimum segment length when sorting variable-length records. The VL5L6L7 PARM or the SYNCHSTO utility program can be used to calculate this value.

Rules for Specifying the Length Parameter

Observe the following rules when specifying length values:

- Make sure that *all* length values for variable-length records include 4 bytes for the Record Descriptor Word.
- Make sure that the l_1, l_2 and l_3 values represent the *maximum* record length and that the l_4 value represents the *minimum* record length. If SyncSort encounters a record that exceeds the maximum length or is shorter than the minimum length, the application will either terminate abnormally or produce unpredictable results. However, if the BYPASS option is specified on the INPFIL control statement, SyncSort will omit variable-length records that are too long or too short and continue processing.
- The sort internally adjusts record lengths when the INREC control statement is used. When INREC is used on the INPFIL control statement, however, no internal adjustment is done. Therefore, the l_1, l_2, l_3, l_4, and l_5 values must be specified as the lengths after INPFIL INREC but before the (non-INPFIL) INREC Control Statement. (See Figure 3 on page 2.18.) When INPFIL INREC reformats records, use the LRECL parameter of INPFIL to specify the original maximum and minimum record lengths.

TYPE Parameter (Required)

The TYPE parameter indicates the record format. Specify this parameter as follows:

TYPE=F	fixed-length EBCDIC or ASCII input records.
TYPE=V	variable-length EBCDIC input records.
TYPE=D	variable-length ASCII input records.
TYPE=VS	variable-length spanned EBCDIC input records.
TYPE=VBS	variable-length blocked spanned EBCDIC input records.

Figure 137. TYPE Parameter Format

If TYPE=D is specified, the BUFOFF parameter can be specified on the OUTFIL statement but the DELBLANK parameter cannot be specified on the RECORD control statement. If TYPE=D is specified but DATA=A is not specified on the INPFIL control statement, variable-length ASCII input records are assumed. The restrictions for ASCII data, which are outlined under the DATA=EA parameter on the INPFIL control statement, will apply.

If TYPE=VS or TYPE=VBS is specified, SyncSort insures that all output files are also spanned.

Only one type of record can be specified for an application.

Note: when FTOV or VTOF is used on the INPFIL control statement to reformat the input records before sorting, the TYPE parameter must specify the format of the records ***after*** the format conversion is done.

DELBLANK Parameter (Optional)

The DELBLANK parameter allows certain records to be deleted before sort/merge processing. This parameter cannot be used if either an INCLUDE statement or an OMIT statement is specified. The DELBLANK parameter also cannot be specified when DATA=A is specified on the INPFIL control statement or TYPE=D is specified on the RECORD control statement.

The DELBLANK parameter can be specified in two formats.

When DELBLANK=(p,q) is specified, p specifies the position of q. The q value represents any EBCDIC value except a left or right-hand parenthesis, a comma or a blank. SyncSort for z/VSE omits any record with the specified q value in the p position.

If DELBLANK=p is specified, SyncSort for z/VSE omits any record that has a *blank* in the p position.

For variable-length records, remember to add 4 bytes for the Record Descriptor Word when specifying the p value.

All references to field positions within the data record refer to the record after processing by an E15 exit, if specified. The p position must be located within the first 4092 bytes of the logical record.

Sample RECORD Control Statement

```
RECORD TYPE=V,LENGTH=(104,,,84,104),DELBLANK=(5,X)
```

Figure 138. Sample RECORD Control Statement

This sample RECORD statement defines the record as follows:

- The file contains variable-length records.
- The l_1 or longest input record to the sort will be 104 bytes. (This includes the 4-byte Record Descriptor Word.) This is the only required length value.

- The l_2 value is omitted because INREC and/or E15 processing did not alter the record length. A comma indicates the omitted l_2 value.
- The l_3 value is omitted because OUTREC and/or E35 processing did not alter the record length. A comma indicates the omitted l_3 value.
- The l_4 value indicates that the minimum record length is 84 bytes.
- The l_5 value indicates that the most frequent record length is 104 bytes.
- The DELBLANK parameter specifies that any record with an X in byte 5 be omitted.

REFORMAT Control Statement

The REFORMAT control statement defines the record layout to be produced by the join processing specified on an application's JOINKEYS control statement.

Use the REFORMAT control statement to specify which fields from the SORTIN1 and SORTIN2 files are to be included in each record created by the join operation.

The REFORMAT control statement is normally required if JOINKEYS is specified. It is optional if a JOIN control statement with the ONLY option has been specified since no records will actually be joined. In that instance, if a REFORMAT control statement is not provided and only the unpaired records from one join input (SORTIN1 or SORTIN2) are requested on the JOIN control statement, the records will not be reformatted and the record type and length of the join input file will be retained.

If a REFORMAT control statement is not provided and unpaired records from both join inputs are requested, the resultant records will be variable-length, regardless of the record formats of the join input files, and the record length will be the maximum of any fixed-length input file record length plus four (for an RDW) and any variable-length input file record length.

REFORMAT Control Statement Format

The format of the REFORMAT control statement is illustrated below:

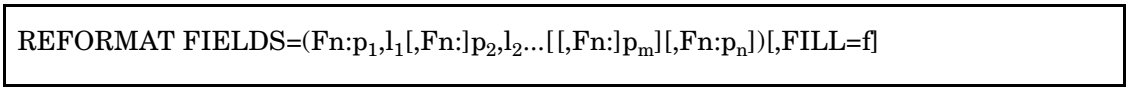


Figure 139. REFORMAT Control Statement Format

FIELDS Parameter (Required)

The FIELDS parameter specifies fields to be included in the record produced by the join function.

Each data field specified in the FIELDS parameter is identified by the file it originates from Fn, its position p and length l.

Fn: The Fn value indicates the input file from which the data field should be copied. Code 'F1' for SORTIN1 and 'F2' for SORTIN2. This field is optional after the first field specification. By default, the file of the prior field specification will be used to determine the current field specification.

p The position value indicates the first byte of the field relative to the beginning of the input record

l The length value indicates the length of the field.

Specifying the FIELDS Parameter for Variable-Length Records

If the REFORMAT control statement only defines p,l fields, then the output of the join will be a fixed-length record. If a variable-length record format is desired when one or both input files are variable-length, then the first p,l REFORMAT field must be l,4 (from either SORTINn input file) to define the RDW. This p,l specification of l,4 must reference an Fn that is a variable-length file. The variable portion of the record must then be specified as the last REFORMAT field by coding a position p without a length l. If both files are variable-length, then the variable portion from each of the variable-length input files may be specified once at the end of the REFORMAT statement. For example:

```
REFORMAT FIELDS=(F1:1,4,F1:10,10,F2:25,3,F1:40,F2:50)
```

Figure 140. Sample REFORMAT Control Statement

FILL Parameter (Optional)

The FILL parameter defines a fill byte to be used for any missing p,l field bytes. *f* specifies the fill byte. *f* can be specified as either a character or hexadecimal value. Specify either C'c' where c is a single EBCDIC character, or X'hh' where hh represents a hexadecimal digit pair (00-FF).

The need for a fill byte can arise from two conditions:

- A portion or an entire p,l field specification is missing due to a short variable-length record.
- A JOIN UNPAIRED was used and the REFORMAT FIELDS specification requires a field from the file that is not being used to generate the current joined record.

For example, an application contains the following JOIN and REFORMAT control statements:

```
JOIN UNPAIRED,F1
REFORMAT FIELDS=(F1:10,10,F2:12,5),FILL=C'0'
```

Figure 141. Sample JOIN and REFORMAT Control Statements

If a record is found in SORTIN1 that does not match a record in SORTIN2, the unpaired record will be included in the join output and would look as follows:

POSITION	VALUE
1-10	Contents of the SORTIN1 record positions 10 through 19.
11-15	Filled with 0's since SORTIN2 does not participate in building this record.

Figure 142. Sample Output of Unmatched SORTIN1/SORTIN2 Records

The default FILL character is blank. Binary zeros will be used instead of the FILL character for the first four bytes of a variable-length record requiring FILL processing. This indicates that a record was not present for the REFORMAT due to JOIN UNPAIRED.

SORT Control Statement

The SORT control statement defines the application as a sort. The SORT statement can also define a copy application.

The SORT control statement is required for every sort.

The format of the SORT control statement is illustrated below.

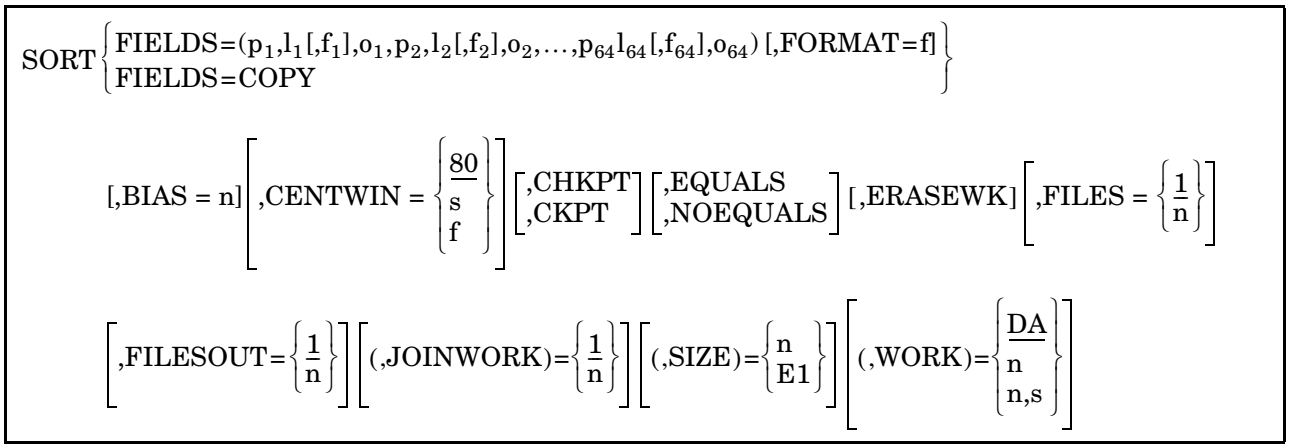


Figure 143. SORT Control Statement Format

FIELDS Parameter (Required for a Sort)

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

As many as sixty-four control fields can be specified. List the control fields in order of greatest to least priority, with the primary control field (p_1,l_1,f_1,o_1) listed first, followed by progressively less significant fields (p_2,l_2,f_2,o_2,...,p_{64},l_{64},f_{64},o_{64}).

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 may 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 integral number of bytes except for the length of a binary control field, which may be specified in bits. For example, a length value of 0.6 refers to a binary control field 6 bits long.

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

f The format value indicates the data format. Refer to Table 37 below for a list of valid formats. If the format value is omitted for a control field, then the format value in the FORMAT=f subparameter is used for this control field.

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

A=Ascending order

D=Descending order

Valid Formats for Control Fields

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

Code	Data Format	Acceptable field length in bytes
AC	EBCDIC characters are translated to their ASCII equivalents before sorting.	1 to 256
AQ	Character. Records are sorted according to an alternate sequence specified either in the ALTSEQ control statement or as an installation default.	1 to 256
ASL	Leading separate sign. An ASCII + or - precedes a numeric field. One digit per byte.	2 to 256
AST	Trailing separate sign. An ASCII + or - trails a numeric field. One digit per byte.	2 to 256
BI	Binary. Unsigned.	1 bit to 4092
CH	Character. Unsigned. Characters are collated according to the EBCDIC collating sequence unless LOCALE has been used to chose an alternate set of collating rules based on a specific national language. LOCALE can be set on the OPTION statement or as an installation default.	1 to 4092
CLO or OL	Leading overpunch sign. Hexadecimal F,C,E or A in the first 4 bits of the 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

Table 37. (Page 1 of 3) Format Code Chart

Code	Data Format	Acceptable field length in bytes
CSF or 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 or LS	Leading separate sign. An EBCDIC + or - precedes a numeric field. One digit per byte. CMP=CLC is forced.	2 to 256
CST or TS	Trailing separate sign. An EBCDIC + or - follows a numeric field. One digit per byte. CMP=CLC is forced.	2 to 256
FI	Fixed point. Signed.	1 to 256
FL	Signed floating point. Normalized.	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
SFF	Signed free format. Decimal digits (0-9) are extracted from right to left to form a number value. A character of - or) found within the field will cause the value to be treated as a negative number. All other non-decimal digit values in the field are ignored.	1 to 44
UFF	Unsigned free format. Decimal digits (0-9) are extracted from right to left to form a number value. All non-decimal digit values in the field are ignored.	1 to 44
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

Table 37. (Page 2 of 3) Format Code Chart

Code	Data Format	Acceptable field length in bytes
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 is 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 formats. Enable sorting or merging a variety of date fields. The formats can process dates ending or starting with year digits and non-date data commonly used with the dates. Two-digit years are interpreted according to the CENTWIN setting. In most cases, for CH, ZD, and PD dates fields the full-date data formats are easier to use than the 2-digit year formats. For a description of these formats see Table 38 on page 2.231.	
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 or CTO	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

Table 37. (Page 3 of 3) Format Code Chart

The table below describes the valid full-date formats: Y2T, Y2U, Y2V, Y2W, Y2X, and Y2Y.

Full-Date Format	Date 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	dddyy	5
		xxxxyy	mmddyy	6
Y2X	PD	xyy (X'xyys')	qyy	2
		xxxyy (X'xxxyys')	dddyy	3
Y2Y	PD	xxyy (X'0xxyys')	mmyy	3
		xxxxyy (X'0xxxxyys')	mmddyy	4

Note: The following symbols are used in the table:

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

Table 38. Full-Date Formats

For information on using the full-date formats, see “CENTWIN (Century Window) Processing with SORT: Full-Date Formats” on page 2.240 and “Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC or OUTFIL OUTREC: Full-Date Data” on page 2.196.

For information on the 2-digit year formats (Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z) plus the related data format PD0) see "CENTWIN (Century Window) Processing with SORT: 2-Digit Year Formats" on page 2.235 and "Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC or OUTFIL OUTREC: 2-Digit Year Data" on page 2.194.

Rules for Specifying Control Fields

- The sum of the lengths of all control fields cannot exceed 4092 bytes. When the EQUALS option is specified, the sum of their lengths cannot exceed 4088 bytes.
- Control fields can be in contiguous or noncontiguous locations in the record.
- 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.

Comparing PD and ZD Control Fields

When you specify CMP=CPD, SyncSort may use the Compare Decimal (CP) instructions for the comparison, which means that ZD data fields are packed first then compared. This method provides performance advantages. However, invalid PD data may cause a Data Exception abend and program termination. Moreover, the integrity of ZD fields is only guaranteed when the fields contain valid ZD data. Since the zone bits (the left most 4 bits of each byte) are lost during PACKing, UNPKing the field only restores valid ZD data to its original state. For example, blanks are transferred into ZD zeros and alphabetic character data that packs to a valid PD field is converted into valid ZD data.

When CMP=CLC is in effect, SyncSort does not perform data validation and the integrity of the output is maintained.

FIELDS=COPY (Required for a Copy)

Use FIELDS=COPY to copy input file(s). Other control statements such as INCLUDE/OMIT, INREC, OUTREC and OUTFIL and user exit routines can 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 the CONVERT parameter of the OUTFIL control statement, cannot be specified with FIELDS=COPY.

BIAS Parameter (Optional)

The BIAS parameter is ignored.

CENTWIN Parameter (Optional)

The CENTWIN option defines a sliding or fixed 100-year window that determines the century to which 2-digit year data belongs when processed by SORT, MERGE, INCLUDE, INREC, OMIT, OUTREC or OUTFIL OUTREC control statements.

There are two types of date data formats: 2-digit year and full-date:

- For details on CENTWIN processing and the 2-digit year data formats with the SORT control statement, see "CENTWIN (Century Window) Processing with SORT: 2-Digit Year Formats" on page 2.235.

  For information on using INCLUDE/OMIT with 2-digit year formats, see "Specifying Standard Comparisons for Date Fields" on page 2.50.

  For information on using the 2-digit year formats for OUTREC processing, see "Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC or OUTFIL OUTREC: 2-Digit Year Data" on page 2.194 and "Sample OUTREC Control Statements with CENTWIN Processing: 2-Digit Year Formats" on page 2.213.

- For details on CENTWIN processing and the full-date formats with the SORT control statement, see "CENTWIN (Century Window) Processing with SORT: Full-Date Formats" on page 2.240.

  For information on using INCLUDE/OMIT with full-date formats, see "Specifying Standard Comparisons for Date Fields" on page 2.50 and "Sample INCLUDE/OMIT Control Statements with Date Data" on page 2.60.

  For information on using the full-date formats for OUTREC processing, see "Converting Year Data with Century Window Processing on INREC, OUTREC, INPFIL INREC or OUTFIL OUTREC: Full-Date Data" on page 2.196.

CKPT/CHKPT Parameter (Optional)

This parameter instructs SyncSort to take a single checkpoint during a sort. SORTWK files must be saved in order to restart the sort.

When the CKPT parameter is specified, a SYS000 direct access device or a 2400/3400 series tape device must be assigned as a checkpoint file. (See Table 45 on page 5.4.) This file must have standard labels and must be given the filename SORTCKP in the TLBL/DLBL statement.

Use standard restart procedures to restart the sort from the checkpoint. Refer to the appropriate *System Management Guide* for detailed instructions.

Because taking a checkpoint does cause some performance degradation, specify this parameter only for sorts that take a long time to complete.

EQUALS/NOEQUALS Parameter (Optional)

The EQUALS parameter preserves the original order of records with equal control fields. These records are written in the same order in which they were read. The NOEQUALS parameter specifies that equal-keyed records be written in random order. NOEQUALS is the default.

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

ERASEWK Parameter (Optional)

The ERASEWK parameter specifies that the contents of the work files opened by the sort are to be erased at the end of the sort. (If an incore sort is performed, no files are erased.)

FILES Parameter (Optional)

The FILES parameter specifies the number of input files to be sorted. The 'n' value may be any number from 1 to 32. If the FILES parameter is not specified, one input file is assumed. **Note:** When using with the JOIN process, one output file is assumed.

FILESOUT Parameter (Optional)

The FILESOUT parameter specifies the number of output files and is required when creating multiple output files. The 'n' value may be any number from 1 to 32. If the FILESOUT parameter is not specified, one output file is assumed.

JOINWORK Parameter (Optional)

The JOINWORK parameter specifies the maximum number of sortwork files that the join process will use. The ‘n’ value can be any number from 1 to 9. If the join process is used and JOINWORK is not specified, one sortwork file is assumed.

SIZE Parameter (Optional)

The SIZE parameter specifies the total number of records in the input file(s). Replace 'n' with an accurate estimate of the number of records. Because sort performance is not affected by the SIZE parameter, this parameter is used primarily in conjunction with the ANALYZE control statement and the CALCAREA parameter on the OPTION statement. These features determine the amount of sort work space required for a particular application.

SIZE=E1 can be used to call a Parmexit to override SyncSort defaults for a particular sort. A default can be specified at installation time to call a Parmexit for every sort, providing the ability to vary certain parameters on the basis of internal criteria, e.g., adjusting

storage requirements according to the time of day. For information on Parmexits, see the *SyncSort for z/VSE Installation Guide*.

WORK Parameter (Optional)

The WORK parameter specifies the total number of disk files to be used for sort work.

WORK=DA is equivalent to WORK=1; SyncSort will determine whether the sort work file is a DA file, an SD file or a VSAM-managed SAM file. If multiple extents are used for sort work files and the JCL statements follow Method 1, in which one DLBL statement precedes all EXTENT statements (see "Chapter 5. Job Control Language and Sample Control Statement Streams"), WORK=DA must be specified on the SORT control statement.

For an incore sort, specify WORK=0. In any event, SyncSort may perform an incore sort if it is more efficient.

For sequential disk files, replace 'n' with the number of sort work files SyncSort should use for primary space. Specify WORK=(n,s) only in conjunction with Disk Space Management packages. Replace 's' with the number of secondary work packs SyncSort can access within the DSM pool, overriding the default set at installation time. For multiple sortwork files, performance can be enhanced by assigning the files to separate devices. See "Setting up Disk Work File Statements" on page 5.5.

Note: If OPTWORK=m has been specified as an installation parameter, with m>n, the sort may improve performance by using more than 'n' work files. See the *SyncSort for z/VSE Installation Guide* for details.

CENTWIN (Century Window) Processing with SORT: 2-Digit Year Formats

The CENTWIN run-time or installation option acts on 2-digit year data that spans centuries. CENTWIN treats 2-digit year data as a 4-digit year and sequences the data according to the 4-digit representation.

For information on using full-date data, see "CENTWIN (Century Window) Processing with SORT: Full-Date Formats" on page 2.240.

CENTWIN generates a century window (for example, 1950 through 2049) that determines what century a two-digit year field belongs to. CENTWIN ensures that year data spanning centuries will be sequenced correctly. 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 more information on specifying the CENTWIN option, see "CENTWIN Parameter (Optional)" on page 2.119.

CENTWIN processing applies to data defined as 2-digit year formats: Y2B, Y2C, Y2D, Y2P, Y2S, and Y2Z. 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/MERGE processing, you should be familiar with the CENTWIN-related data formats.

The following describes each of the date formats and provides example SORT control statements:

Note: For simplicity, the sample date fields in the example SORT statements below begin at byte 20. Also note that date data is always sorted in the following order: year (yy), month (mm), day (dd).

- Y2B

  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 39. Possible Values Representing Year Data with Y2B

- Y2C and Y2Z

  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 mmddyy 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,   * sort yy field as 4-digit year
             20,2,CH,A,    * sort mm field
             22,2,CH,A)    * sort dd field
```

Figure 144. Sample SORT Control Statement using Y2C

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'.

- Y2D

  This format is used to sort 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

  yyddds

  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 years processing, use the following SORT control statement:

```
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)
```

Figure 145. Sample SORT Control Statement using Y2D

- Y2P

  This format is used to sort or merge 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 two bytes. For example, a packed decimal date of the form yymmdd would be stored as four 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
```

Figure 146. Sample SORT Control Statement using Y2P

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

- PD0

  This format is used to sort or merge 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'ymmddC' 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'ymmddC' as X'mmdd'
```

Figure 147. Sample SORT Control Statement using PD0

- Y2S

  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 begins 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:

  - 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 begins in column 20, you would use the following sort key specification:

```
SORT FIELDS=(20,2,Y2S,A)  * Sorts yy field as 4-digit year
```

Figure 148. Sample SORT Control Statement using Y2S

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
```

Figure 149. Sample SORT Control Statement using Y2Z

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 the header and trailer records lose their position at the start and end of the file.

CENTWIN (Century Window) Processing with SORT: Full-Date Formats

SyncSort's full-date 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, dddyy, or mmddyy)
- 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 formats are easier to use than the 2-digit year 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 for-

mats, on the other hand, process such dates automatically. For information on the 2-digit year formats, see Table 38 on page 2.231 and "CENTWIN (Century Window) Processing with SORT: 2-Digit Year Formats" on page 2.235.

The following two examples illustrate how you might use Table 38 on page 2.231 to develop SORT control statements:

- 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 date field starts in position 30, you would specify the following SORT control statement to sort the date in descending order:

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

Figure 150. Sample SORT Control Statement using Y2Y

  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)
```

Figure 151. Sample SORT Control Statement using Y2T

  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 be 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).

Table 38 on page 2.231 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 table shows the order for ascending collation when using full-date formats with the CENTWIN option:

Full-Date Format	Data 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 40. Collation Order for Full-Date Formats

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 then higher century dates.

Sample SORT Control Statements with CENTWIN: Full-Date Formats

The following examples illustrate the use of full-date formats with the SORT control statement.

- **Example 1 (Y2W)**

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

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

Figure 152. Sample SORT Control Statement using Y2W

Table 38 on page 2.231 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 (mmddyy)	Record Order after Sorting (mmddyy)	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.
```

Figure 153. Sample SORT Control Statement using Y2T

Table 38 on page 2.231 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'yyyyddd', with non-date data (nines and zeros) appearing correctly at the beginning and end of the sorted output.

SORTIN Input (mmddd)	Record Order after Sorting (mmddd)	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'mmddyy' (X'0mmddyys') in ascending order, with the previously set fixed century window 1921-2020:

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

Figure 154. Sample SORT Control Statement using Y2Y

Table 38 on page 2.231 indicates that the 4-byte Y2Y field 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 and end of the sorted output. Note that the first two columns are in hexadecimal.

SORTIN Input (mmddyy)	Record Order after Sorting (mmddyy)	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

Sample SORT Control Statements without Date Data

```
SORT FIELDS=(2.3,2,BI,D,8,2.4,BI,A,25,10,CH,A,15,10,LS,D)
```

Figure 155. 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 in 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,WORK=4
```

Figure 156. 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.
- There are four sort work files.

SUM Control Statement

The SUM control statement deletes records with equal control fields and optionally sums specified numeric fields on those records. Equal keyed records are processed pair by pair. If numeric fields are to be summed, the data in the sum fields is added, the sum is placed in one of the records, and the other record is deleted. Provided arithmetic overflow does not occur, the SUM statement produces only one record per sort key in the output data set.

The SUM control statement cannot be used when FIELDS=COPY is specified on the SORT or MERGE control statement, or when FIELDS=COMPARE is specified on the MERGE control statement.

The format of the SUM control statement is illustrated below.

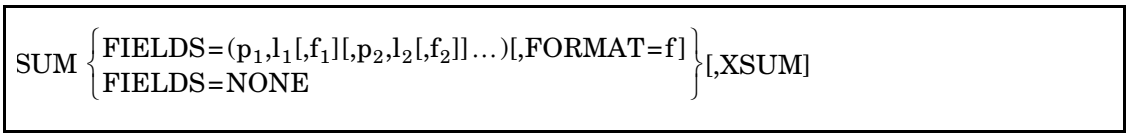


Figure 157. SUM Control Statement Format

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 or FIELDS=(NONE) to reduce the sorted data to one record per sort key without summing 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 integral number of bytes. Refer to Table 41 on page 2.247 for the permissible lengths.

f The format value indicates the data format. Fields with BI, FI, FL, PD and ZD formats can be summed. If all the sum fields have the same format, you can specify the format value once by using the FORMAT=f subparameter.

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 31 bytes

Table 41. Permissible Lengths for SUM Fields

XSUM Parameter (Optional)

Specify the XSUM parameter if you want the records deleted by SUM processing to be written to a file named SORTXSM (this name may be changed by the XSUMNM parameter in the OPTION statement). These records will be written to SORTXSM at the time of SUM processing. These records will not undergo OUTREC, E35, and OUTFIL processing as such processing occurs after SUM processing.

Characteristics of the SORTXSM file, such as BLKSIZE, can be specified via the XSUMFIL statement. The default is unblocked output.

The SORTXSM file will be sequenced in the same order as the SORTOUT file.

Note that XSUM may increase system resource 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.
- XSUM may require additional storage. Do not specify VSCORE below 512K.

Rules for Specifying Sum Fields

- If NOEQUALS is in effect, the record that is retained is determined arbitrarily. If EQUALS is in effect, the record that is retained is the first record read. The EQUALS parameter can be specified on the SORT, MERGE or OPTION control statement.
- A sort or merge control field cannot be summed. A portion of a control field cannot be included in a sum field.
- Sum fields cannot overlap each other.

- Non-sum fields remain unchanged and are retained from the record that contains the sum.
- If arithmetic overflow occurs during the summing of two records, those records are not summed and neither record is deleted. However, the sum process continues so that subsequent pairs of records with equal control fields can be summed, provided they do not cause overflow. To avoid arithmetic overflow, use the INREC statement to insert binary zeros, or X'F0's if the field is in ZD format, immediately before the sum field.
- 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.

Sample SUM Control Statements

The following SUM statement deletes equal-keyed records without summing numeric fields.

```
SUM FIELDS=NONE
```

Figure 158. Sample SUM Control Statement

Records with equal control fields will be deleted so that only one record is retained.

The following SUM statement sums two numeric fields on records with equal control fields.

```
SUM FIELDS=(20,4,32,4),FORMAT=PD
```

Figure 159. Sample SUM Control Statement

When the control fields are equal, this SUM statement sums 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 4. How to Use SyncSort Data Utility Features".

XDUPFIL Control Statement

The XDUPFIL control statement describes the XDUP (records deleted by DUPKEYS processing) output file. It is required when XDUP is specified in the DUPKEYS statement and you want any of the following:

- To specify an output blocksize
- VSAM output
- To reformat output records
- To use other parameters available in the XDUPFIL statement

The format and the parameters for the XDUPFIL statement are identical to the OUTFIL statement, except that the following parameters are not supported:

- EXIT
- FILES=
- FNAMES=

Refer to the “OUTFIL Control Statement” on page 2.141 for an explanation of the parameters that may be specified in an XDUPFIL statement.

XSUMFIL Control Statement

The XSUMFIL control statement describes the XSUM (records deleted by SUM processing) output file. It is required when XSUM is specified in the SUM statement and you want any of the following:

- To specify an output blocksize
- VSAM output
- To reformat output records
- To use other parameters available in the XSUMFIL statement

The format and the parameters for the XSUMFIL statement are identical to the OUTFIL statement, except that the following parameters are not supported:

- EXIT
- FILES=
- FNAMES=

Refer to the “OUTFIL Control Statement” on page 2.141 for an explanation of the parameters that may be specified in an XSUMFIL statement.

Chapter 3. Using the SyncSort Dictionary Feature

The Dictionary Feature

The SyncSort Dictionary Feature allows you to create symbolic 'dictionary_names' for frequently used fields or constants and use these dictionary_names in SyncSort control statements.

Using the Dictionary Feature has many benefits:

- Easier coding of sort control statements speeds development of sort applications and improves accuracy.
- More readable and understandable sort control statements facilitate debugging and adaptation to changes in the future.
- Reduction or elimination of changes to sort control statements when record layouts or user parameters change saves time and reduces errors.

To use the SyncSort Dictionary Feature, do the following:

- Build a dictionary using the dictionary statements. A dictionary statement creates a dictionary_name and associates it with a field specification (position, length, format) or a constant value.
- Activate the Dictionary Feature using the SYMNAMES parameter on the OPTION control statement. This parameter tells SyncSort the name and location of the dictionary to use.

- Use dictionary_names in sort control statements.

A dictionary consists of one or more dictionary statements. A dictionary can be catalogued as a VSE library member, or it can be placed immediately after the OPTION control statement in SYSIPT.

Each time a Dictionary Feature activated sort executes, it reads the dictionary statements from the VSE library or SYSIPT, and substitutes any dictionary_name found in the following control statements with the associated field specification or constant.

When a record layout changes, just modify the dictionary statements. The next sort execution will read in the updated dictionary statements and apply the new values during dictionary_name substitutions.

Activating the Dictionary Feature

Activate the SyncSort for z/VSE dictionary feature by specifying SYMNAMES on the SyncSort OPTION statement. If you wish to use the dictionary feature, the OPTION statement must be the first statement read by SyncSort. SYMNAMES tells SyncSort where to find the dictionary statements.

SYMNAMES=SYSIPT indicates that the dictionary statements are immediately after the OPTION statement in SYSIPT. In this case, the last dictionary statement must be followed by a statement with /* in column 1 and 2 followed by the remaining SyncSort control statements, such as SORT, RECORD etc. Note that this form of the SYMNAMES parameter cannot be used in a SyncSort execution from within another program such as an assembler program.

The following sample application illustrates the use of SYMNAMES=SYSIPT:

```
// EXEC PGM=SORT
 OPTION PRINT=ALL,ROUTE=LST,SYMNAMES=SYSIPT
*    Start of the Dictionary statements
Field1,1,10,ch             First field in data record
Field2,11,4,zd             Second field in data record
      .
      .
      .
Field40,251,15,ch          Last field in data record
*    End of the Dictionary statements
/*
 SORT FIELDS=(Field2,A,Field40,D),FILES=1,WORK=3
 RECORD TYPE=F,LENGTH=265
 INPFIL BLKSIZE=7950
 OUTFIL BLKSIZE=7950
 END
/*
```

Figure 160. Sample Application of SYMNAMES=SYSIPT

The SYMNAMES parameter of the OPTION statement can also indicate that the dictionary statements are contained in a VSE library member. In this case, the library member

must consist of fixed-length, 80 byte records. When SYMNAMES is used to specify a VSE library member, SYMNAMES has two possible formats.

- **SYMNAMES=library.sublib.member.type** tells SyncSort all the information that it needs to find the dictionary statements. In this case, SyncSort will read the dictionary statements from the library member: member.type in the VSE library: library.sublib, as shown in the following example:

```
// EXEC PGM=SORT
 OPTION PRINT=ALL,ROUTE=LST,SYMNAMES=(MAINLIB.MYSUB.DATAREC.CARDS)
 SORT FIELDS=(Field2,A,Field40,D),FILES=1,WORK=3
 RECORD TYPE=F,LENGTH=265
 INPFIL BLKSIZE=7950
 OUTFIL BLKSIZE=7950
 END
/*
```

Figure 161. Sample SYMNAMES=library.sublib.member.type

The library member DATAREC.CARDS in the above example might contain dictionary statements such as the following:

```
Field1,1,10,ch              First field in data record
Field2,11,4,zd              Second field in data record
      .
      .
      .
Field40,251,15,ch           Last field in data record
```

Figure 162. Sample Dictionary Statements

- **SYMNAMES=member.type** specifies only the member.type information. In this case, SyncSort will read the data dictionary statements from library member member.type in the first VSE library that SyncSort finds contains member.type. The libraries that SyncSort searches for member.type are specified by the SYMNLIB installation default or by the SYMNLIB parameter of the OPTION statement.

  The SYMNLIB parameter specifies a list of 1 to 14 VSE libraries that should be searched for the library member specified by SYMNAMES=member.type. The following shows the format for SYMNLIB:

SYMNLIB=(lib_1.$sublib_1$ [,lib_2.$sublib_2$]...[,lib_{14}.$sublib_{14}$])

Figure 163. SYMNLIB Format

SyncSort will attempt to find the member specified by SYMNAMES=member.type first in lib_1.$sublib_1$. If it is not found, SyncSort checks each lib.sublib in turn.

The following sample application uses SYMNAMES with SYMNLIB:

```
// EXEC PGM=SORT
 OPTION SYMNAMES=DATAREC.CARDS,SYMNLIB=(TESTLIB.TSTSUB.MAINLIB.MYSUB)
 SORT FIELDS=(Field2,A,Field40,D),FILES=1,WORK=3
 RECORD TYPE=F,LENGTH=265
 INPFIL BLKSIZE=7950
 OUTFIL BLKSIZE=7950
 END
/*
```

Figure 164. Sample Application of SYMNAMES with SYMNLIB

The library member DATAREC.CARDS in the above example might contain dictionary statements such as the following:

```
Field1,1,10,ch             First field in data record
Field2,11,4,zd             Second field in data record
      .
      .
      .
Field40,251,15,ch          Last field in data record
```

Figure 165. Sample Dictionary Statements

An additional statement is allowed within dictionary statements to enable embedding or concatenating dictionary sources. This is the SYMNAMES statement. The SYMNAMES statement has the same function as the SYMNAMES parameter on the OPTION statement. Either SYMNAMES=library.sublib.member.type or SYMNAMES=member.type may be specified. (SYMNAMES=SYSIPT may not be specified on the SYMNAMES statement.) The SYMNAMES statement may start with 0, 1 or more blanks followed by SYMNAMES= and the required library member information. Continuation of the SYMNAMES statement is not allowed. The maximum nesting level allowed using the SYMNAMES statement is 15.

The following sample application uses the SYMNAMES statement:

```
// EXEC PGM=SORT
 OPTION SYMNAMES=SYSIPT,SYMNLIB=(TESTLIB.TESTSUB,MAINLIB.MYSUB)
Field1,1,10,ch             First field in data record
Field2,11,4,zd             Second field in data record
      .
      .
      .
Field40,250,15,ch
SYMNAMES=DATAREC.CARDS
/*
 SORT FIELDS=(Field2,A,Field40,D),FILES=1,WORK=3
 RECORD TYPE=F,LENGTH=514
 INPFIL BLKSIZE=7710
 OUTFIL BLKSIZE=7710
 END
/*
```

Figure 166. Sample Application of SYMNAMES Statement

The library member DATAREC.CARDS in the above example might contain dictionary statements such as the following:

```
Field41,265,15,ch
SYMNAMES=EMBEDDED.CARDS
Field85,511,4,zd
```

Figure 167. Sample Dictionary Statements

When using the SYMNAMES statement to embed or concatenate dictionary members, you must be careful not to introduce an embedding loop. For example: member1 contains a SYMNAMES statement to read member2. Member2 contains a SYMNAMES statement indicating SyncSort should read member3. Member3 contains a SYMNAMES statement causing SyncSort to read member1. This is an embedding loop and is not allowed. The following shows sample dictionary statements in an embedded member:

```
Field42,280,24,ch
        .
        .
        .
Field84,500,11,ch
```

Figure 168. Sample Dictionary Statements in an Embedded Member

Specifying Dictionary Listing Output

The SYMNOUT parameter of the OPTION statement allows you to specify if you want the dictionary listing to be printed on SYSLST and/or SYSLOG or not at all:

- SYMNOUT=NONE prevents the printing of the dictionary listing.

- SYMNOUT=LST directs the output to SYSLST.
- SYMNOUT=LOG directs the output to SYSLOG
- SYMNOUT=LAL directs the output to both SYSLST and SYSLOG.

The dictionary listing consists of two parts.

- The first part is the user-provided dictionary statements.
- The second part is the dictionary table generated by SyncSort from the dictionary statements.

See also “OPTION Control Statement” on page 2.114 for additional details of the SYMNAMES, SYMNLIB, and SYMNOUT parameters.

Sample Dictionary Statements

This section provides examples to give you a quick understanding of dictionary statements. For details of syntax and usage, see “Dictionary Statement Format” on page 3.8.

The following shows the basic format of a dictionary statement:

```
dictionary_name,value comment
```

Figure 169. Basic Format of Dictionary Statement

where dictionary_name can be either:

- field_name
- constant_name

A field_name can be represented as:

position,length,format (p,l,f)
position,length (p,l)
position (p)

A constant_name can be represented as:

C'string', c'string', or 'string'
X'string' or x'string'
Y'string' or y'string'
B'string' or b'string'
n, +n, or -n.

As a quick introduction, here are some sample dictionary statements:

Example 1

```
Order#,10,8,ch
Catlog#,18,6,ch

Color,24,1,ch
    Grey,C'G'
    Red,C'R'
    red,C'r'
    Black,C'B'
    White,C'W'

Qty,25,3,ZD
Price,28,4,PD
```

Figure 170. Sample Dictionary Statements

This example uses leading blanks before some color dictionary_names to create indentation for clarity.

A blank statement is used before and after the "color" section. Such blank statements are ignored but can be printed.

Dictionary_names are case-sensitive. Thus, *Red* and *red* are separate dictionary_names.

Example 2

```
Street,38,45,ch
City,*,26,ch                City is "83,26,CH"
State,*,2,ch                State is "109,2,CH"
Zip,*,9,zd                  Zip is "111,9,ZD"
```

Figure 171. Example of Use of Asterisk to Replace Position (p)

In this example, note how position (p) is replaced with an asterisk (*) to indicate that the value of p should be the next position after the previous field. This is a powerful feature. Using an asterisk (*) for position allows you to map consecutive fields automatically. Thus, if a field specification changes, it is not necessary to calculate and specify the changed positions of subsequent fields.

Example 3

```
Name,5,40,ch
SKIP,20
Phone#,*,11,ch              Phone# is "65,11,CH"
```

Figure 172. Example of Use of SKIP Operator

In this example, the SKIP operator advances the next position by n bytes. Since the position of the next field is specified with an asterisk (*), the position will be calculated automatically.

Other operators, in addition to SKIP, are POSITION and ALIGN. POSITION,q or POSITION,dictionary_name resets the next position. ALIGN,H aligns the next position on a halfword, while ALIGN,F aligns it on a fullword and ALIGN,D aligns it on a doubleword.

Example 4

```
Date_of_Birth,10,6,Y2T      A 6-byte field in yymmdd format
DOB_YY,=,2,Y2C              Mapping byte 1,2 of Date_of_Birth
DOB_MM,*,=,ZD               Mapping byte 3,4 of Date_of_Birth
DOB_DD,*,=,=                Mapping byte 5,6 of Date_of_Birth
```

Figure 173. Example of Use of Equal Sign to Replace p,l, or f

In this example, an equal sign (=) is used instead of p,l, or f. The equal sign assigns the previous position,length or format to the equal sign, mapping one field onto another.

Although field_names and constant_names can usually be specified in any order, using the asterisk (*) or equal sign (=) forces a dependency on field order.

Note the use of comments in the above examples. As for SyncSort control statements, a blank after the value indicates the beginning of a comment. You can also use comment statements, which begin with an asterisk (*) in column 1.

Dictionary Statement Format

There are three basic types of dictionary statements:

- The constant_name statement
- The field_name statement
- The operator statement

These types of dictionary statements are described below in separate sections.

In addition, you can have a comment statement or a blank statement:

- A statement with an asterisk (*) in the first column is a comment statement. SyncSort will not process it, but it can be printed.

- A blank statement contains blanks in fields 1 - 80. SyncSort will not process it, but it can be printed.

The following shows the complete syntax for a dictionary statement:

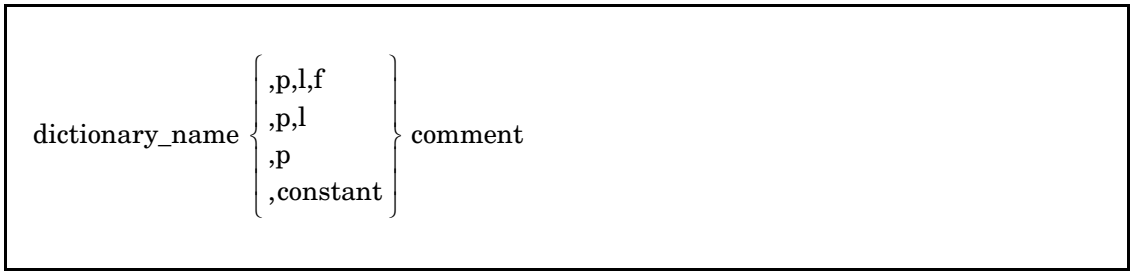


Figure 174. Dictionary Statement Syntax

where dictionary_name can be either:

- field_name

- constant_name

The following general rules apply to dictionary statements:

- The dictionary statement can start in any column, 1-80. However, a dictionary statement can only occupy one line; a continuation to a second line is not permissible.

- One or more blank spaces indicate the beginning of a comment. Characters following the blank space(s) are not processed by SyncSort but can be printed.

- A semicolon (;) can be used instead of a comma (,) to separate the dictionary_name and the value.

The following rules apply to dictionary_name:

- May consist of 1 to 50 EBCDIC characters.

- May be a combination of uppercase letters (A-Z), lowercase letters (a-z), numbers (0-9), the number sign (#), the dollar sign ($), the commercial at sign (@), the underscore (_), or the dash (-).

- May not have a number (0-9) or a dash (-) as the first character.

- Are case-sensitive; for example, 'Address', 'ADDRESS', and 'address' are treated as different fields.

- Cannot be a SyncSort reserved word. The following table lists the SyncSort reserved words:

A	DATE2P	M00 through M99	TE3
AC	DATE3	NONE	TE4
ADD	DATE3P	NUM	TIME
ALIGN	DATE4	OL	TIME1
ALL	DATENS	OR	TIME1P
AND	DC1	OT	TIME2
AQ	DC2	PAGE	TIME2P
ASL	DC3	PAGEHEAD	TIME3
AST	DE1	PD	TIME3P
AVG	DE2	PD0	TIMENS
BI	DE3	PDC	TM1
CH	DIV	PDF	TM2
CLO	DT1	POSITION	TM3
COMPARE	DT2	PSI	TM4
COPY	DT3	PZ	TS
COUNT	E	SEQNUM	UFF
CSF	F	SFF	VALCNT
CSL	FI	SKIP	VLEN
CST	FL	SS	X
CTO	FS	SUB	Y2x*
D	H	SUBCOUNT	Y2xx*
D1	HEX	TC1	Z
D2	LS	TC2	ZD
DATE	MAX	TC3	ZDC
DATE1	MIN	TC4	ZDF
DATE1P	MOD	TE1	ZSI
DATE2	MUL	TE2	
* x represents any character			

Table 42. SyncSort Reserved Words

All SyncSort reserved words are in uppercase. Thus, mixed case and lowercase forms of SyncSort reserved words are permissible as a dictionary_name. Similarly, dictionary statement operators such as POSITION, SKIP and ALIGN are in uppercase, thus mixed and lowercase forms of these words are permissible as a dictionary_name. For example, *position*, *Skip*, and *align* are all acceptable as a dictionary_name.

Some permissible dictionary_names:

cust_name
ADDR2

FAX#
num
Primary-SSN

The following are not permissible dictionary names:

100_Addr (begins with a number)
NUM (a reserved word)
ALIGN (an operator)

The following sections describe the three types of dictionary statements in detail:

- "The Constant_name Statement: Rules and Syntax" on page 3.11
- "The Field_name Statement: Rules and Syntax" on page 3.13
- "The Operator Statement: Rules and Syntax" on page 3.21

The Constant_name Statement: Rules and Syntax

The value of a constant may be a decimal number, a character string, a hexadecimal string, or a bit string.

A constant_name representing a specific value can be used for headings, titles, comparisons, etc. However, take care that the SyncSort control statement specifies the correct data format. For example, consider the the dictionary_name CONSR, defined by the following dictionary statement:

```
CONSR,B'00000010'
```

Figure 175. Example of constant_name in Dictionary Statement

Since CONSR is a binary field, it can be specified as such on the SyncSort control statement:

```
INCLUDE COND=(14,1,BI,EQ,CONSR)
```

Figure 176. Example of constant_name in Control Statement

If you specified CH as the data format, the control statement would be invalid.

The rules for fields specified in a control statement always apply to a field specified as a dictionary_name since internally SyncSort substitutes the actual field specification.

The following table describes the types of constant values:

Value Type	Description	Valid Examples	Invalid Examples
Character string	Valid formats: 'xx...x' C'xx...x' nC'xx...x' c'xx...x' nc'xx...x' where x is an EBCDIC character and n is a repetition factor for the string. Maximum length: 64 characters. To include a single apostrophe (') in a character string, use two single apostrophes ('), which count as 2 characters.	'+0.245' c'BOOK' C'O''DOOLE'	C'CITY'' (unnecessary extra apostrophe after Y) c'city (missing ending apostrophe)
Decimal number	Valid formats: n +n -n Maximum length is 1 to 15 significant digits. Decimal points are invalid.	+100 100 -500 00049	++100 (too many plus signs) 500- (minus sign in wrong place) 5.5 (decimal point not allowed)
Hexadecimal string	Valid formats: X'yy...yy' x'yy...yy' where yy represents any pair of hexadecimal digits. Maximum length is 32 pairs of hexadecimal digits. Hexadecimal digit are 0-9, A-F, or a-f.	X'F1C5' x'3fb91e' X'06'	X'F1H5' (H not valid hexadecimal digit) x'cf2' (unpaired hexadecimal digit 2)

Table 43. (Page 1 of 2) Types of Constant Values

Value Type	Description	Valid Examples	Invalid Examples
Bit string	Valid formats: B'bbbbbbbb...bbbbbbbb' b'bbbbbbbb...bbbbbbbb'. Each group of 8 bs represents the 8 bits that comprise one byte. Maximum length is 8 groups of 8 bits each. A bit is a 1, 0 or . (period).	b'11110000' B'11...0000' b'01101111'	b'0011' (only 4 bits, 8 needed) b'' (no bits specified) B'00001112' (invalid bit value 2)
Date constant	Valid formats: Y'LOW' Y'HIGH' Y'BLANKS' Y'x...x' where x...x represents 2 to 6 decimal digits.	Y'HIGH' Y'LOW' Y'980731' Y'012'	Y'1' (less than 2 digits) Y' ' (blank not valid digit) Y'1234567' (more than 6 digits) Y'BLANK' (should be 'BLANKS')

Table 43. (Page 2 of 2) Types of Constant Values

The Field_name Statement: Rules and Syntax

A field_name in a dictionary statement can be defined by its position in the record and its length and format type (p,l,f). Length and format type are optional.

The following are permissible field_names in a dictionary statement:

field_name,p,l,f
field_name,p,l
field_name,p

Take care that the format of the field name is acceptable on the SyncSort control statement. For example, consider the dictionary_name CITY, defined by the following dictionary statement:

```
CITY,10,29,CH
```

Figure 177. Example of field_name in Dictionary Statement

The following control statement specifying CITY would be valid because a CH field is permissible on the SORT control statement:

```
SORT FIELDS=(CITY,A)
```

Figure 178. Example of field_name in Control Statement

However, the dictionary_name CITY could not be used on a SUM control statement because CH is not a valid format with SUM.

The rules for fields specified in a control statement always apply to a field specified as a dictionary_name since internally SyncSort substitutes the actual field specification.

Note that you can specify a field_name with p, l, and f, then use the field_name in a control statement that only requires p and l. SyncSort will substitute p and l during processing and ignore the f specification. For example, consider the following dictionary statements:

```
Account#,1,12,ch
CheckNumber,*,4,zd
CheckDate,*,6,y2t
CheckAmount,*,6,pd
```

Figure 179. Example of p,l,f in Dictionary Statements

Suppose you use these field_names in the following control statements:

```
OMIT   COND=(CheckDate,LT,Y'980101')
SORT   FIELDS=(Account#,A,CheckDate,A)
SUM    FIELDS=(CheckAmount),FORMAT=PD
OUTFIL OUTREC=(Account#,CheckDate,CheckAmount,M23)
```

Figure 180. Example of field_names in Control Statements

Based on the field_names defined in the dictionary statements, SyncSort will make the following substitutions:

- In the OMIT statement, 'CheckDate' is substituted with '17,6,Y2T'.
- In the SORT statement, 'Account#' is substituted with '1,12,CH' and 'CheckDate' is substituted with '17,6,Y2T'.
- In the SUM statement, 'CheckAmount' is substituted with '23,6'.
- In the OUTFIL OUTREC statement, 'Account#' is substituted with '1,12', 'CheckDate' is substituted with '17,6,Y2T', and 'CheckAmount' is substituted with '23,6,PD'.

Here are the resulting control statements:

```
OMIT   COND=(17,6,Y2T,LT,Y'980101')
SORT   FIELDS=(1,12,CH,A,17,6,Y2T,A)
SUM    FIELDS=(23,6),FORMAT=PD
OUTFIL OUTREC=(1,12,17,6,Y2T,23,6,PD,M23)
```

Figure 181. Example of p,l in Control Statements

The following three subsections describe the rules for specifying position (p), length (l) and format (f) in field_name dictionary statements.

Specifying Position (p) in Field_name Dictionary Statements

The following are the rules for specifying position (p) in field_name dictionary statements:

- p can be a number from 1 to 65532 whenever p,l or p,l and f are used. However, if position (p) is used alone (for example, CITY,20), the p can not be more than 15 significant digits.

  Note that any value of p greater than 65532 may cause a syntax error when it is processed as a position.

- Alternately, p can be in the form of m.n for specifying a bit position, where m is a number from 1 to 65532 and n is a number between 0 and 7. Note that m.0 is equivalent to m.

- p can be an asterisk (*), which indicates that the next position should be assigned to p. Since l represents length, the next position is set to p + l each time the field_name for p,l,f or p,l is read. If the next position has not yet been determined, as when the asterisk is used in the first field_name, then p defaults to 1 (one).

  The dictionary table, which can be printed on request, displays the actual positions assigned to p when the asterisk is used for p. For example, consider the following OPTION statement:

```
OPTION SYMNAMES=PARTS.LAYOUT
```

Figure 182. Sample OPTION Statement

Suppose the members contain field_name statements as follows:

The PARTS.LAYOUT member contains:

```
Part#,1,8,ch
Desc$,*,20,ch
SYMNAMES=MFC.LAYOUT
SYMNAMES=STOCK.LAYOUT
```

The MFC.LAYOUT member contains:

```
Mfc_code,*,6,ch
Mfc_part#,*,10,ch
```

The STOCK.LAYOUT member contains:

```
Stock_Shelf#,*,4,bi
Stock_Bin#,*,4,bi
Stock_Qty,*,4,pd
```

Figure 183. Sample field_name Statements in Members

SyncSort will print the following dictionary table:

```
Part#,1,8,CH
Desc$,9,20,CH
Mfc_code,29,6,CH
Mfc_part#,35,10,CH
Stock_Shelf#,45,4,BI
Stock_Bin#,49,4,BI
Stock_Qty,53,4,PD
```

Figure 184. Sample Dictionary Table

Note that using the asterisk (*) for position (p) enables you to:

- Map consecutive record fields without having to calculate their actual positions.
- Map new fields inserted between other fields without having to modify the p values of existing or inserted fields.
- Map contiguous fields in concatenated field name sets.

Field_names from all three field_name sets in the above example could be used in SyncSort control statements as follows:

```
OUTFIL FNAME=ORDER,INCLUDE=(Stock_Qty,LT,100),
                   OUTREC=(Part#,10X,Mfc_code,5X,Mfc_part#)
OUTFIL FNAME=INTEL,INCLUDE=(Mfc_code,EQ,C'INTEL'),
                   OUTREC=(Part#,10X,Stock_Shelf#,LENGTH=6,
                                 8X,Stock_Bin#,LENGTH=6,
                                 8X,Stock_Qty,LENGTH=8)
```

Figure 185. Sample field_names in Control Statements

- The equal sign (=) can be used instead of p to indicate that the previous position should be assigned to p. If the previous position has not yet been determined, an error message will be issued. Each time the field_name for p,l,f (or p,l) is read, the previous position is set to p.

  Note that the value of the previous position can also be modified by a POSITION operator statement. See "Using POSITION in Operator Statements" on page 3.22.

  If the dictionary table is printed, it will display the actual positions that result from the = specification. For example, consider the following dictionary statements:

```
POSITION,40
Payment_type,=,1,Bi
@cash$,B'......00'
@check,B'......01'
@credt,B'......1.'
Teller,*,20,Ch
Check#,=,4,Zd
CCard#,=,12,Zd
```

Figure 186. Sample Dictionary Statements

SyncSort will print the following dictionary table:

```
Payment_type,40,1,BI
@cash$,B'......00'
@check,B'......01'
@credt,B'......1.'
Teller,41,20,CH
Check#,41,4,ZD
CCard#,41,12,ZD
```

Figure 187. Sample Dictionary Table

Before using = for p, ensure that the previous position is the one you want. If you happen to add a new field_name with a different position and insert that field_name in front of a field_name that uses = for p, the value of the previous position will be wrong. In this case you will need to use the actual position value instead of = .

Specifying Length (l) in Field_name Dictionary Statements

The length (l) indicator, can be an equal sign (=), a number from 1 to 65532, or a bit length in the form of m.n, with m a number from 0 to 65532 and n a number from 0 to 7, while m and n cannot both be zero. Using an equal sign (=) in place of l assigns the previous length to l. If the previous length has not yet been determined when = is read, SyncSort will issue an error message.

If the dictionary table is printed, it will display the actual lengths that were assigned when = was specified for l. For example, consider the following definition statements:

```
Student_name,1,40,ch
Test_score_1,*,4,zd
Test_score_2,*,=,zd
Test_score_3,*,=,zd
```

Figure 188. Sample Definition Statements

The dictionary table will reflect substitutions made by SyncSort:

```
Student_name,1,40,CH
Test_score_1,41,4,ZD
Test_score_2,45,4,ZD
Test_score_3,49,4,ZD
```

Figure 189. Sample Dictionary Table

Before using = for length (l), ensure that the previous length is what you intended. If you happen to add a new field_name with a different length and insert that field_name in front of a field_name that uses = for l, the value of the previous length will be wrong. In this case you will need to use the actual length value instead of = .

Specifying Format (f) in Field_name Dictionary Statements

The format (f) indicator, can be an equal sign (=) or a specific format. The following formats are permissible:

AC, AQ, ASL, AST, BI, CH, CLO, CSF, CSL, CST, CTO, FI, FL, FS, LS, OL, OT, PD, PD0, PSI, SS, TS, Y2B, Y2C, Y2D, Y2DP, Y2ID, Y2IP, Y2P, Y2PP, Y2S, Y2T, Y2TP, Y2U, Y2UP, Y2V, Y2VP, Y2W, Y2WP, Y2X, Y2XP, Y2Y, Y2YP, Y2Z, ZD, ZSI

Formats can be specified using uppercase, lowercase, or mixed case letters.

When either p or l is specified in the bit form (m.n) and the bit number is not zero (n≠0), then the only valid format is BI.

As with p and l, an equal sign (=) indicates that the previous format should be assigned to f. If the previous format has not yet been determined when an = sign is used for f, SyncSort issues an error message.

If the dictionary table is printed, it will display the actual formats SyncSort substituted for =. For example, consider the following dictionary statement:

```
Student_name,1,40,ch
Test_score_1,*,4,zd
Test_score_2,*,=,=
Test_score_3,*,=,=
```

Figure 190. Sample Dictionary Statement

After processing, the substitutions will be reflected in the dictionary table:

```
Student_name,1,40,CH
Test_score_1,41,4,ZD
Test_score_2,45,4,ZD
Test_score_3,49,4,ZD
```

Figure 191. Sample Dictionary Table

Before using = for f, ensure that the previous format is what you intend. If you happen to add a new field_name with a different format and insert that field_name in front of a field_name that uses = for f, the value of previous format will be wrong. In this case you will need to change the = to the actual format.

If records are rearranged (for example by using E15, E35, INREC, or OUTREC), be sure to use field_names that map to the new positions of your fields. For example, consider the following dictionary statement:

```
Name1,1,5,ZD
Name2,*,8,ZD
Name3,*,3,ZD
Name4,*,7,ZD
```

Figure 192. Sample Dictionary Statement

Suppose the following INREC control statements is used:

```
INREC FIELDS=(Name2,Name4)
```

Figure 193. Sample INREC Control Statement

Only Name2 and Name4 will appear in the resulting records. If you want to use field_names from the rearranged records (for example with a SORT statement), you will need to use dictionary_statements that map the field_names to the rearranged records. For example:

```
New_Name2,1,8,ZD
New_Name4,*,7,ZD
```

Figure 194. Sample Dictionary Statements

If the rearranged fields are given unique names, as in the example above, you can concatenate the old and new dictionary statements and use both the old and new dictionary_names, as follows:

```
INREC FIELDS=(Name2,Name4)
SORT FIELDS=(New_Name2,A,New_Name4,A)
```

Figure 195. Example of Dictionary Statements in Control Statements

Alternative Field_name Syntax

SyncSort allows the following alternative form of the field_name statement:

field_name, { * / = } { + / - } n

Figure 196. Alternative Form of field_name Statement

where n is a number from 1 to 65532.

This syntax form is useful for defining a field_name for record length to be used in the RECORD control statement. For example:

```
field_name1,1,10,CH
  .
  .
  .
  .
field_namen,*,10,CH
rec_len,*-1
```

Figure 197. Sample Dictionary Statements

You would then use rec_len in the RECORD control statement, as follows:

```
RECORD TYPE=F,LENGTH=rec_len
```

Figure 198. Example of field_name in RECORD Control Statement

The Operator Statement: Rules and Syntax

The Operator Statement controls the position of fields you are mapping. Following is the format of an operator statement:

```
operator,value comment
```

Figure 199. Operator Statement Format

where operator represents:

POSITION Specifies a starting position.

SKIP Skips unwanted positions.

ALIGN Aligns fields on boundaries.

Syntax rules for the operator statement are the same as for the other types of dictionary statement. However, one rule applies specifically to the operator statement: To be treated as an operator statement, the operator must be all uppercase. The following are permissible operator statements:

```
POSITION,q
POSITION,field_name
SKIP,n
ALIGN,H
ALIGN,F
ALIGN,D
```

The following subsections describe the three operators: POSITION, SKIP, and ALIGN.

Using POSITION in Operator Statements

There are two forms of the operator statement with POSITION:

- POSITION,q
- POSITION,field_name

The first form of the OPERATOR statement (POSITION,q) sets the next position and the previous position to q:

- The next position is used when an asterisk (*) replaces p in a field_name statement.
- Previous position is used when an equal sign (=) replaces p in a field_name statement.

q can be any number from 1 to 65532, or it can be in the form of m.n for specifying a bit position, with m being a number from 1 to 65532 and n being a number from 0 to 7. Following a POSITION,q statement, either an asterisk (*) or an equal sign (=) can be used for position (p) in the next field_name statement.

Consider the following example, where POSITION,q is used with other dictionary statements:

```
POSITION,45
Volser,*,8,ch
```

Figure 200. Example of POSITION,q in Dictionary Statements

If the dictionary table is printed, it will reflect the substitutions:

```
Volser,45,8,CH
```

Figure 201. Sample Dictionary Table

The second form of the operator statement (POSITION,field_name) sets the next position and the previous position to the position of the specified field name:

- The next position is used when an asterisk (*) replaces p in a field_name statement.

- The previous position is used when an equal sign (=) replaces p in a a field_name statement.

Following a POSITION,field_name statement, either an * or an = can be used for p in the next field_name statement.

The field_name used with the POSITION operator can be any previously defined field_name. As a result, POSITION,field_name functions like the assembler ORG instruction, allowing you to map different fields onto the same area. Consider the following example, where POSITION,field_name is used with other dictionary_statements:

```
Filename,1,8,ch
Filetype,*,8,ch
Filemode,*,2,ch
POSITION,Filename
Filespec,*,18,=
```

Figure 202. Example of POSITION,field_name in Dictionary Statements

SyncSort will print the following dictionary table:

```
Filename,1,8,CH
Filetype,9,8,CH
Filemode,17,2,CH
Filespec,1,18,CH
```

Figure 203. Sample Dictionary Table

Using SKIP in Operator Statements

The operator statement has the following form with SKIP:

```
SKIP,n
```

Figure 204. SKIP Format in Operator Statement

where n can be any number from 1 to 65532, or it can be in the form of u.v for specifying a bit length, with u being a number from 0 to 65532 and v being a number from 0 to 7 (while u and v cannot both be zero).

SKIP,n increases the next position by n bytes. As described above, the next position is used when an asterisk (*) replaces p in a field_name statement.

Consider the following example, which uses SKIP,n with other dictionary_statements:

```
Make,1,10,ch
SKIP,4
  pp1,=              (Last position not changed by SKIP)
Model,*,10,ch
```

Figure 205. Example of SKIP,n in Dictionary Statements

SyncSort will print the following dictionary table:

```
Make,1,10,CH
pp1,1
Model,15,10,CH
```

Figure 206. Sample Dictionary Table

Using ALIGN in Operator Statements

The ALIGN operator statement has three forms:

ALIGN,B
ALIGN,H
ALIGN,F
ALIGN,D

ALIGN,B aligns the next position on a byte boundary, for example 1, 2, 3, etc. As described above, the next position is used when an asterisk (*) replaces p in a field_name statement.

The following example uses ALIGN,B with other dictionary_statements:

```
ZD1_zone,5.0,0.4,BI
ALIGN,B
Ten_bits,*,1.2,BI
ALIGN,B
nxt_byte,*,2,ch
```

Figure 207. Example of ALIGN,B in Dictionary Statements

SyncSort will print the following dictionary table:

```
ZD1_zone,5,0.4,BI
Ten_bits,6,1.2,BI
nxt_byte,8,2,CH
```

Figure 208. Sample Dictionary Table

ALIGN,H aligns the next position on a halfword boundary, for example 1, 3, 5, etc. As described above, the next position is used when an asterisk (*) replaces p in a field_name statement.

The following example uses ALIGN,H with other dictionary_statements:

```
box_1,1,1,BI
ALIGN,H
box_2,*,1,BI
ALIGN,H           (Last position not changed by ALIGN)
sel_2,=,1,BI
```

Figure 209. Example of ALIGN,H in Dictionary Statements

SyncSort will print the following dictionary table:

```
box_1,1,1,BI
box_2,3,1,BI
sel_2,3,1,BI
```

Figure 210. Sample Dictionary Table

ALIGN,F aligns the next position on a fullword boundary, for example 1, 5, 9, etc. As described above, the next position is used when an asterisk (*) replaces p in a field_name statement. Uppercase (F) and lowercase (f) are both acceptable.

The following example uses ALIGN,F with other dictionary_statements:

```
DIVISION,34,3,FI
ALIGN,F              (already aligned)
DEPT_1,*,3,FI
ALIGN,F
DEPT_2,*,3,FI
```

Figure 211. Example of ALIGN,F in Dictionary Statements

SyncSort will print the following dictionary table:

```
DIVISION,34,3,FI
DEPT_1,37,3,FI
DEPT_2,41,3,FI
```

Figure 212. Sample Dictionary Table

ALIGN,D aligns the next position on a doubleword boundary, for example 1, 9, 17, etc. As described above, the next position is used when an asterisk (*) replaces p in a field definition statement. Uppercase (D) and lowercase (d) are both permissible.

The following example uses ALIGN,D with other dictionary statements:

```
Account#,42,8,ch
ALIGN,D
Balance,*,8,csL
```

Figure 213. Example of ALIGN,D in Dictionary Statements

SyncSort will print the following dictionary table:

```
Account#,42,8,CH
Balance,57,8,CSL
```

Figure 214. Sample Dictionary Table

Using Dictionary_names in SyncSort Control Statements

Field_names can be used in the following SyncSort control statements:

DUPKEYS
INCLUDE
INPFIL
INREC
JOINKEYS
MERGE
OMIT
OUTFIL
OUTREC
REFORMAT
SORT
SUM
XDUPFIL
XSUMFIL

You can use a field_name wherever you would specify a position, length, and format (p,l,f or p,l or p). In addition, a special form of field_name can be used with the LENGTH parameter of the RECORD statement. (See “Alternative Field_name Syntax” on page 3.20.)

Constant_names can be used in the following control statements:

INCLUDE
INPFIL
INREC
JOINKEYS
OMIT
OUTFIL
OUTREC
XDUPFIL
XSUMFIL

You can use a constant_name wherever you would specify a constant (X'nn...', B'bbbb,...',C'ccc...', Y'xx...'), that is:

- A constant compared to a field in the data record (INCLUDE/OMIT)
- A constant inserted into a data record (INREC/OUTREC, OUTFIL, XSUMFIL).
- A constant used in a HEADERn/TRAILERn parameter, where n=1, 2, or 3 (OUTFIL, XSUMFIL).

When substituting field_names in control statements, SyncSort checks the context in which each field_name appears and determines which field specification is appropriate: p,l,f or only p,l. A field_name substitution results in position and length (p,l) under the following conditions:

- There is a separate FORMAT=parameter present in the control statement.
- A format is explicitly specified for the field; that is, the field_name is followed by a format specification.
- The field_name appears in HEADERn/TRAILERn (where n=1, 2, or 3) in an OUTFIL/XSUMFIL statement but outside a TOTAL/SUBTOTAL/MIN/MAX/AVG subparameter.
- The field_name appears in an INREC/OUTREC statement or in the OUTREC parm of an OUTFIL/XSUMFIL statement, with the following exceptions:
  - The field_name is followed by an EDIT mask (Mnn), EDIT=parameter, SIGN=parameter, or LENGTH=parameter.
  - The field_name is enclosed in parentheses.
  - The field_name is an operand of an arithmetic operation (ADD/SUB/MUL/DIV/MOD/MIN/MAX).

- The field_name is a Y2x field and it is not followed by H, F, D, HEX, or CHANGE=parameter.

Note that you cannot use a dictionary_name for any of the following:

- EDIT parameter
- DATE parameter
- Replacement length in CHANGE=parameter

Error Handling for Dictionary Statements

SyncSort will scan each dictionary statement for errors. When the first error is detected, an error message is printed. If appropriate, a marker will print just below the Dictionary statement, near the error. SyncSort stops scanning at the first error, then resumes with the next Dictionary statement.

Once an error has been detected, positions calculated with the use of an asterisk (*) for p or with the POSITION,field_name statement in subsequent dictionary statements will not be checked for errors. If an error is detected in any Dictionary statement, SyncSort terminates processing after all Dictionary statements are scanned.

If SyncSort detects an error in a control statement while substitution is taking place, it may respond in either of the following ways:

- Print the statement that was in error, followed by a corresponding error message and marker (if appropriate), then continue with the next statement and terminate when all substitutions have been completed.
- Stop the substitution for the statement in error and continue processing, letting subsequent processing handle the error. If this occurs, the original field or constant name rather than the substituted value may be displayed in a translated statement.

If the substitution process is successful, SyncSort will substitute values for field_names or constant_names wherever they are allowed. If substituted values prove invalid for a particular statement or operand, it will be detected after the substitution has been performed, making it easier to determine the source of the error.

Chapter 4. How to Use SyncSort 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 AVG, MAX, MIN, or 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 examples of the JOIN facility, you may refer to Syncsort's booklet, *Exploiting SyncSort for z/VSE: JOIN*.

Sample Data Utility Applications

The following chart lists applications that demonstrate SyncSort's features.

Feature	Application	Page
Selecting Input Records	Including Relevant Records Omitting Irrelevant Records	4.4 4.5
Selecting Relevant Fields from the Input Records	Selecting a Number of Fields from Longer Records Eliminating Irrelevant Data Field(s) Selecting Fields from Variable-Length Records	4.8 4.9 4.11
Combining Records within a File	Combining Records and Placing AVG, MAX, MIN, or SUM Values in Retained Records Eliminating Duplicate Records	4.14 4.15
Making Output Records Printable and Easy to Read	Reordering the Positions of Record Fields Inserting Blanks and Repositioning Record Fields Inserting Binary Zeros Converting Unprintable Data to Readable Form Converting Unprintable Data to Hexadecimal Format Converting and Editing Unprintable Data Putting a Data Field in Standard Format Printing Input Records	4.18 4.20 4.22 4.23 4.25 4.27 4.29 4.34
Dividing a Report into Sections	Dividing Output into Sections	4.34
Writing Headers and Trailers for a Report	Writing a Title Page for a Report Writing a Page Header Writing a Section Header Using a Header to Eliminate Duplication Information within a Section Writing a Report Trailer or Summary Writing a Page Trailer	4.38 4.40 4.41 4.42 4.44 4.45
Totaling and Subtotaling Data	Totaling Data at the End of a Report Subtotaling Data at the End of a Page Totaling Data at the End of a Section	4.47 4.49 4.50
Counting Data Records	Obtaining a Count of Data Records Obtaining a Cumulative, or Running, Count of Data Records	4.53 4.54
Creating Multiple Output Files	Generating Several Output Files with Different Information Writing Identical Output Files to Different Devices	4.58 4.60

Table 44. Index to Data Utility Applications

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. Alternately, 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 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 in Figure 215.

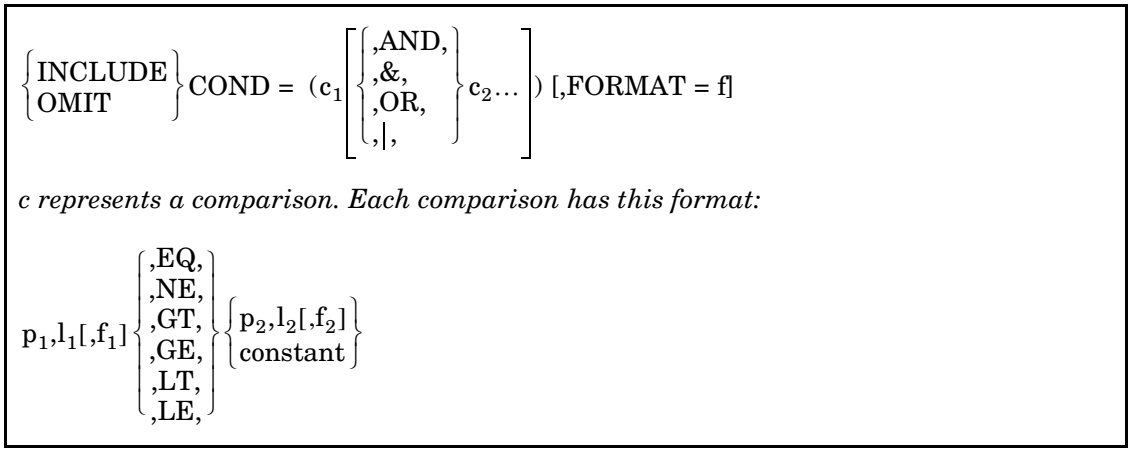


Figure 215. INCLUDE/OMIT Format

INCLUDE Use INCLUDE to specify the conditions for including records.

OMIT Use OMIT if you find it easier to specify the conditions for omitting records.

p_n,l_n[,f_n] Specify record fields by starting position (p), length in bytes (l), and the data format (f).

EQ, NE, etc. Use the relational operators (EQ, NE, GT, GE, LT, LE) to specify the relations upon which conditions are based.

constant Constants may be specified as a decimal of any length with an optional + or -, a hexadecimal coded as X'hh...hh', or a literal string coded as C'literal string'.

AND/&/OR/| Use these connectives when specifying more than one comparison. AND conditions are evaluated before OR conditions unless you use parentheses, in which case the evaluation of the conditions proceeds from the innermost to the outermost parentheses. Only one level of parentheses is permitted.

FORMAT You may use this parameter when the data format of all the record fields is identical.

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 216 and a sample record is given in Figure 217.)

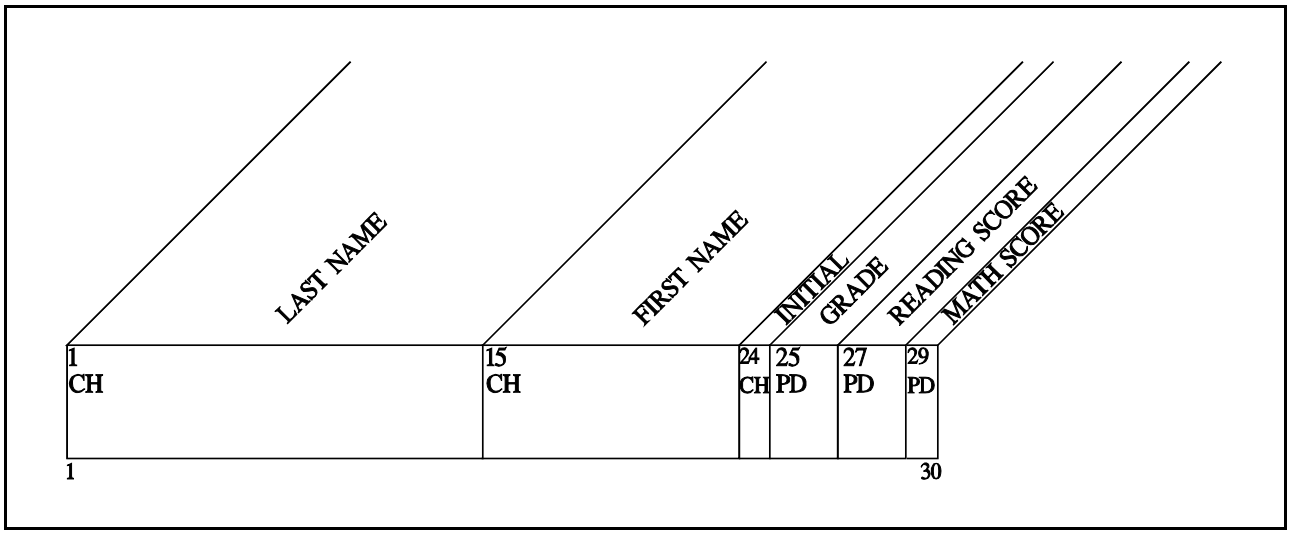


Figure 216. Input Record Layout

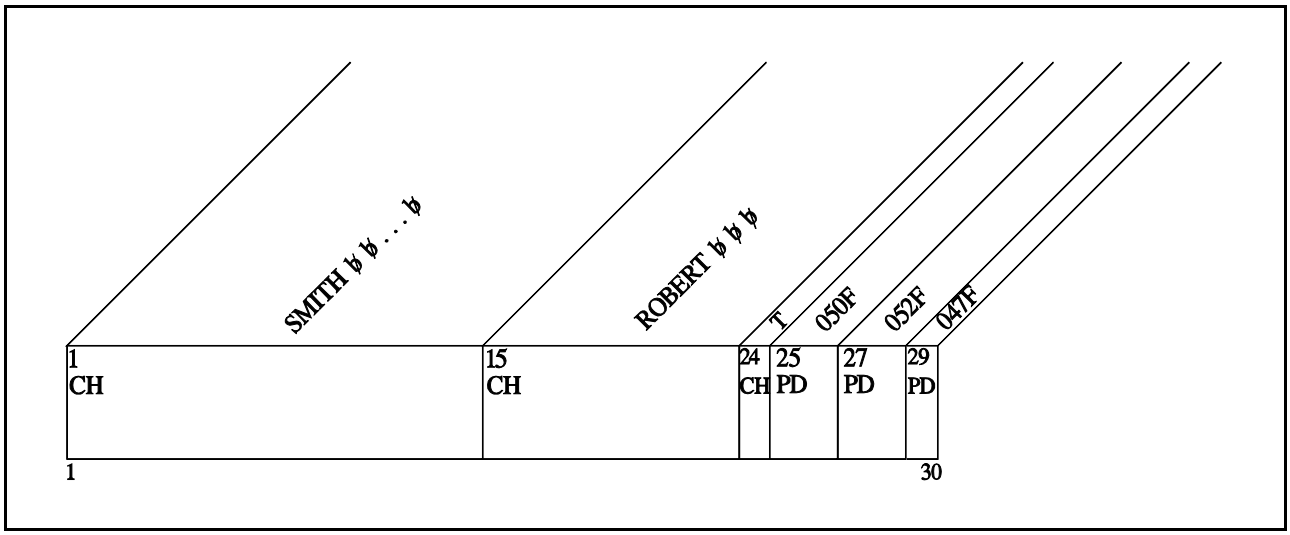


Figure 217. Sample Student Record

To generate the list, the following is coded.

```
// JOB      SUBLEV                    Signals the Start of Job
// ASSGN    SYS001,SYSLST             Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'             Assigns Logical Unit to Tape
// TLBL     SORTIN1,'STUDENTS'        Contains Input Tape File
                                      Information
// ASSGN    SYS003,X'251'             Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1                   Contains SORTWK File Information
// EXTENT   SYS003                    Defines Direct Access Area for SORTWK
// EXEC     SORT                      Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=30          Describes Input Records
   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 218. 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 218) 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 lay-out is given in Figure 219 and a sample input record is given in Figure 220.)

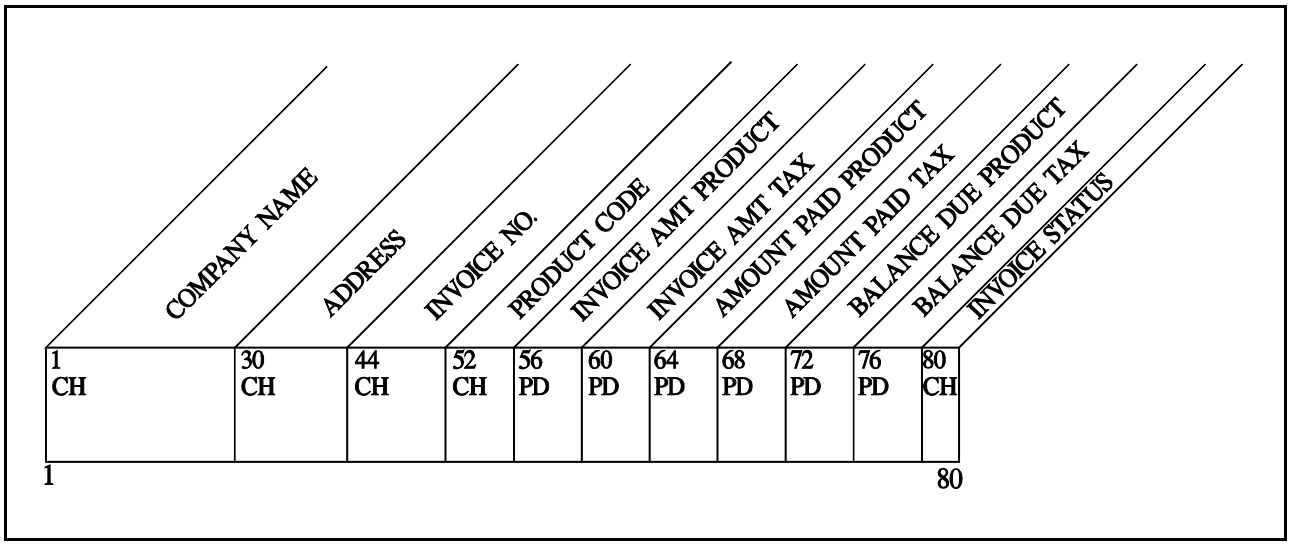


Figure 219. Input Record Layout

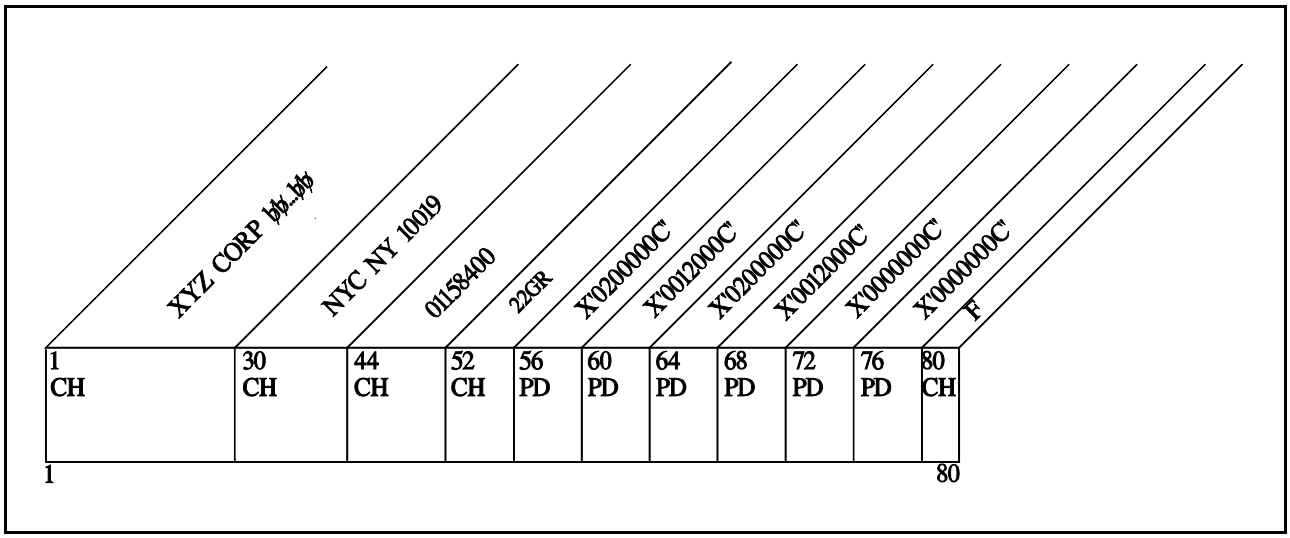


Figure 220. Sample Input Record

To produce this list of customers selected from the masterfile, the following is coded.

```
// JOB       OUTPAY                    Signals the Start of Job
// ASSGN     SYS001,SYSLST             Assigns Logical Unit to Printer
// ASSGN     SYS002,X'181'             Assigns Logical Unit to Tape
// TLBL      SORTIN1,'NEWINV'          Contains Input Tape File
                                       Information
// ASSGN     SYS003,X'251'             Assigns Logical Unit to SORTWK
                                       Device
// DLBL      SORTWK1                   Contains SORTWK File Information
// EXTENT    SYS003                    Defines Direct Access Area for
                                       SORTWK
// EXEC      SORT                      Signals Beginning of Sort Program
   RECORD    TYPE=F,LENGTH=80          Describes Input Records
   OMIT      COND=(80,1,CH,EQ,C'F')    Omits Records
   SORT      FIELDS=(1,29,CH,A)        Sorts Records
```

Figure 221. 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 221) 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.

To select relevant record fields, use the INREC statement as described below:

```
INREC FIELDS=(p1,l1[,p2,l2,...,pn,ln])
```

Figure 222. Basic INREC Statement Format

For the complete format of the INREC control statement, see "INREC Control Statement" on page 2.83.

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 223.)

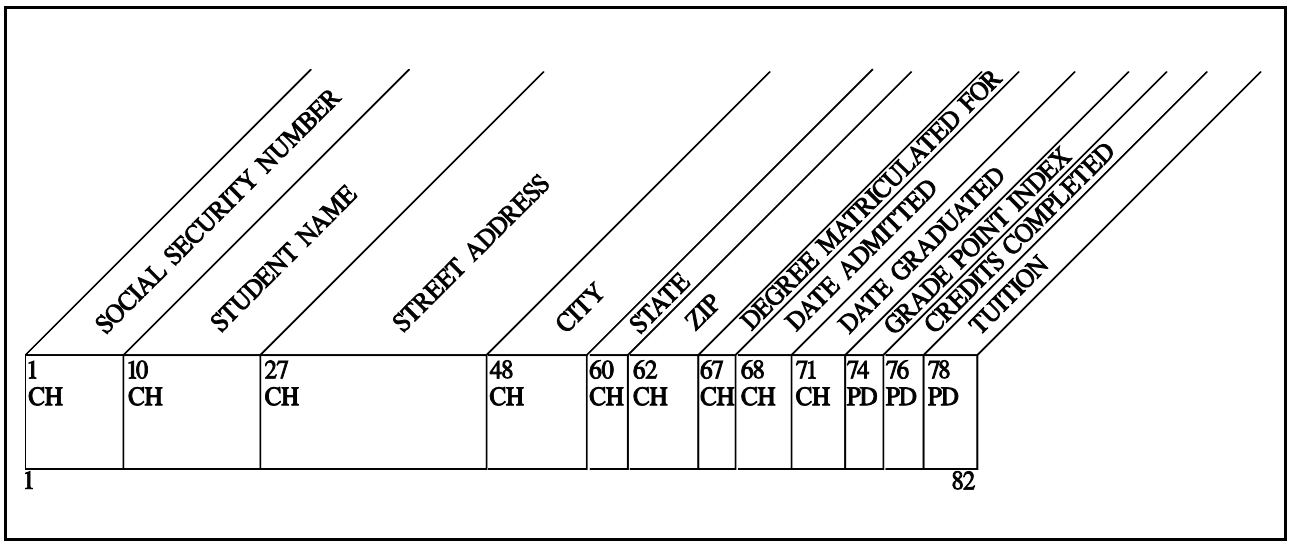


Figure 223. Input Record Layout

To include only the relevant fields and generate the ranked list of students, the following is coded.

```
// JOB      RANK                    Signals the Start of Job
// ASSGN    SYS001,SYSLST           Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'           Assigns Logical Unit to Tape
// TLBL     SORTIN1,'STUDENTS'      Contains Tape File Information
// ASSGN    SYS003,X'251'           Assigns Logical Unit to SORTWK
                                    Device
// DLBL     SORTWK1                 Contains SORTWK File Information
// EXTENT   SYS003                  Defines Direct Access Area for
                                    SORTWK
// EXEC     SORT                    Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=82        Describes Input Records
   INREC    FIELDS=(1,9,            Selects Record Fields
                    74,2)
   SORT     FIELDS=(10,2,PD,D)       Sorts Records
```

Figure 224. JCL and Required Control Statements

Figure 225 shows the input record after INREC processing.

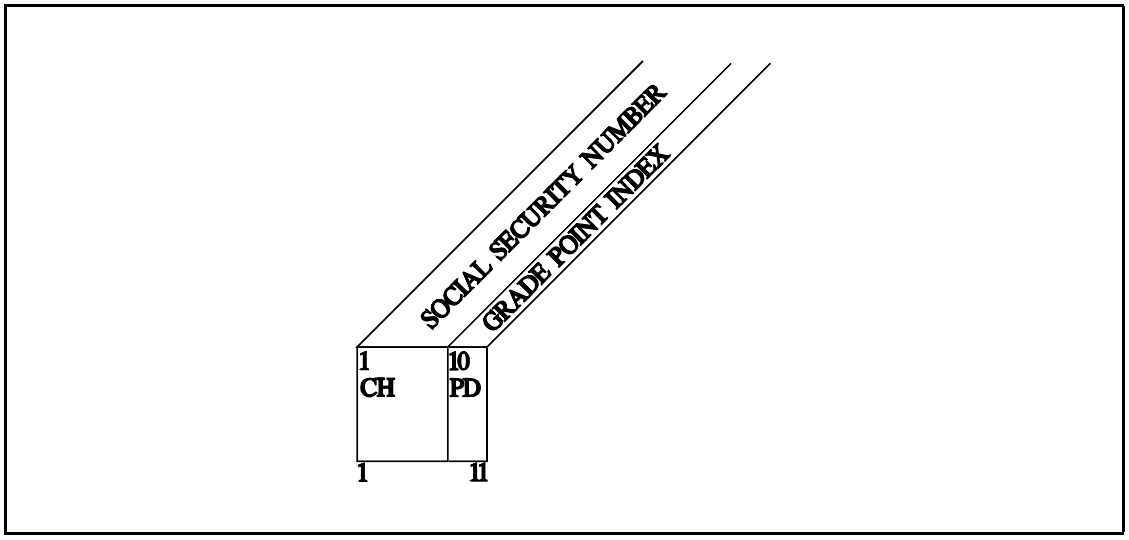


Figure 225. Form of Post-INREC Record

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 226.)

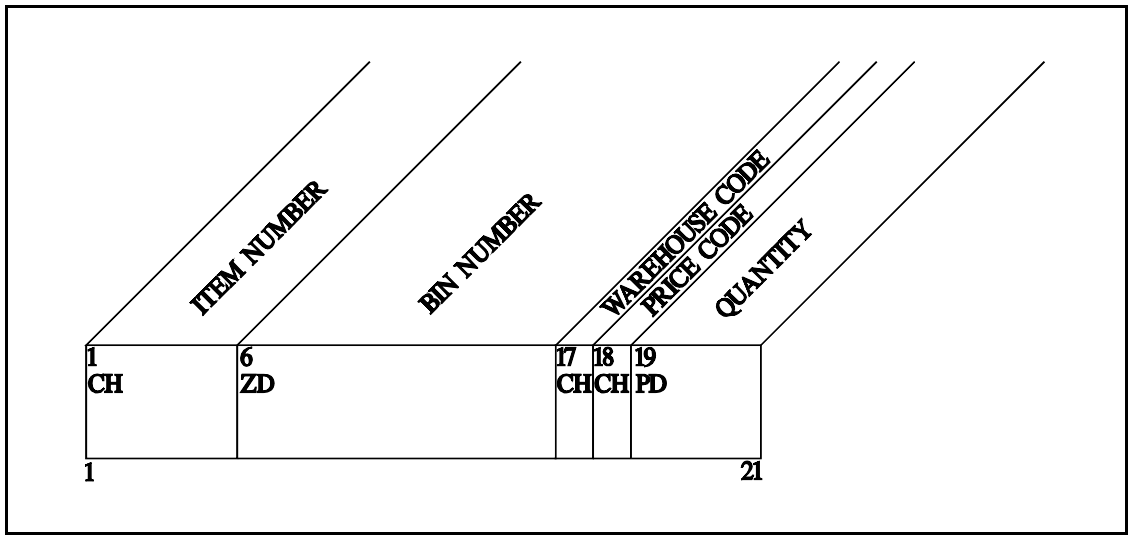


Figure 226. INPUT Record Layout

To eliminate the Price Code field and generate the inventory list, the following is coded.

```
// JOB      INVENTR                Signals the Start of Job
// ASSGN    SYS001,SYSLST          Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'          Assigns Logical Unit to Tape
// TLBL     SORTIN1,'WAREHSE'      Contains Input Tape File Information
// ASSGN    SYS003,X'251'          Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1                Contains SORTWK File Information
// EXTENT   SYS003                 Defines Direct Access Area for SORTWK
// EXEC     SORT                   Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=21       Describes Input Records
   INREC    FIELDS=(1,17,          Selects Record Fields
                    19,3)
   SORT     FIELDS=(1,5,CH,A)      Sorts Records
                   .
```

Figure 227. JCL and Required Control Statements

Figure 228 shows the input record after INREC processing.

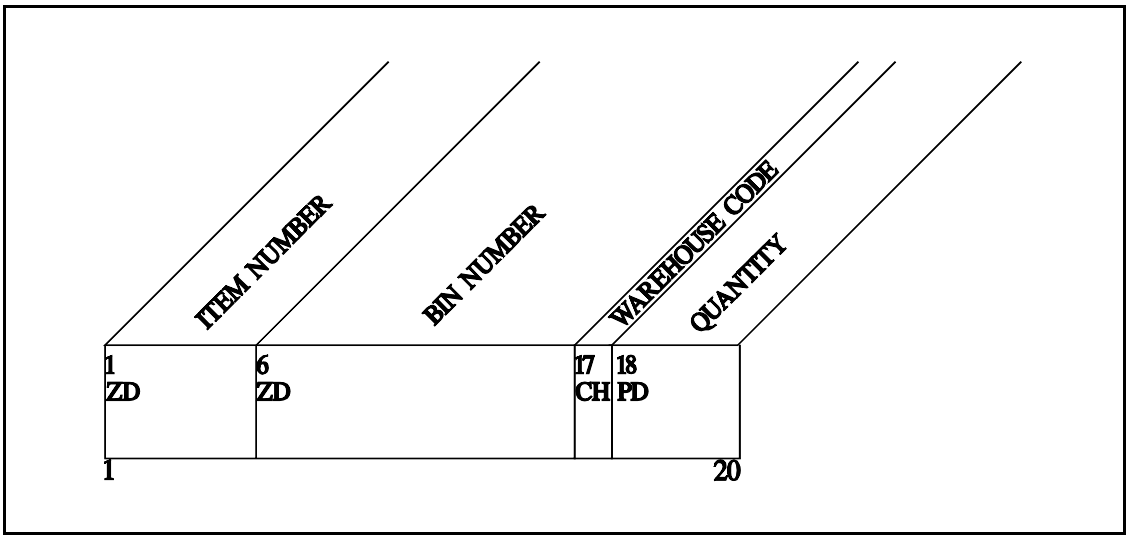


Figure 228. Post-INREC Record Layout

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 229.)

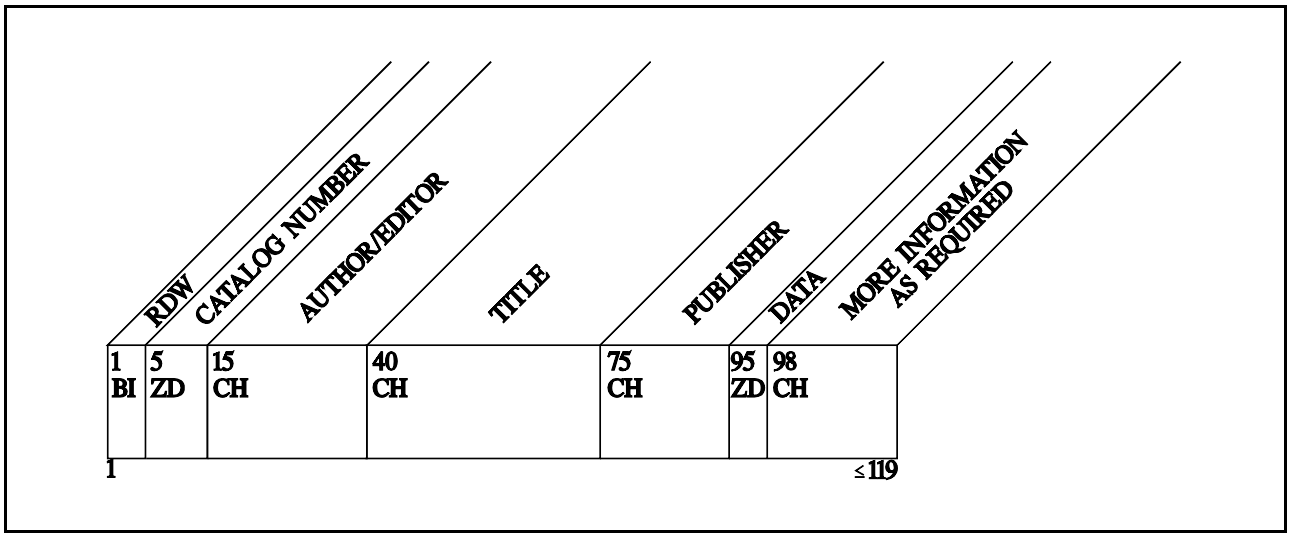


Figure 229. Sample Record Layout

To include only the relevant fields on the input records and to generate this list, the following is coded.

```
// JOB      LISTCAT                 Signals the Start of Job
// ASSGN    SYS001,SYSLST           Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'           Assigns Logical Unit to Tape
// TLBL     SORTIN1,'CATALOG'       Contains Input Tape File Information
// ASSGN    SYS003,X'251'           Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1                 Contains SORTWK File Information
// EXTENT   SYS003                  Defines Direct Access Area for SORTWK
// EXEC     SORT                    Signals Beginning of Sort Program
   RECORD   TYPE=V,LENGTH=119       Describes Input Records
   INREC    FIELDS=(1,14,           Selects Record Fields
                    98)
   SORT     FIELDS=(5,10,ZD,A)      Sorts Records
```

Figure 230. JCL and Required Control Statements

Figure 231 shows the input record after INREC processing.

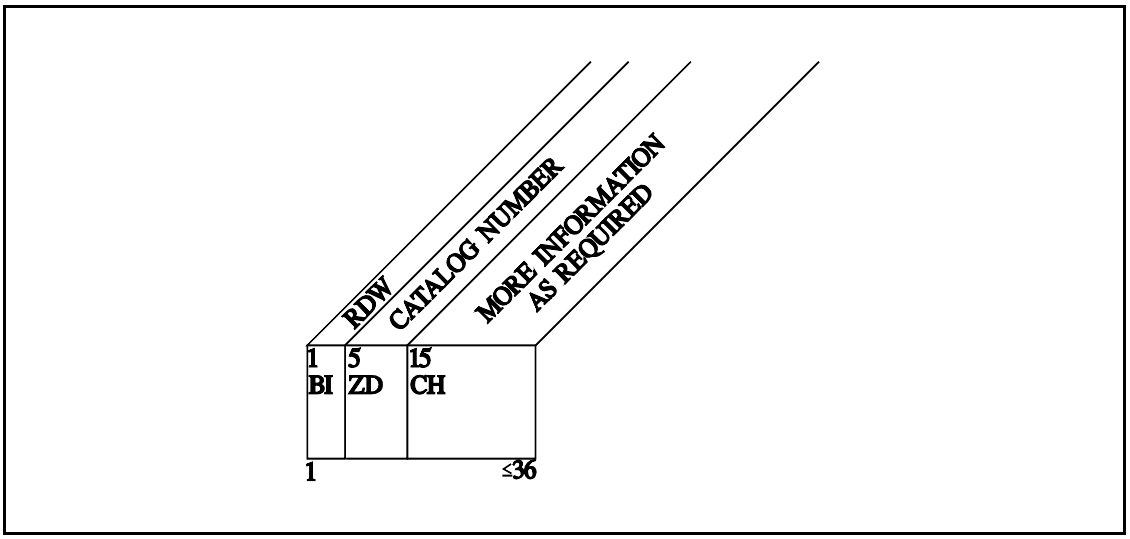


Figure 231. Form of Post-INREC Record

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. It is coded as follows.

$$\text{SUM}\left\{\begin{array}{l}\text{FIELDS} = (p_1,l_1[,f_1]\ldots,p_n,l_n[,f_n]) \\ \text{FIELDS} = \left\{\begin{array}{l}\text{NONE} \\ (\text{NONE})\end{array}\right\}\end{array}\right\}\ [,\text{XSUM}]\ [,\text{FORMAT} = f]$$

Figure 232. SUM Control Statement Format

p_n,l_n,f_n Specify the starting position (p), the length in bytes (l), and the data format (f) for each field for which you desire the sum to appear on the combined, cumulative record.

Note: Make sure that none of these fields has been specified on the SORT statement.

FORMAT Use this parameter to specify the default format for any field without "f" specified.

NONE Use this parameter, with or without parentheses, to eliminate all but one of two or more records that have identical information in the fields specified on the SORT statement.

XSUM Use this parameter to direct the sort to save all records eliminated by SUM processing in a separate output file.

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 233 gives the sample record layout.)

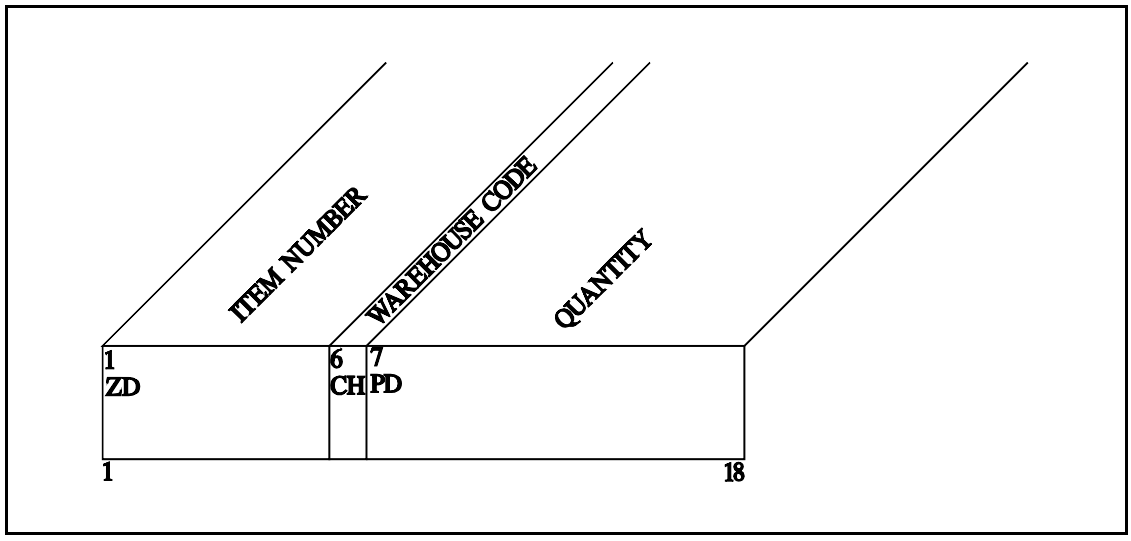


Figure 233. Input Record Layout

To combine those inventory records with identical item numbers and warehouse codes and to produce the required list, the following is coded.

```
// JOB      INVENTR                 Signals the Start of Job
// ASSGN    SYS001,SYSLST           Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'           Assigns Logical Unit to Tape
// TLBL     SORTIN1,'WAREHSE'       Contains Input Tape File Information
// ASSGN    SYS003,X'251'           Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1                 Contains SORTWK File Information
// EXTENT   SYS003                  Defines Direct Access Area for SORTWK
// EXEC     SORT                    Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=18        Describes Input Records
   SORT     FIELDS=(6,1,CH,A,1,5,ZD,A)  Sorts Records
   SUM      FIELDS=(7,12,PD)        Combines Records and Sums Numeric Data
```

Figure 234. 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 235 gives the sample record layout.)

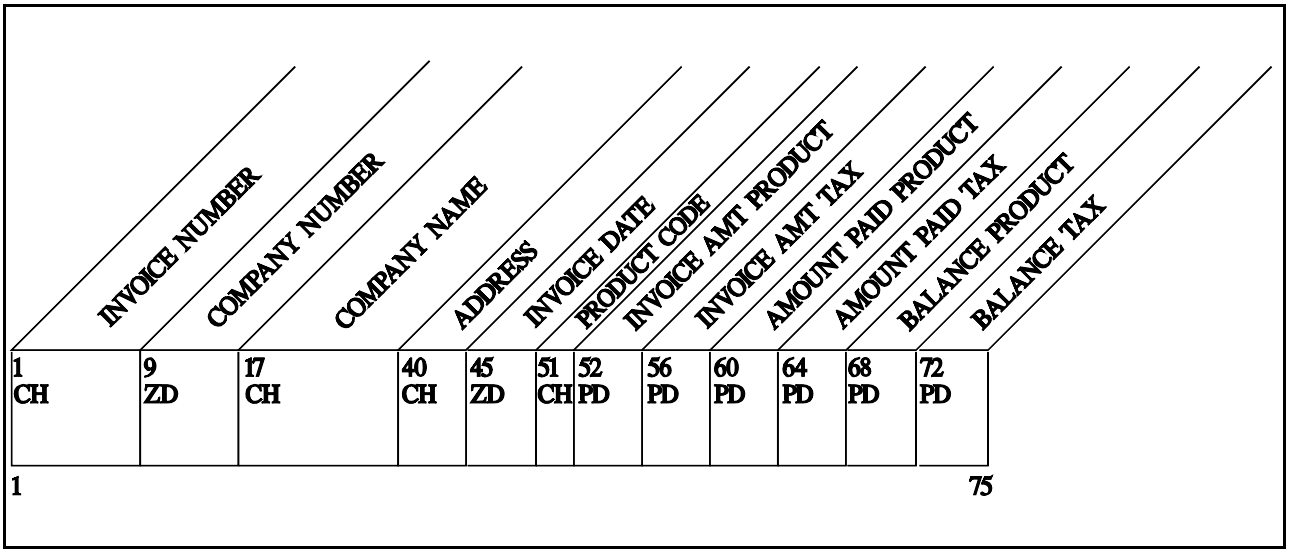


Figure 235. Input Record Layout

To combine multiple invoice records and generate the mailing list, the following is coded.

```
// JOB       MAILIST               Signals the Start of Job
// ASSGN     SYS001,SYSLST         Assigns Logical Unit to Printer
// ASSGN     SYS002,X'181'         Assigns Logical Unit to Tape
// TLBL      SORTIN1,'MASTINV'     Contains Input Tape File Information
// ASSGN     SYS003,X'251'         Assigns Logical Unit to SORTWK Device
// DLBL      SORTWK1               Contains SORTWK File Information
// EXTENT    SYS003                Defines Direct Access Area for SORTWK
// EXEC      SORT                  Signals Beginning of Sort Program
   RECORD    TYPE=F,LENGTH=75      Describes Input Records
   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 236. 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.

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.)

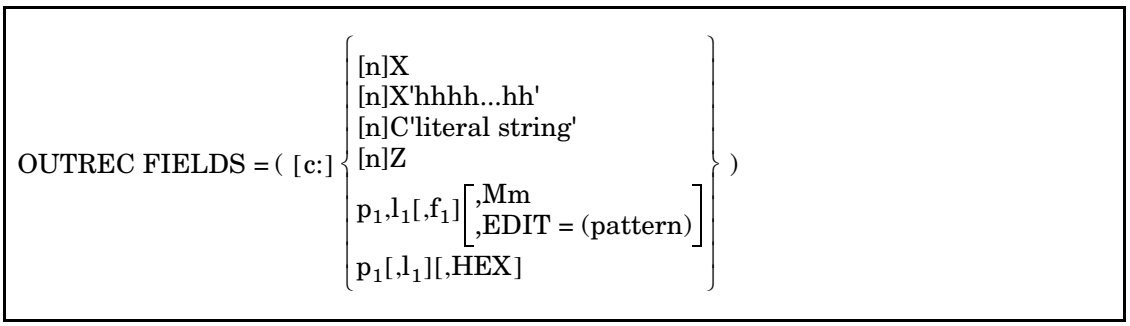


Figure 237. Basic OUTREC Control Statement Format

Note: See "OUTREC Control Statement" on page 2.174 for the complete format of the OUTREC statement.

c: Specify the column, or byte, in which you want a field to begin *after* OUTREC processing. If this is not coded, the first field specified by position, length, and, when required, format (see p,l,f, below) will begin in column one or the first byte of the post-OUTREC record.

[n]X Specify the number (n) of blanks (X).

[n]X'hhhh...hh' Specify hexadecimal digits (X'hhhh...hh').

[n]C'literal string' Specify the literal strings (C'literal string').

[n]Z Specify bytes of binary zeros (Z) you want to insert.

p,l Specify the starting position (p) and the length in bytes (l) of each input field. Be sure to allow for any processing, such as INREC, that may have altered the original input record.

f Specify the format of any numeric data that you want converted to printable (ZD) data.

Mm|EDIT=(pattern) To format the converted numeric data, specify one of ten SyncSort editing masks (Mm) or a pattern of your own, using I for insignificant digits and T for significant digits. (Note that you may replace the I and the T in EDIT with any characters you want and then use these to mark insignificant and

significant digits.) If neither Mm nor EDIT is specified, a default mask is used. (See “Mm Subparameter (Editing Masks)” on page 2.202 and “EDIT Subparameter” on page 2.200.)

p[,l],HEX Specify the starting position (p) and length in bytes (l) of the field you want converted to printable hexadecimal (HEX) format.

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 238 gives the layout for the masterfile record.)

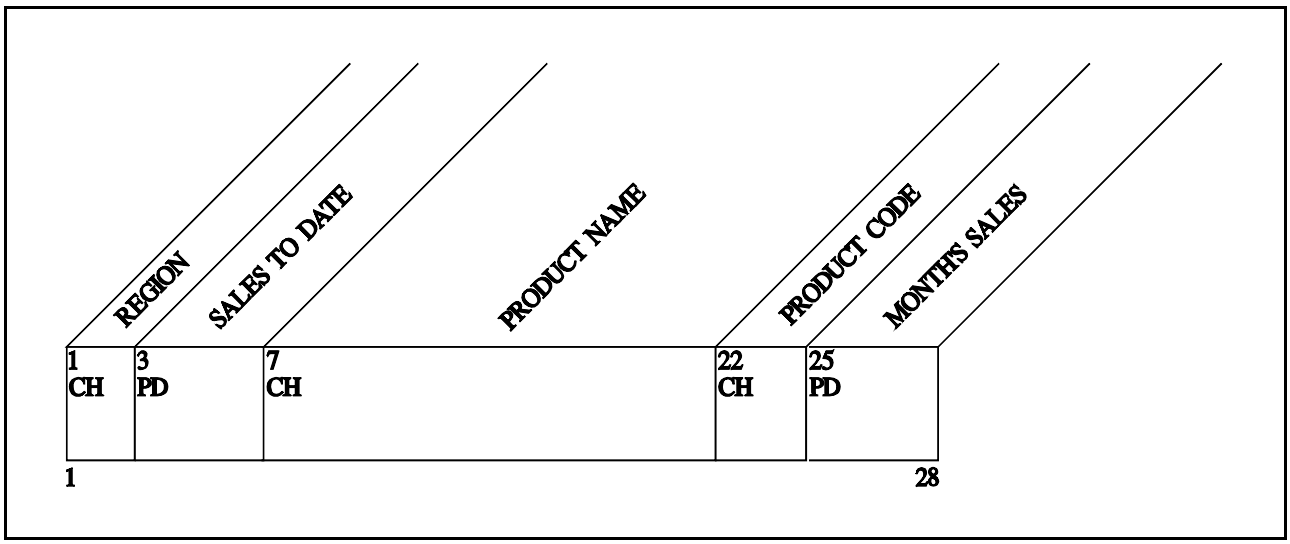


Figure 238. Input Record Layout

To sort the records alphabetically by product name and reposition the data fields, the following is coded.

```
// JOB       SORTPRD               Signals the Start of Job
// ASSGN     SYS001,SYSLST         Assigns Logical Unit to Printer
// ASSGN     SYS002,X'181'         Assigns Logical Unit to Tape
// TLBL      SORTIN1,'MASTPRD'     Contains Input Tape File Information
// ASSGN     SYS003,X'251'         Assigns Logical Unit to SORTWK Device
// DLBL      SORTWK1               Contains SORTWK File Information
// EXTENT    SYS003                Defines Direct Access Area for SORTWK
// EXEC      SORT                  Signals Beginning of Sort Program
   RECORD    TYPE=F,LENGTH=28      Describes Input Records
   SORT      FIELDS=(7,15,CH,A)    Sorts Records
   OUTREC    FIELDS=(22,3,         Repositions Fields on Output Records
                     7,15,
                     1,2,
                    25,4,
                     3,4)
```

Figure 239. JCL and Required Control Statements

Figure 240 shows the output record after OUTREC processing.

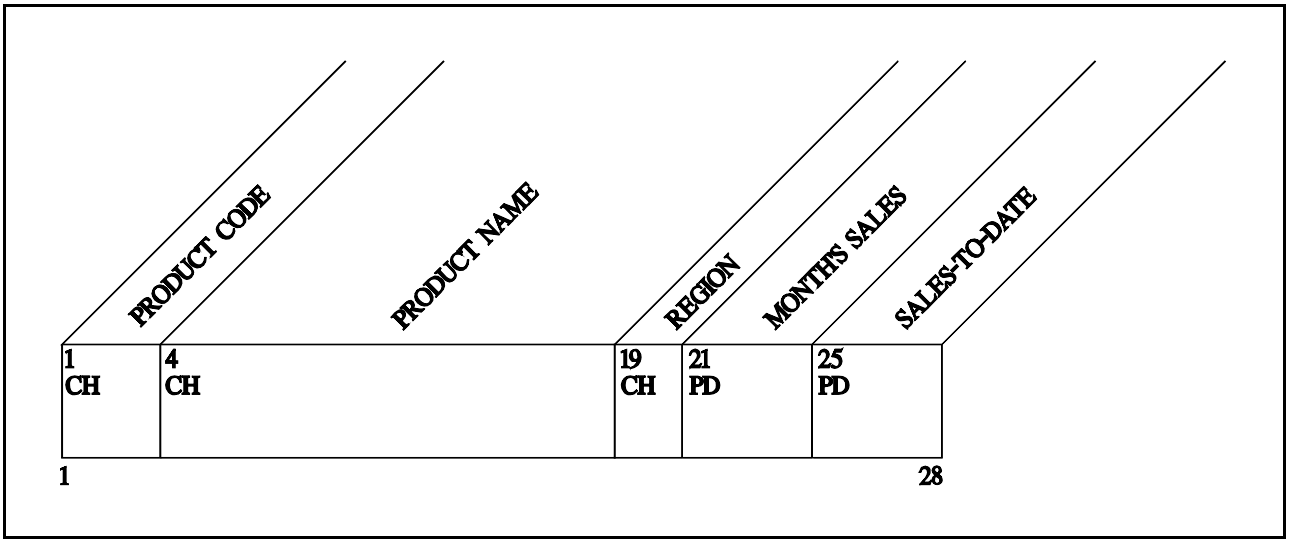


Figure 240. Post-OUTREC Record Layout

Explanation: After the records are sorted alphabetically by product 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 241. Branch A, however, has formatted its masterfile records as outlined in Figure 242.

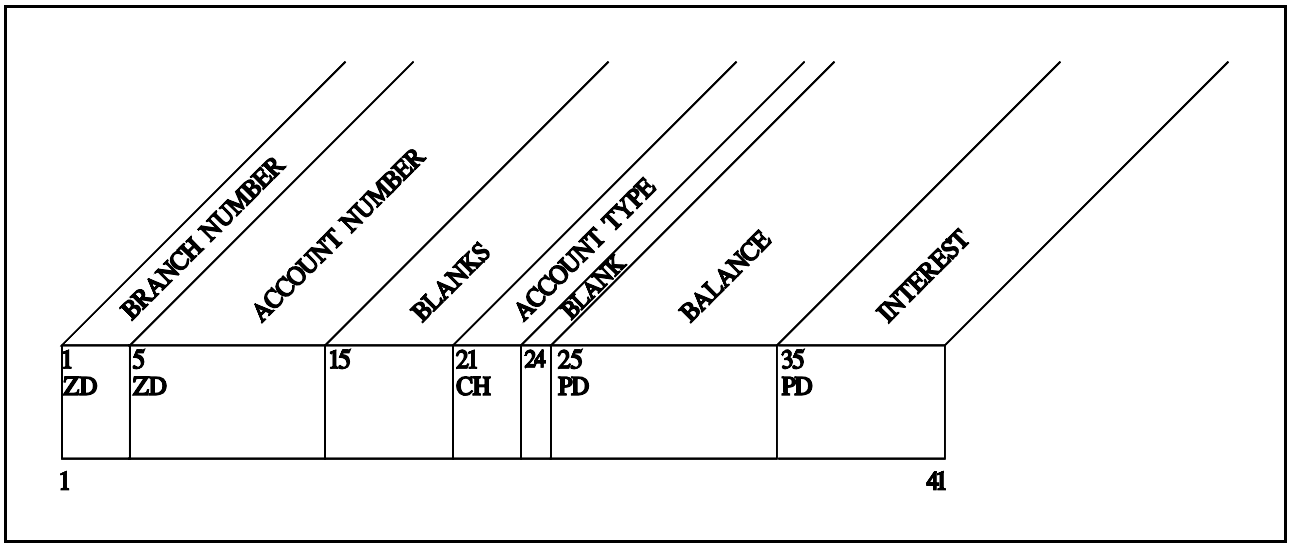


Figure 241. Required Format

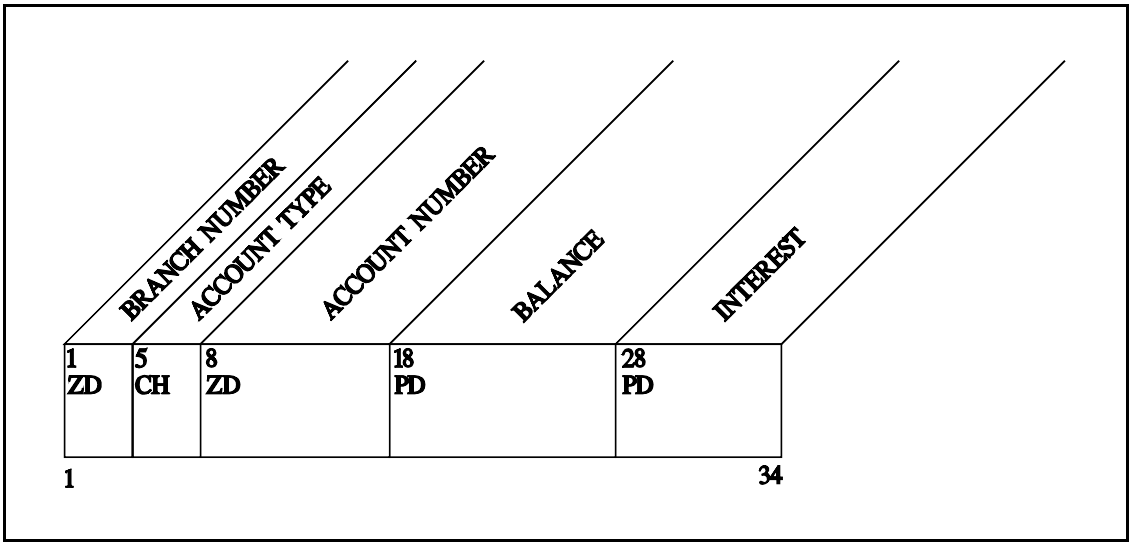


Figure 242. Input Record Layout

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.

```
// JOB      FORMAT                Signals the Start of Job
// ASSGN    SYS001,SYSLST         Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'         Assigns Logical Unit to Tape
// TLBL     SORTIN1,'ACCTMST'     Contains Input Tape File Information
// ASSGN    SYS003,X'251'         Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1               Contains SORTWK File Information
// EXTENT   SYS003                Defines Direct Access Area for SORTWK
// EXEC     SORT                  Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=41      Describes Input Records
   SORT     FIELDS=COPY           Copies Records
   OUTREC   FIELDS=(1,4,          Repositions Fields on Output Records
                    8,10,
                   6X,
                    5,3,
                   1X,
                   18,17)
```

Figure 243. JCL and Required Control Statements

Figure 244 shows the effect of OUTREC processing on the output record.

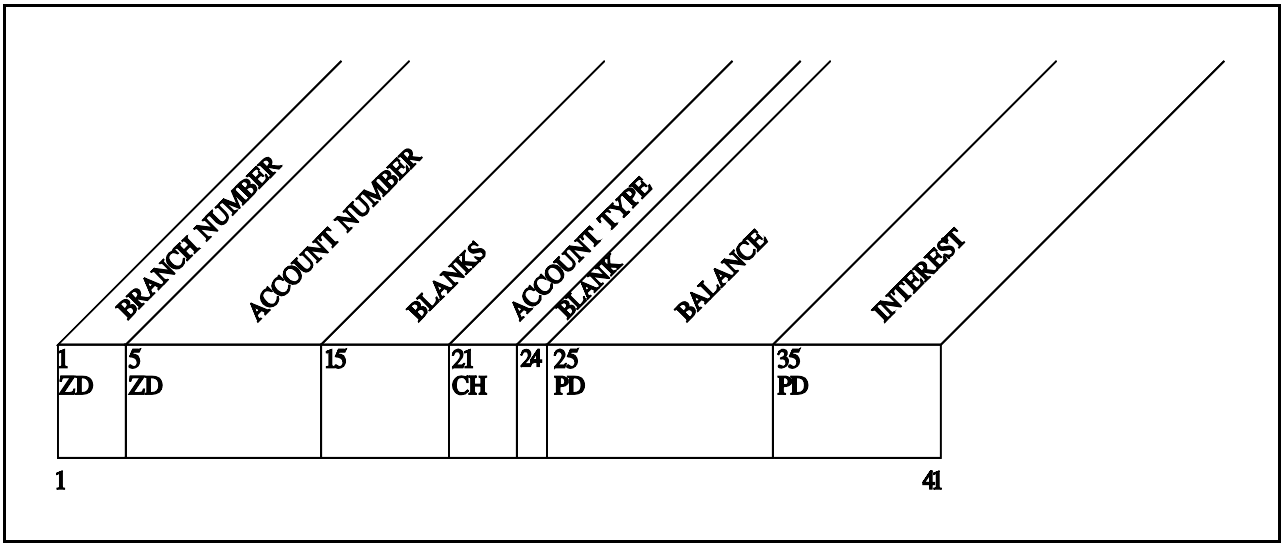


Figure 244. Post-OUTREC Record Layout

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 244, 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 245 gives the input record layout.

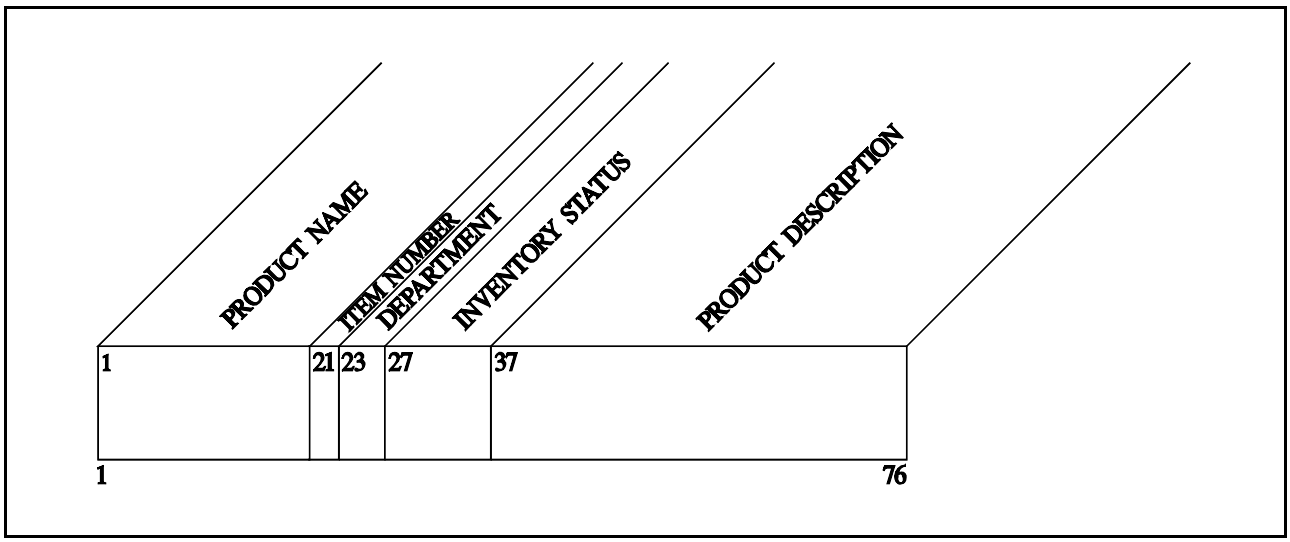


Figure 245. Input Record Layout

To copy the records and insert the 4 bytes of binary zeros, the following is coded.

```
// JOB      COLUMNS              Signals the Start of Job
// ASSGN    SYS001,SYSLST        Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'        Assigns Logical Unit to Tape
// TLBL     SORTIN1,'INVREC'     Contains Input Tape File Information
// ASSGN    SYS003,X'251'        Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1              Contains SORTWK File Information
// EXTENT   SYS003               Defines Direct Access Area for SORTWK
// EXEC     SORT                 Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=76     Decsribes Input Records
   SORT     FIELDS=COPY          Copies Records
   OUTREC   FIELDS=(1,20,        Inserts Binary Zeros and
                   4Z,           Reformats Records
                25:21,56)
```

Figure 246. JCL and Required Control Statements

The effect of OUTREC processing is shown in Figure 247.

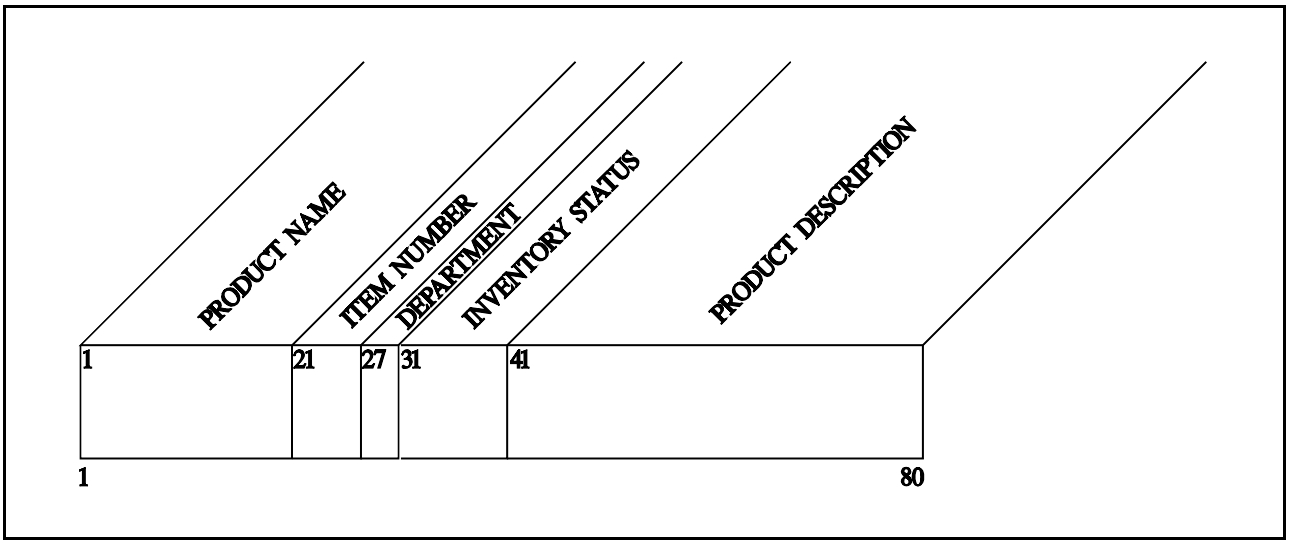


Figure 247. Post-OUTREC Record Layout

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 248 gives the input record layout.)

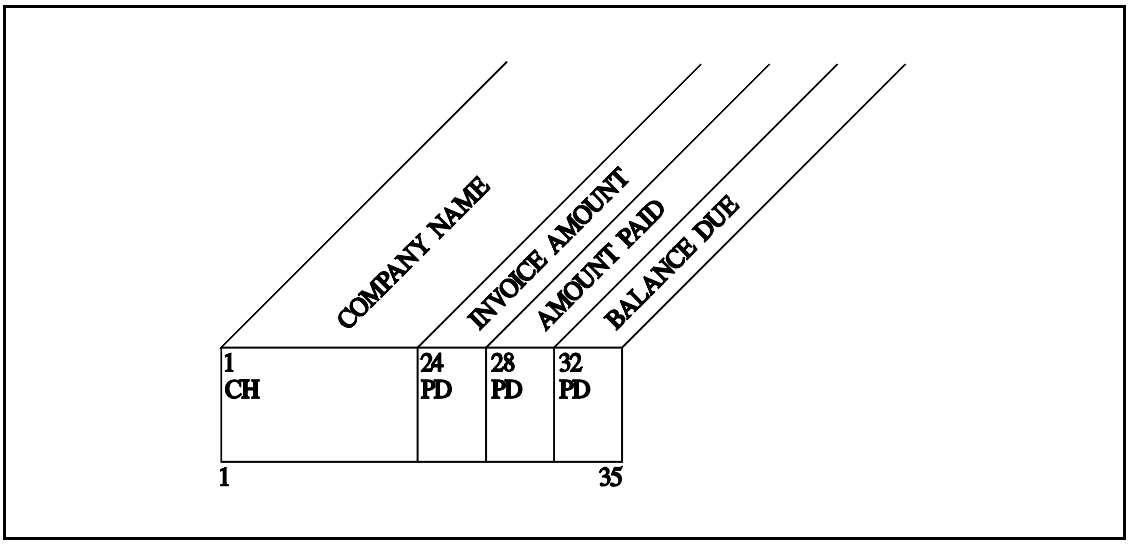


Figure 248. Input Record Layout

To sort the records, convert the three fields of packed-decimal data, and insert the commas and decimal points, the following is coded.

```
// JOB      INVOICE                Signals the Start of Job
// ASSGN    SYS001,SYSLST          Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'          Assigns Logical Unit to Tape
// TLBL     SORTIN1,'NEWINV'       Contains Input Tape File Information
// ASSGN    SYS003,X'251'          Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1                Contains SORTWK File Information
// EXTENT   SYS003                 Defines Direct Access Area for SORTWK
// EXEC     SORT                   Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=35       Describes Input Records
   SORT     FIELDS=(1,23,CH,A)     Sorts Records
   OUTREC   FIELDS=(17:1,23,       Repositions Record Fields and
                    52:24,4,PD,M2, Converts Data
                    74:28,4,PD,M2,
                    96:32,4,PD,M2)
```

Figure 249. JCL and Required Control Statements

The effect of OUTREC processing on the input record is shown in Figure 250.

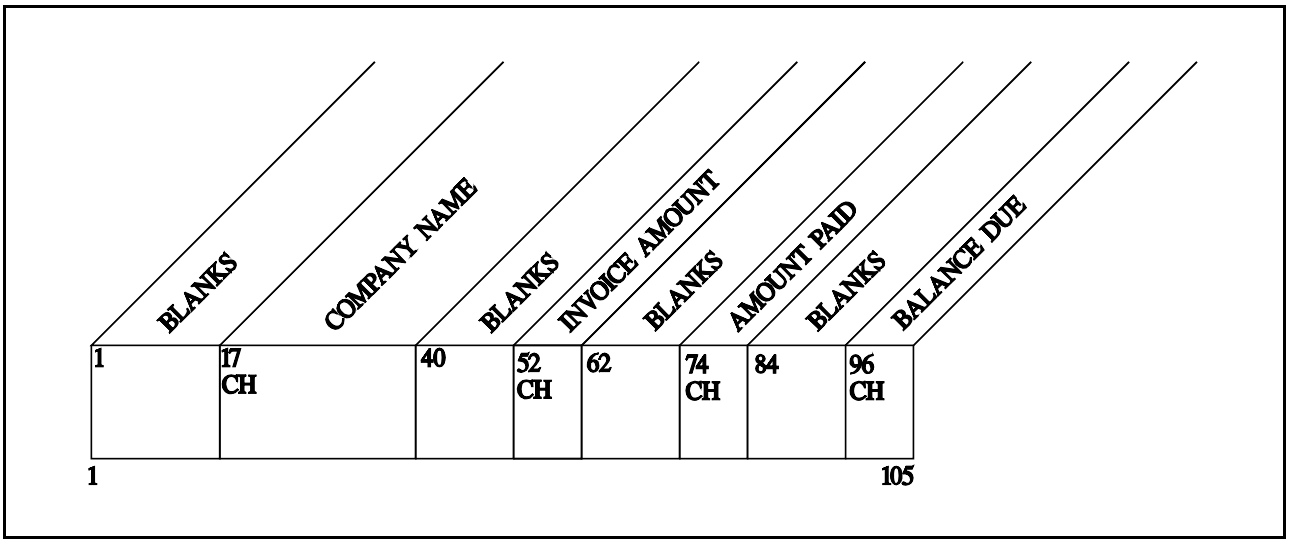


Figure 250. Post-OUTREC Record Layout

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 edited 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 251 shows the input record layout.

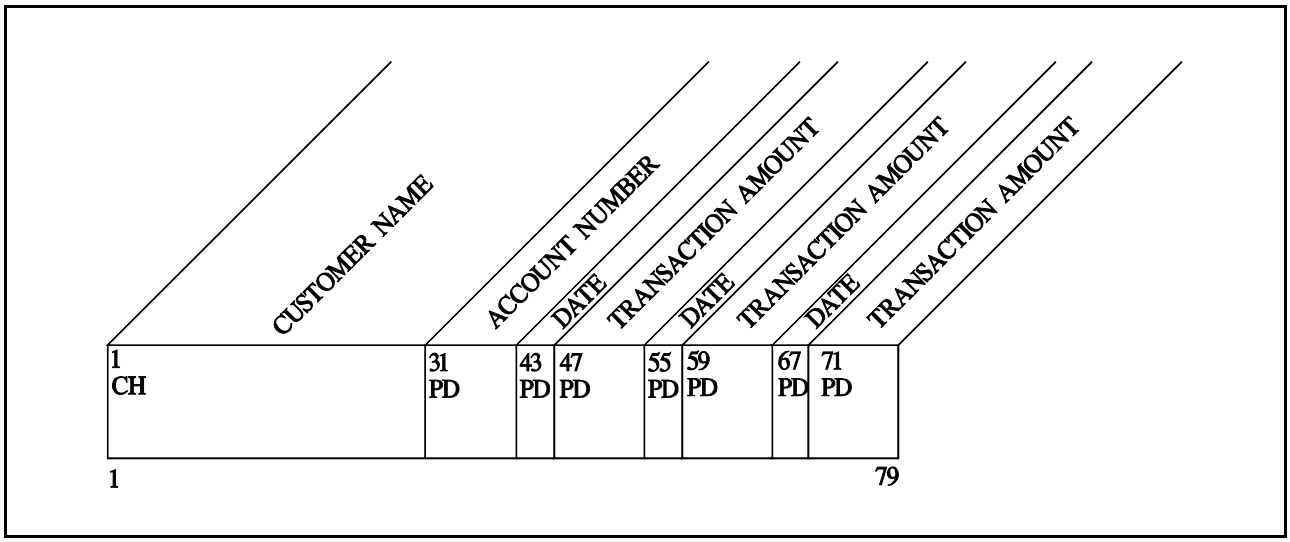


Figure 251. Sample Input Record Layout

In order to find the invalid data, the following is coded.

```
// JOB      ACCTNO                Signals the Start of Job
// ASSGN    SYS001,SYSLST         Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'         Assigns Logical Unit to Tape
// TLBL     SORTIN1,'CUSTNO'      Contains Input Tape File Information
// ASSGN    SYS003,X'251'         Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1               Contains SORTWK File Information
// EXTENT   SYS003                Defines Direct Access Area for SORTWK
// EXEC     SORT                  Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=79      Describes Input Records
   SORT     FIELDS=COPY           Copies Records
   OUTREC   FIELDS=(1,30,         Reformats Output Records and
                36:31,12,HEX)     Converts Data
```

Figure 252. JCL and Required Control Statements

The effect of OUTREC processing on the input record is shown in Figure 253.

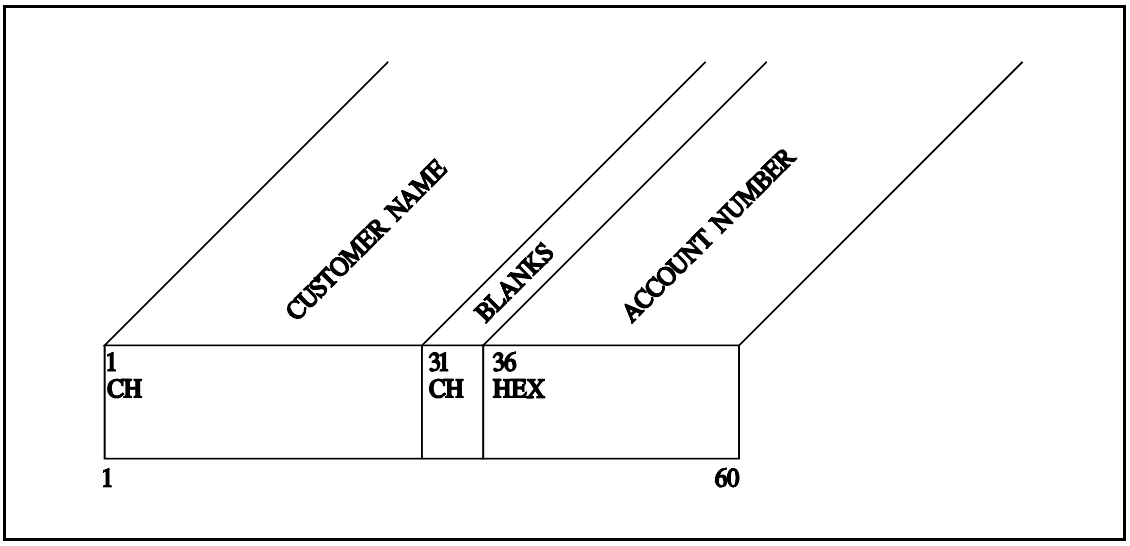


Figure 253. Post-OUTREC Record Layout

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 254 gives the input record layout.)

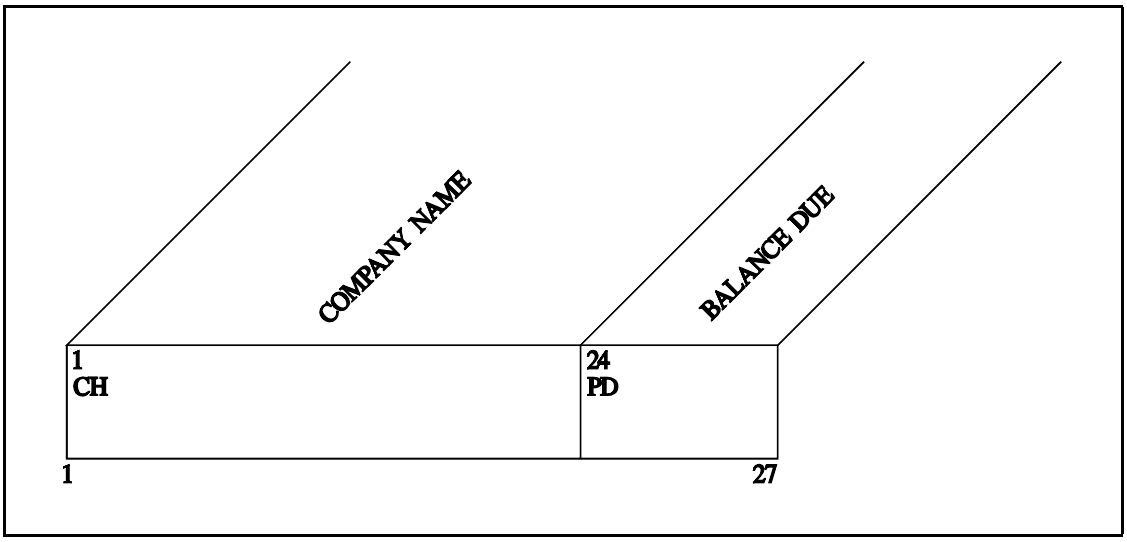


Figure 254. Input Record Layout

To sort the records and accomplish the conversion and editing, the following is coded.

```
// JOB      PAYMENT              Signals the Start of Job
// ASSGN    SYS001,SYSLST        Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'        Assigns Logical Unit to Tape
// TLBL     SORTIN1,'INVOICE'    Contains Input Tape File Information
// ASSGN    SYS003,X'251'        Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1              Contains SORTWK File Information
// EXTENT   SYS003               Defines Direct Access Area for SORTWK
// EXEC     SORT                 Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=27     Describes Input Records
   SORT     FIELDS=(1,23,CH,A)   Sorts Records
   OUTREC   FIELDS=(1,23,        Converts and Edits Data, Inserts Blanks
                    10X,
                     24,4,PD,EDIT=($II,IIT.TT))
```

Figure 255. JCL and Required Control Statements

Figure 256 shows the effect of OUTREC processing on the input record.

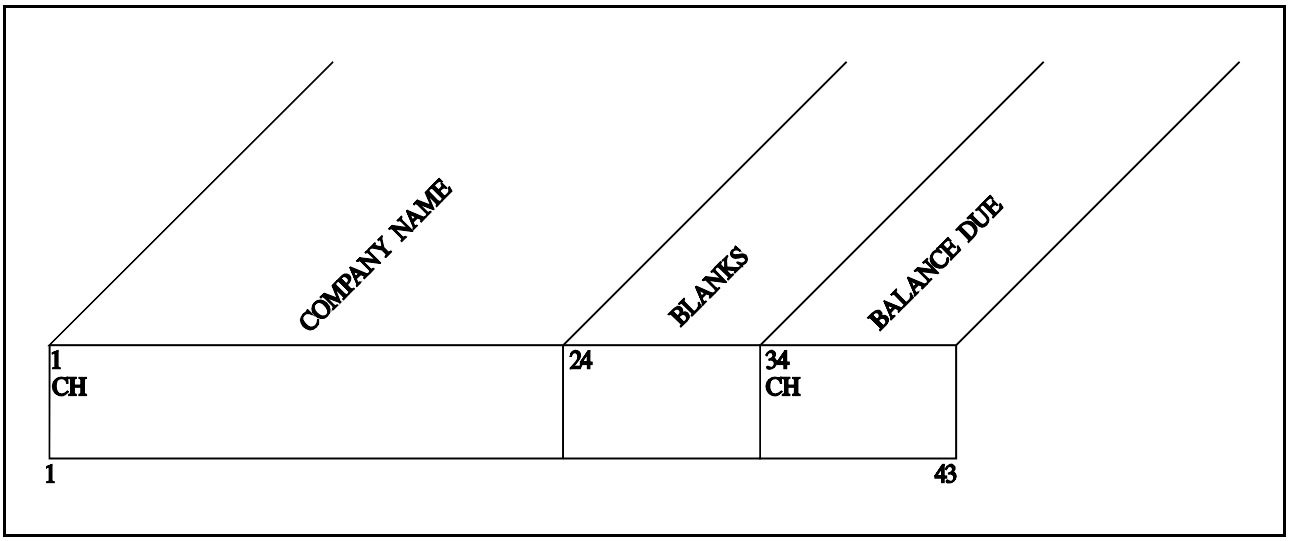


Figure 256. Post-OUTREC Record Layout

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 *I*s indicate that leading zeros should *not* be printed and the *T*s 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 257 gives the input record layout.)

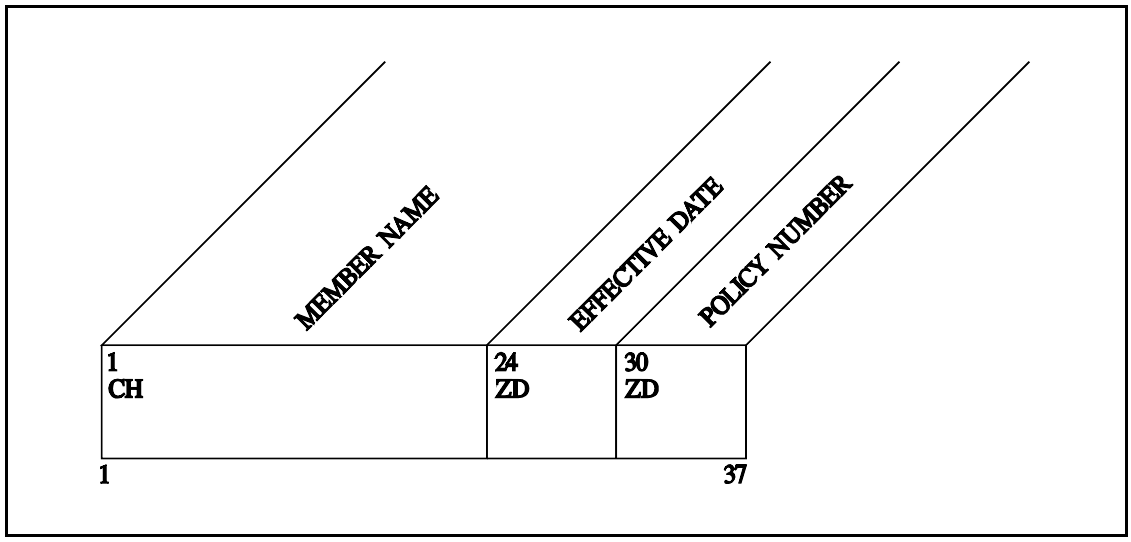


Figure 257. Input Record Layout

To sort the records and format the date field with the required slashes, the following is coded.

```
// JOB      SORTDT                    Signals the Start of Job
// ASSGN    SYS001,SYSLST             Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'             Assigns Logical Unit to Tape
// TLBL     SORTIN1,'NEWPLCY'         Contains Input Tape File Information
// ASSGN    SYS003,X'251'             Assigns Logical Unit to SORTWK
                                      Device
// DLBL     SORTWK1                   Contains SORTWK File Information
// EXTENT   SYS003                    Defines Direct Access Area for
                                      SORTWK
// EXEC     SORT                      Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=37          Describes Input Records
   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 258. JCL and Required Control Statements

The effect of OUTREC processing is shown in Figure 259.

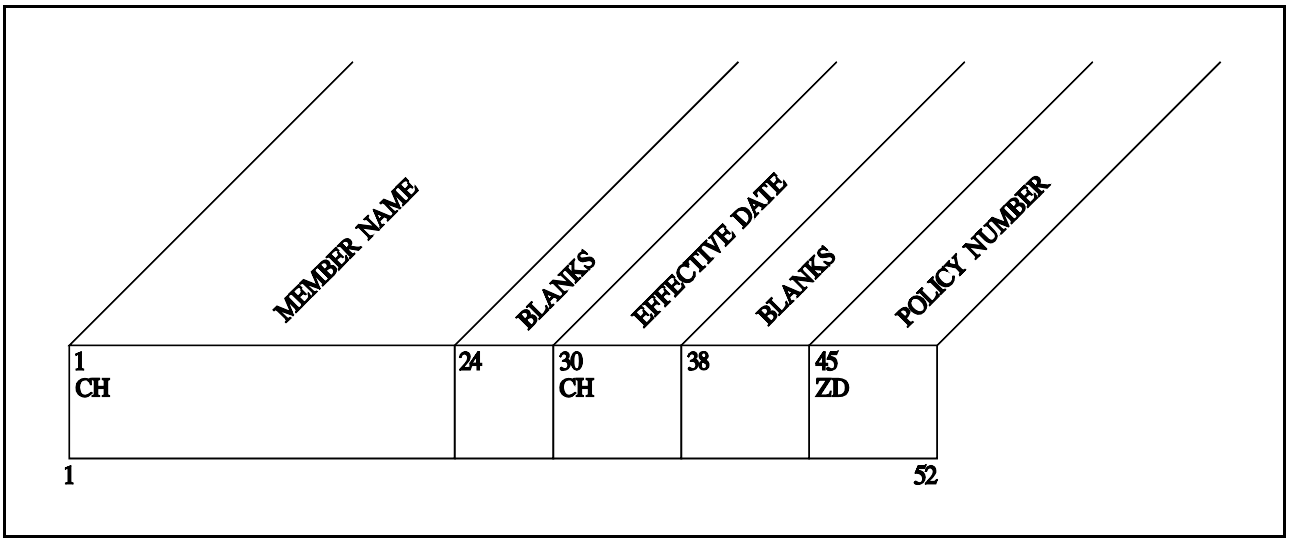


Figure 259. Post-OUTREC Record Layout

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.

Printing Input Records on Multiple Output Lines

Example: In this example, five input record fields, shown in Figure 260, are copied to an output file with each field printed as a separate output line.

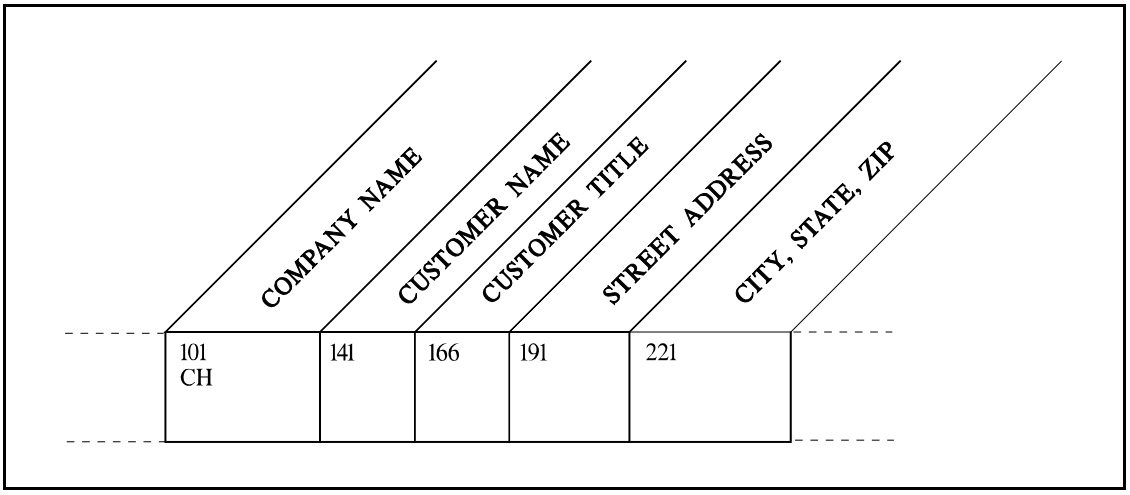


Figure 260. Input Record Layout

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 261, the newline character follows the specification of each input field's starting position and length.

```
// JOB      SORTDT                    Signals the Start of Job
// ASSGN    SYS001,SYSLST             Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'             Assigns Logical Unit to Tape
// TLBL     SORTIN1,'NEWPLCY'         Contains Input Tape File Information
// ASSGN    SYS003,X'251'             Assigns Logical Unit to SORTWK
                                      Device
// DLBL     SORTWK1                   Contains SORTWK File Information
// EXTENT   SYS003                    Defines Direct Access Area for
                                      SORTWK
// EXEC     SORT                      Signals Beginning of Sort Program
     SORT FIELDS=(101,40,CH,A)              Sorts Records
     RECORD TYPE=F,LENGTH=300               Record Length
     OUTFIL HEADER2=('CUSTOMER ADDRESS LIST',3/), Prints a Page
                                                  Header
            OUTREC=(101,40,/,               Prints the Data in the
                                            Field and Starts a
                                            New Output Line
                    141,25,/,               As Above
                    166,25,/,               As Above
                    191,30,/,               As Above
                    266,35,2/)              As Above but Starts
                                            2 New Output Lines
```

Figure 261. 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 262 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 multiline 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 262. 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 as described below.

$$\text{SECTIONS} = (p,l,\text{SKIP} = \left\{\begin{matrix} n\text{L} \\ \text{P} \end{matrix}\right\})$$

Figure 263. Basic SECTIONS Parameter Format

Note: See "SECTIONS Parameter (Optional)" on page 2.162 for the complete syntax of the SECTIONS parameter.

p,l Specify the starting position and the length in bytes of the field that you have chosen to determine section breaks. (Note: Typically, this field is one on which the data has been sorted.) A section break will occur every time the data in this sort field changes.

When specifying the fields that control section breaks, keep the following in mind:

- Give the position and length of the field as it appears *after* any processing by an E15 or E32 exit, the INREC control statement, the OUTREC control statement, and an E35 exit, but *before* processing due to the OUTREC and/or other OUTFIL parameters.
- You may specify more than one field in order to subdivide the report within sections. Specify the major control field first, followed (in major to minor order) by the control fields that will generate breaks within a major section.

SKIP=nL Specifies the number of lines (nL) that should be skipped between each section. Spacing occurs after the last section trailer (TRAILER3) in the section.

SKIP=P Specifies a page break (P). Spacing occurs after the last section trailer (TRAILER3) in the section.

Dividing Output into Sections

Example: A personnel roster is to be divided into sections by Department. (Figure 264 presents the layout for the input record.)

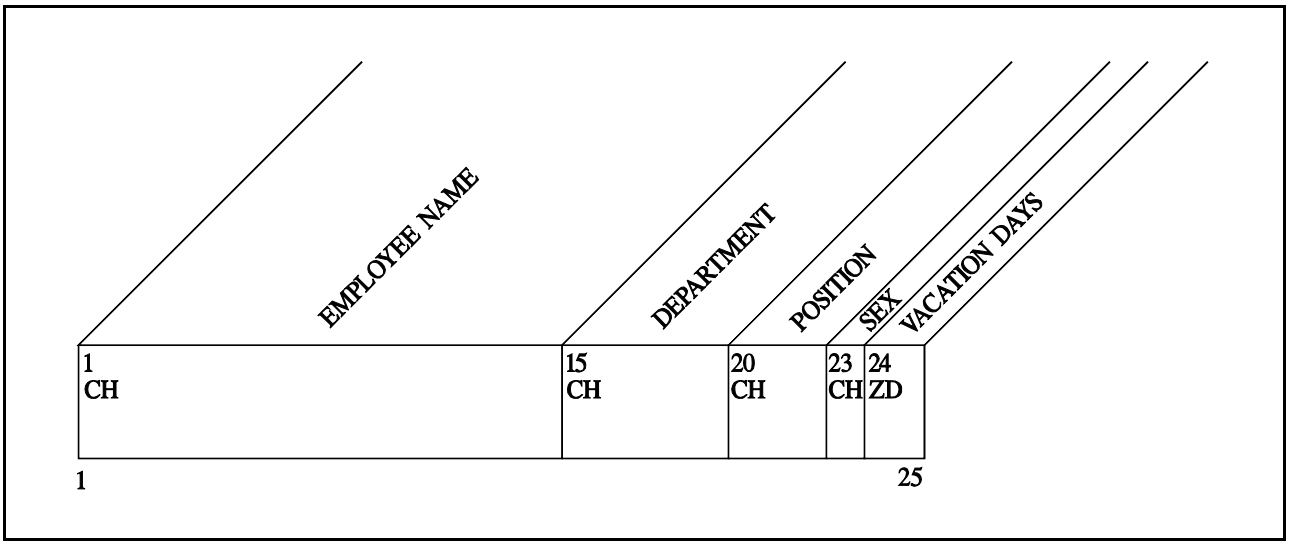


Figure 264. Input Record Layout

To sort the records and generate a list that is divided by Department, the following is coded.

```
// JOB      ROSTER                  Signals the Start of Job
// ASSGN    SYS001,SYSLST           Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'           Assigns Logical Unit to Tape
// TLBL     SORTIN1,'PRSNL'         Contains Input Tape File Information
// ASSGN    SYS003,X'251'           Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1                 Contains SORTWK File Information
// EXTENT   SYS003                  Defines Direct Access Area for SORTWK
// EXEC     SORT                    Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=25        Describes Input Records
   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 265. JCL and Required Control Statements

A sample of the listing generated is shown in Figure 266.

```
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 266. 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" on page 4.46) or to indicate the end of a section with, for example, a string of asterisks or to provide a list of abbreviations used in the report.

To generate Headers and/or Trailers, code the OUTFIL statement as described below:

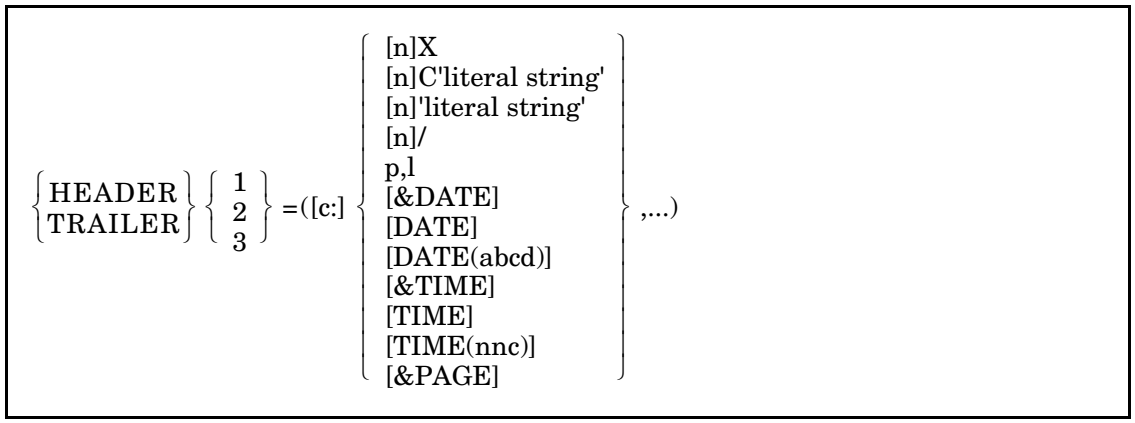


Figure 267. Basic HEADER and TRAILER Parameter Formats

Note: See "HEADER1/HEADER2 Parameters (Optional)" on page 2.149 and "TRAILER1/ TRAILER2 Parameters (Optional)" on page 2.164 for complete formats of HEADER and TRAILER parameters.

HEADERn — Use a HEADER1 for a title page, a HEADER2 for a page header, a HEADER3 for a section header. Note: HEADER3 must be specified as a subparameter of the SECTIONS parameter.

TRAILERn — Specify a TRAILER1 for a report trailer, a TRAILER2 for a page trailer, a TRAILER3 for a section trailer. Note: TRAILER3 must be specified as a subparameter of the SECTIONS parameter.

c: — Specify the column in which you want the entry to begin printing.

n — Specify the number of times you want the blank (X), character data ('literal string'), or forward spacing (/) repeated.

p,l — Specify the starting position (p) and the length in bytes (l) of the data-record field(s) that you want to include in either a header or trailer. For a HEADER1, the field(s) will be extracted from the first data record in the report; for a HEADER2, from the first data record on the page; for a HEADER3, from the first data record in the section. For a TRAILER1, the field(s) will be extracted from the last data record in the report; for a TRAILER2, from the last data record on a page; for a TRAILER3, from the last data record in the section.

&DATE=(abcd)

The &DATE or DATE subparameter specifies the current generated system date and requires 8 bytes. abcd indicates that you arrange the year, month, and day in any order. c is the separator character. Use M to denote month, D to denote day, and Y to denote the last two characters of the year. You can specify 4 as an alternative to Y if you want the year expressed in four digits (2006 as opposed to 06).

For example, if the current date is 4/24/06 and &DATE=(DMY.), the output date is 24.04.06. With the same example, if &DATE=(4MD-), the output date is 2006-04-24.

&TIME=(nnc)

The &TIME subparameter specifies the current step start time, which is the time that the current execution of SyncSort began. The default time format is hh:mm:ss.

nn can be either 12 or 24. 24 specifies twenty-four hour clock with the format hhcmmcss and requires 8 bytes. 12 specifies twelve hour clock with the format hhcmmcss xx and requires 11 bytes. xx can be either am or pm. c denotes the separation character. For example, &TIME=(24-) expresses 7:31 pm as 19-31-00.

&PAGE

Specify this parameter if you want the logical pages to be numbered sequentially. Page numbers appear in a six-byte format that is right justified with leading zeros suppressed.

Note: If any specified HEADER or TRAILER exceeds the length of the post-OUTREC record, be sure to code the LRECL parameter on the OUTFIL statement to match the length of the *longest* header or trailer. If this length is unknown, you may specify LRECL=132.

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.

```
// JOB      DSRPT                    Signals the Start of Job
// ASSGN    SYS001,SYSLST            Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'            Assigns Logical Unit to Tape
// TLBL     SORTIN1,'SALES'          Contains Input Tape File Information
// ASSGN    SYS003,X'251'            Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1                  Contains SORTWK File Information
// EXTENT   SYS003                   Defines Direct Access Area for SORTWK
// EXEC     SORT                     Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=80         Describes Input Records
   SORT     FIELDS=(1,15,CH,A)       Sorts Records
      .
      .
      .
   OUTFIL   HEADER1=(15/,49:'D E P A R T M E N T A L  S A L E S',
                    4/,59:'F E B R U A R Y',
                    4/,63:'2 0 0 0'),  Generates Title Page
      .
      .
      .
```

Figure 268. JCL and Required Control Statements

Figure 269 shows the header that is generated by the above HEADER1 parameter:

```
            D E P A R T M E N T A L  S A L E S

                    F E B R U A R Y

                        2 0 0 0
```

Figure 269. Sample HEADER1

Explanation: The HEADER1 parameter produces a header that will print on a separate page, with no page number, at the beginning of the report. The first number-slash (n/) entry, 15/, causes the printer to skip 15 lines before printing. The following number-colon entry (c:), 49:, specifies the column in which the literal string
'D E P A R T M E N T A L S A L E S' begins to print. Note that the literal string prints *exactly* as it is entered between the single quotes, with a space between each letter and a double space between the words.

The next entry, 4/, causes the printer to skip 3 more blank lines before starting to print the literal string 'F E B R U A R Y' in column 59.

Finally, three more lines are left blank (4/) and the literal string '2 0 0 0' begins printing in column 63.

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.

```
// JOB      DSRPT                 Signals the Start of Job
// ASSGN    SYS001,SYSLST         Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'         Assigns Logical Unit to Tape
// TLBL     SORTIN1,'SALES'       Contains Input Tape File Information
// ASSGN    SYS003,X'251'         Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1               Contains SORTWK File Information
// EXTENT   SYS003                Defines Direct Access Area for SORTWK
// EXEC     SORT                  Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=80      Describes Input Records
   SORT     FIELDS=(1,15,CH,A)    Sorts Records
      .
      .
      .
   OUTFIL .
          .
            HEADER2=(1:'PGM NUMBER 5',  Generates Page Heading
                    46:'DEPARTMENTAL SALES REPORT FOR FEBRUARY 2000',
                   101:'DATE:',
                   107:&DATE,
                   121:'PAGE:',
                   127:&PAGE,//,
                     1:'DEPARTMENT',
                    40:'SALES MANAGER',
                    61:'SALES REP',
                    78:'SALES THIS PERIOD',
                   103:'SALES YEAR TO DATE',//),
          .
          .
```

Figure 270. JCL and Required Control Statements

Figure 271 shows a representation of the header that is generated by the above HEADER2 parameter.

```
PGM NUMBER 5  DEPARTMENTAL SALES REPORT FOR FEBRUARY 2000 DATE: 02/01/00 PAGE: 1

DEPARTMENT  SALES MANAGER  SALES REP   SALES THIS PERIOD    SALES YEAR TO DATE
```

Figure 271. 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.

```
// JOB      DSRPT              Signals the Start of Job
// ASSGN    SYS001,SYSLST      Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'      Assigns Logical Unit to Tape
// TLBL     SORTIN,'SALES'     Contains Input Tape File Information
// ASSGN    SYS003,X'251'      Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1            Contains SORTWK File Information
// EXTENT   SYS003             Defines Direct Access Area for SORTWK
// EXEC     SORT               Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=80   Describes Input Records
   SORT     FIELDS=(1,15,CH,A) Sorts Records
   OUTFIL   LRECL=114,
            OUTREC=(1:1,15,    Repositions Fields on Output Records
                   23:23,7,    and Edits Data
                   51:48,3,
                   72:60,4,PD,EDIT=($II,IIT.TT),
                  101:64,4,PD,EDIT=($II,IIT.TT)),
            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 272. JCL and Required Control Statements

Figure 273 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 273. 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. LRECL=114 has been specified on the OUTFIL statement so that the length of the output record 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 header's 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.

```
// JOB      DSRPT                 Signals the Start of Job
// ASSGN    SYS001,SYSLST         Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'         Assigns Logical Unit to Tape
// TLBL     SORTIN1,'SALES'       Contains Input Tape File Information
// ASSGN    SYS003,X'251'         Assigns Logical Unit to SORTWK Device
// TLBL     SORTWK1               Contains SORTWK File Information
// EXTENT   SYS003                Defines Direct Access Area for SORTWK
// EXEC     SORT                  Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=80      Describes Input Records
   SORT     FIELDS=(1,15,CH,A)    Sorts Records
   OUTFIL   LRECL=114,
            OUTREC=(51:48,3,      Repositions Fields on Output Records
                    72:60,4,PD,EDIT=($II,IIT.TT),   and Edits Data
                   101:64,4,PD,EDIT=($II,IIT.TT)),
            SECTIONS=(1,15,SKIP=5L, Generates Section Breaks
            HEADER3=(1:1,15,        Generates Section Headings
                     23:23,7,
                     48:'SALES REP',
                     68:'SALES THIS PERIOD',
                     97:'SALES YEAR TO DATE'//))
```

Figure 274. JCL and Required Control Statements

Figure 275 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 275. 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 23:23,7 (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 23; the literal string "SALES REP", in column 48, and so on. Blanks are automatically inserted in the space between the columns. Note that LRECL=114 has been specified on the OUTFIL statement 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 2000 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.

```
// JOB      DSRPT                 Signals the Start of Job
// ASSGN    SYS001,SYSLST         Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'         Assigns Logical Unit to Tape
// TLBL     SORTIN1,'SALES'       Contains Input Tape File Information
// ASSGN    SYS003,X'251'         Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1               Contains SORTWK File Information
// EXTENT   SYS003                Defines Direct Access Area for SORTWK
// EXEC     SORT                  Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=80      Describes Input Records
   SORT     FIELDS=(1,15,CH,A)    Sorts Records
      .
      .
      .
   OUTFIL .
          .
          .
            TRAILER1=(20/,        Generates Report Trailer
                      33:'FEBRUARY SALES FIGURES INCLUDE RESIDUAL 2000',
                  'SALES NOT PREVIOUSLY RECORDED')
```

Figure 276. JCL and Required Control Statements

Figure 277 shows the trailer that is generated by the above TRAILER1 parameter.

```
FEBRUARY SALES FIGURES INCLUDE RESIDUAL 2000 SALES NOT PREVIOUSLY RECORDED
```

Figure 277. 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 2000 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.

```
// JOB      DSRPT              Signals the Start of Job
// ASSGN    SYS001,SYSLST      Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'      Assigns Logical Unit to Tape
// TLBL     SORTIN1,'SALES'    Contains Input Tape File Information
// ASSGN    SYS003,X'251'      Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1            Contains SORTWK File Information
// EXTENT   SYS003             Defines Direct Access Area for SORTWK
// EXEC     SORT               Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=80   Describes Input Records
   SORT     FIELDS=(1,15,CH,A) Sorts Records
      .
      .
      .
   OUTFIL .
          .
          .
        TRAILER2=(5'*','CONFIDENTIAL INFORMATION',
                               Generates Page Trailer
        5'*','CONFIDENTIAL INFORMATION',5'*')
            .
            .
            .
```

Figure 278. JCL and Required Control Statements

Figure 279 shows the trailer that is generated by the above TRAILER2 parameter.

```
*****CONFIDENTIAL INFORMATION*****CONFIDENTIAL INFORMATION*****
```

Figure 279. 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, 'CONFIDENTIAL INFORMATION', 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. The trailer's other entries specify the printing of another five asterisks followed by the literal string 'CONFIDENTIAL INFORMATION' 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.

The TOTAL and SUBTOTAL entries of OUTFIL's TRAILER parameters and subparameter are described below.

$$\text{TRAILER}\left\{\begin{matrix}1\\2\\3\end{matrix}\right\}=([c:]\left\{\begin{matrix}\text{TOTAL}\\\text{SUBTOTAL}\end{matrix}\right\}=(p,l,f\left[\begin{matrix},Mm\\,EDIT=(pattern)\end{matrix}\right][,SIGNS=(...)][,LENGTH=(n)]...)$$

Figure 280. TRAILER Format for TOTALS and SUBTOTALS

c: Specify the column in which you want the entry to begin printing.

TRAILERn Specify a TRAILER1 for a report trailer, a TRAILER2 for a page trailer, a TRAILER3 for a section trailer. NOTE: Specify a TRAILER3 *only* as a subparameter of the SECTIONS parameter.

TOTAL/TOT Use this parameter if you want the numeric data that you specify totaled at the end of a report, page, or section.

SUBTOTAL/SUB Use this parameter if you want a running total of the numeric data you specify.

p	Specify the beginning position of the numeric field on the input record. **Note:** This position should reflect any changes in the record due to any processing other than OUTFIL processing.
l	Specify the length of the numeric field that you want totaled.
f	Specify the format of the numeric data.
Mm	Specify one of ten SyncSort editing masks to format the total. (See "Mm Subparameter (Editing Masks)" on page 2.202.) **Note:** If neither Mm nor EDIT is specified, a default edit mask is used.
EDIT=(pattern)	Specify your own editing mask or pattern to format the total. (See "EDIT Subparameter" on page 2.200.)
SIGNS=(...)	Specify leading and/or trailing sign indicators to be used with the edit pattern or SyncSort mask.
LENGTH=(n)	Specify the length of the edited total to alter the default length of an edit pattern or SyncSort mask.

Note: If any specified TRAILER exceeds the length of the post-OUTREC record, be sure to code the LRECL parameter on the OUTFIL statement to match the length of the *longest* header or trailer. If this length is unknown, you may specify LRECL=132.

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.

```
// JOB      DSRPT                  Signals the Start of Job
// ASSGN    SYS001,SYSLST          Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'          Assigns Logical Unit to Tape
// TLBL     SORTIN1,'SALES'        Contains Input Tape File Information
// ASSGN    SYS003,X'251'          Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1                Contains SORTWK File Information
// EXTENT   SYS003                 Defines Direct Access Area for SORTWK
// EXEC     SORT                   Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=80       Describes Input Records
   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 281. JCL and Required Control Statements

Figure 282 shows the trailer that is generated by the above TRAILER1 parameter.

```
            SALES THIS PERIOD: $35,807.85   SALES TO DATE: $62,305.25
```

Figure 282. 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.

```
// JOB      INVLST               Signals the Start of Job
// ASSGN    SYS001,SYSLST        Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'        Assigns Logical Unit to Tape
// TLBL     SORTIN1,'INVOICE'    Contains Input Tape File Information
// ASSGN    SYS003,X'251'        Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1              Contains SORTWK File Information
// EXTENT   SYS003               Defines Direct Access Area for SORTWK
// EXEC     SORT                 Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=155    Describes Input Records
   SORT     FIELDS=(9,23,A,36,2,A,32,4,A),FORMAT=CH  Sorts Records
      .
      .
      .
   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 283. JCL and Required Control Statements

Figure 284 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 284. 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.

```
// JOB      ACTREC              Signals the Start of Job
// ASSGN    SYS001,SYSLST       Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'       Assigns Logical Unit to Tape
// TLBL     SORTIN1,'NEWINV'    Contains Input Tape File Information
// ASSGN    SYS003,X'251'       Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1             Contains SORTWK File Information
// EXTENT   SYS003              Defines Direct Access Area for SORTWK
// EXEC     SORT                Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=80    Describes Input Records
   SORT     FIELDS=(9,23,A,36,2,A,32,4,A),FORMAT=CH   Sorts Records
      .
      .
      .
   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 285. JCL and Required Control Statements

Figure 286 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 286. 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 the 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.

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.

The COUNT and SUBCOUNT entries of OUTFIL's TRAILER parameters and subparameter are described below.

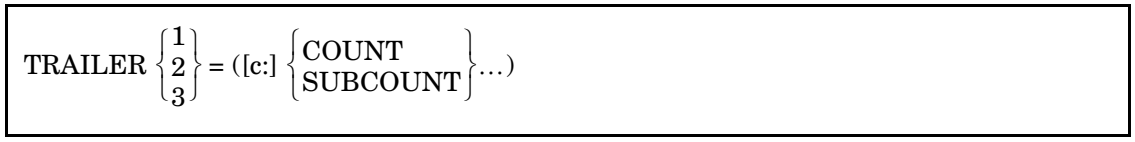


Figure 287. TRAILER Format for COUNT and SUBCOUNT

TRAILERn Specify TRAILER1 for a report trailer, TRAILER2 for a page trailer, TRAILER3 for a section trailer.

Note: Specify TRAILER3 *only* as a subparameter of the SECTIONS parameter.

COUNT Use this entry in TRAILER1 for the total number of output data records in the entire report; in a TRAILER2 for the total number of output data records on a page; in a TRAILER3 for the total number of output data records in a section. This number will be a right-justified, eight-byte field with leading zeros suppressed. The maximum value is 99999999.

SUBCOUNT Use this entry in a TRAILER1 if you want a count of the output data records in the entire report; in a TRAILER2 if you want a cumulative, or running, count of output data records on a given page and all pages that precede it; in a TRAILER3 if you want a cumulative, or running, count of output data records in that section and all sections that precede it. This number will be a right-justified, eight-character field with leading zeros suppressed. The maximum value is 99999999.

Note: You can use the COUNT and SUBCOUNT entries in conjunction with all other TRAILER entries.

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.

```
// JOB      INVRPT                Signals the Start of Job
// ASSGN    SYS001,SYSLST         Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'         Assigns Logical Unit to Tape
// TLBL     SORTIN1,'INVOICE'     Contains Input Tape File Information
// ASSGN    SYS003,X'251'         Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1               Contains SORTWK File Information
// EXTENT   SYS003                Defines Direct Access Area for SORTWK
// EXEC     SORT                  Signals Beginning of Sort Program
   SORT     FIELDS=(1,23,CH,A)    Sorts Records
      .
      .
      .
   OUTFIL
        .
        .
        .
     TRAILER1=(20/,40:'NUMBER OF CUSTOMERS WITH OUTSTANDING PAYMENTS:  ',
               COUNT)           Generates Report Summary
```

Figure 288. JCL and Required Control Statements

Figure 289 shows the trailer containing the record count.

```
            NUMBER OF CUSTOMERS WITH OUTSTANDING PAYMENTS:    52
```

Figure 289. 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, or 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.

```
// JOB      INVRPT                    Signals the Start of Job
// ASSGN    SYS001,SYSLST             Assigns Logical Unit to Printer
// ASSGN    SYS002,X'181'             Assigns Logical Unit to Tape
// TLBL     SORTIN1,'INVOICE'         Contains Input Tape File Information
// ASSGN    SYS003,X'251'             Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1                   Contains SORTWK File Information
// EXTENT   SYS003                    Defines Direct Access Area for SORTWK
// EXEC     SORT                      Signals Beginning of Sort Program
   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 290. JCL and Required Control Statements

Figure 291 shows the trailers containing the counts of records.

```
        .              .           .           .
        .              .           .           .
        .              .           .           .
RIBBIT TECHNOLOGIES   12/15/91      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
NORTHEAST INDUST       1/17/92      200.00       20.00
PULER COLLEGE          1/17/92    1,500.00       75.00
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 291. 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.

To generate multiple output files, code the OUTFIL statement as described below.

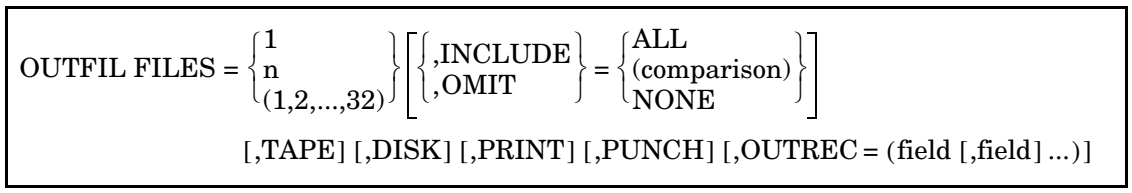


Figure 292. Multiple Output Parameters Format on OUTFIL

Note: See "OUTFIL Control Statement" on page 2.141 for the complete format of the OUTFIL statement.

FILES Use this parameter to connect an OUTFIL statement with one or more output files. (NOTE: FILES and the FILESOUT parameter on the SORT statement must coincide.) These numbers correspond to the output files defined in the JCL. The number 1 designates SORTOUT as the output file; the numbers 2 through 9 designate the output files SORTOF2 through SORTOF9; the numbers 10 through 32 designate the output files SORTO10 through SORTO32, respectively. If the FILES parameter is not specified, the entire output is directed to SORTOUT; the default is FILES=1.

INCLUDE Use this parameter to indicate which records you wish included in each output file. The default is to INCLUDE ALL records. To override the default, specify one or more logical conditions under which records are to be included or omitted. You may also override the default by specifying OMIT=ALL or INCLUDE=NONE. (See "OUTFIL Control Statement" on page 2.141 for the detailed format of comparisons.)

OMIT Use this parameter to indicate which records you wish omitted from each output file. The default is to OMIT=NONE records. To override the default, specify one or more logical conditions under which records are to be included or omitted. You may also override the default by specifying OMIT=ALL or INCLUDE=NONE. (See "INCLUDE/OMIT Control Statement" on page 2.39 for the detailed format of comparisons.)

TAPE Use this parameter to specify that the output file(s) is (are) to be stored on a tape device.

DISK Use this parameter to specify that the output file(s) is (are) to be stored on a disk device.

PRINT Use this parameter, the default, to specify that the output file(s) is (are) to be written to a printer.

PUNCH Use this parameter to specify that the output file(s) is (are) to be written to a card punch.

OUTREC Use the OUTREC parameter to reformat the output file. When specifying the beginning positions of data fields, remember to take into account any changes due to processing by an E15/E32 exit, the INREC control statement, the OUTREC control statement, and/or an E35 exit. (See "OUTREC Control Statement" on page 2.174 for the detailed format of OUTREC fields.)

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.

```
// JOB      CUSTRCD              Signals the Start of Job
// ASSGN    SYS001,X'180'        Assigns Logical Unit to Tape
// TLBL     SORTOUT              Contains Output Tape File Information
// ASSGN    SYS002,X'181'        Assigns Logical Unit to Tape
// TLBL     SORTOF2              Contains Output Tape File Information
// ASSGN    SYS003,X'182'        Assigns Logical Unit to Tape
// TLBL     SORTOF3              Contains Output Tape File Information
// ASSGN    SYS004,X'251'        Assigns Logical Unit to Disk
// DLBL     SORTIN1,'CUSTMRS'    Contains Input Disk File Information
// EXTENT   SYS004               Defines Direct Access Area for Input
// ASSGN    SYS005,X'253'        Assigns Logical Unit to SORTWK Device
// DLBL     SORTWK1              Contains SORTWK File Information
// EXTENT   SYS005               Defines Direct Access Area for SORTWK
// EXEC     SORT                 Signals Beginning of Sort Program
   RECORD   TYPE=F,LENGTH=80     Describes Input Records
   SORT     FIELDS=(10,15,CH,A),FILESOUT=3     Sorts Records
   OUTFIL   FILES=1,             Outfil Statement for SORTOUT
            INCLUDE=ALL,         Including All Records
            TAPE
   OUTFIL   FILES=2,             Outfil Statement for SORTOF2
            INCLUDE=(67,3,CH,EQ,C'USA'),  Including Only USA Records
            TAPE
   OUTFIL   FILES=3,             Outfil Statement for SORTOF3
            INCLUDE=(67,3,CH,EQ,C'EUR'),  Including Only European Records
            TAPE
```

Figure 293. JCL and Required Control Statements

Explanation: Creating the three requested output files requires specifying a SORTOUT and two SORTOFx data sets in the JCL as well as three separate 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 SORTOUT on the JCL. Likewise, specifying 2 connects its OUTFIL statement with the output file defined by SORTOF2, and so on.

The first output file (SORTOUT) in this example will contain the records from the input file (INCLUDE=ALL). The second output file will include the records that contain the character string 'USA' starting 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. Note that because each data set is being written to tape, the TAPE parameter is specified on each OUTFIL statement and because three output files are being written, the FILESOUT parameter is coded as FILESOUT=3 on the SORT control statement.

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.

```
// JOB       MULTOUT              Signals the Start of Job
// ASSGN     SYS001,SYSLST        Assigns Logical Unit to Printer
// ASSGN     SYS002,X'181'        Assigns Logical Unit to Tape
// TLBL      SORTOF2              Contains Output Tape File Information
// ASSGN     SYS003,X'251'        Assigns Logical Unit to Disk
// DLBL      SORTOF3              Contains Output Disk File Information
// EXTENT    SYS003               Defines Direct Access Area for Output
// ASSGN     SYS004,X'252'        Assigns Logical Unit to Disk
// DLBL      SORTIN1,'CUSTMRS'    Contains Input Disk File Information
// EXTENT    SYS004               Defines Direct Access Area for Input
// ASSGN     SYS005,X'253'        Assigns Logical Unit to SORTWK Device
// DLBL      SORTWK1              Contains SORTWK File Information
// EXTENT    SYS005               Defines Direct Access Area for SORTWK
// EXEC      SORT                 Signals Beginning of Sort Program
   RECORD    TYPE=F,LENGTH=80     Describes Input Records
   SORT      FIELDS=(10,15,CH,A),FILESOUT=3   Sorts Records
   OUTFIL    FILES=1,PRINT        Creates Print File
   OUTFIL    FILES=2,TAPE         Creates Tape File
   OUTFIL    FILES=3,DISK         Creates Disk Data Set
```

Figure 294. JCL and Required Control Statements

Explanation: Creating the three copies of the masterfile requires coding three OUTFIL statements. Each OUTFIL statement is coded with a FILES parameter whose number connects it to one of the output files defined in the JCL. The number 1 signifies the SORTOUT data set, and 2 and 3 signify SORTOF2 and SORTOF3, respectively. In addition to the FILES parameter, a device parameter specifies the output device to which each output data set is written. Thus, the SORTOUT data set goes to a printer, the SORTOF2 data set to a tape drive, and the SORTOF3 data set to disk.

Chapter 5. Job Control Language and Sample Control Statement Streams

Six Job Control Statements That May Be Needed for the Sort/Merge

Job control language for SyncSort for z/VSE follows the standard conventions for VSE/ESA and z/VSE. Here are six job control statements that may be needed for sorting or merging, and the format they should follow. For detailed JCL information, refer to the appropriate version of *System Control Statements*.

Brackets in the following examples indicate optional parameters; braces show that one choice must be made. Capital letters mean the capitalized word itself must appear. Capitalized words in brackets and braces, however, may be omitted if not chosen.

JOB

As the first card image in the job stream, the JOB statement signals the start of the job.

```
// JOB jobname
```

Figure 295. JOB Job Control Statement Format

ASSGN

The ASSGN statement assigns symbolic unit names to the input, output, and work devices that will be used for the sort/merge. (See Table 45 on page 5.4.) It connects the sort/merge program to the devices by referring the sort/merge devices to the logical unit block, the log-

ical unit block to the physical unit block, and finally, the physical unit block secures the devices specified. Because it must work with the system's logical unit block, this statement is often described as "assigning a logical name to a physical device."

It is not necessary to move all input files to the same type of device before sorting; input may be drawn from card, disk, or tape files in a single sort/merge. When different disk drives are used for SORTWK files, the characteristics of the smaller device do not constrain the way the larger device is used; the full track is utilized for each device.

An ASSGN statement is needed for every input, output, and work device used unless the system has already been set up with permanent symbolic unit names for the sort/merge program. In that case, this statement may be omitted.

```
// ASSGN SYSxxx,X'nnn' or // ASSGN SYSxxx,cuu
```

Figure 296. ASSGN Job Control Statement Format

TLBL/DLBL

These statements contain tape or DASD file information. A TLBL statement *must* be provided for every tape file with standard labels. A DLBL statement is necessary for every disk file. The format of the TLBL statement is given below in Figure 297.

```
// TLBL    filename [,'file-id'] [,date]
```

Figure 297. TLBL Job Control Statement Format

A DLBL statement would take the form below in Figure 298.

```
// DLBL    filename [,'file-id'] [,date] [,code]
```

Figure 298. DLBL Job Control Statement Format

Filenames can be user-supplied or the pre-chosen, "default" names, i.e., SORTIN1, SORTOUT, and SORTWK for input, output, and work filenames, respectively, can be selected. If user-supplied names are used, they must be specified in the FILNM or WORKNM parameter of the OPTION control statement. Up to 32 input files can be used for a sort or merge. Appropriate numbers can be attached to a single filename (for example, OWNAME1, OWNAME2,. . .,) or a unique name can be used for each file.

Note: The names may not be over seven characters long and the first character must be a letter.

The TLBL and DLBL statements may have continuation statements.

EXTENT

A DLBL statement is usually followed by an EXTENT statement. This defines the direct access area that will be used for that file.

The EXTENT statement must give the symbolic unit name of the device the extent occurs in. (This is the unit name specified in the ASSGN statement.) A user-supplied name may be used for the device so long as it is referenced in the SORTIN1, SORTOUT, SORTWK parameters of the OPTION control statement.

If an input file has multiple extents, only the first EXTENT statement needs to use either the "default" symbolic unit name or the user-provided name and must be referenced in the OPTION control statement. The remaining extents for the file may be given any symbolic unit names that are valid for the system.

```
// EXTENT    symbolic unit[,serial-number]
          [,type][,sequence-number]
          [,relative-track][,number-of-tracks]
```

Figure 299. EXTENT Job Control Statement Format

Job Control Statement: **Device Purpose**	**TLBL/ DLBL** **Filename**	**ASSGN, EXTENT** **Symbolic Unit Names Used When:**			
		Sort/Merge Reads Input & Writes Output	**User E15 Program Reads Input**	**User E35 Program Writes Output**	**User Program Reads Input & Writes Output**
Output	SORTOUT SORTOF2 . . SORTOF9 . . SORTO32	SYS001 . . . . SYS(O)	SYS001 . . . . SYS(O)		
Input	SORTIN1 . . SORTIN9 SORTI10 . . SORTI32	SYS(O+1) . . . . . . SYS(O+I)		SYS001 . . . . . . SYS(I)	
Work	SORTWK1 . . SORTWK9	SYS(O+I+1) . . SYS(O+I+W)	SYS(O+1) . . SYS(O+W)	SYS(I+1) . . SYS(I+W)	SYS001 . . SYS(W)
Joinwork	SORTWKA . . SORTWKI	SYS(N+I+W+1) . . SYS(N+I+W+J)		SYS(I+W+1) . . SYS(I+W+J)	
XSUM or XDUP	SORTXSM or SORTXDP	SYS(N+I+W+J+1)	SYS(N+W+J+1)	SYS(I+W+J+1)	SYS(W+J+1)
Checkpoint	SORTCKP	SYS000	SYS000	SYS000	SYS000
O=Number of Output Files Used J=Number of Joinwork files used I=Number of Input Files Used W=Number of Work Files Used					

Table 45. Symbolic Name Guide

EXEC

The EXEC statement signals the end of the job control information and the beginning of the sort/merge program. Program control statements follow the EXEC statement.

Note that certain SyncSort options can be specified in the PARM field of the EXEC statement. For more information about these options, see “Chapter 6. EXEC PARM Options”.

```
// EXEC [PGM=pgmname] [,SIZE=size] [,PARM='option1,option2,...']
```

Figure 300. EXEC Job Control Statement Format

Symbolic Filenames and Symbolic Unit Names for Job Control

Table 45 on page 5.4 lists the prechosen or “default” names SyncSort uses for the sort/merge’s input, output, and work files. If the checkpoint option is used, it must be referenced in the job control statements using the names shown in Table 45.

Setting up Disk Work File Statements

When multiple sort work files are used, performance can be enhanced by spreading the files among different disk devices. Work file sizes large enough to avoid secondary allocation also can reduce processing time.

There are two ways to set up disk work file statements. (Disk devices of different types may be used. Tape work files are not supported.)

Method 1

The filename "SORTWK1" is placed on one DLBL statement and "DA" is placed in the code parameter. This statement can be followed by 1 to 9 EXTENT statements identifying the number of tracks to be used. Only the first EXTENT statement, however, needs to get the default symbolic unit name or the user-supplied name that is specified in the OPTION control statement. EXTENT statements 2 to 9 may take any SYSnumber that is valid for the system but these numbers must be arranged in consecutive, *ascending* order. The *same* SYSnumber may be used for extents that occur on the same physical disk but *different* SYSnumbers must be used for extents that occur on different devices.

In addition, the WORK parameter of the SORT control statement must be coded WORK=DA.

One version of Method 1 is shown below in Figure 301.

Unless designated as required, the values are arbitrary and should be changed according to user specifications.

```
// ASSGN         SYS003,X'nnn'                                    1
// ASSGN         SYS004,X'nnn'                                    2
// ASSGN         SYS005,X'nnn'                                    3
// DLBL          SORTWK1,,0,DA                                    4
// EXTENT        SYS003,vvvvvv,1,0,aaaaa,zzzzz                    5
// EXTENT        SYS003,vvvvvv,1,1,aaaaa,zzzzz                    6
// EXTENT        SYS004,vvvvvv,1,2,aaaaa,zzzzz                    7
// EXTENT        SYS005,vvvvvv,1,3,aaaaa,zzzzz                    8
// EXTENT        SYS005,vvvvvv,1,4,aaaaa,zzzzz                    9
// EXTENT        SYS005,vvvvvv,1,5,aaaaa,zzzzz                   10
```

Figure 301. Disk Work File JCL Method 1

Notes:

1-3 The ASSGN statements assign symbolic unit names SYS003, SYS004, and SYS005 to three disk work files, each of which resides on a disk device. The variables in the X'nnn' parameter should be replaced with the address of each of the three devices.

4 The DLBL statement uses the *required* filename SORTWK1, places a comma to indicate a missing file-id, codes a zero to show no retention date, and places the *required* (for this method) DA in the code parameter to indicate a direct access file.

5-10 There are six EXTENT statements that describe six extents. The first two are located on device SYS003, the next area is found on SYS004, and the last three are on the disk assigned the name SYS005.

vvvvvv represents the volume serial number of the volume, and should be replaced by the actual number. If this number is less than 6 digits, the system will pad it on the left with zeros.

The 1 that appears in the type parameter in all the EXTENT statements specifies a data area.

The next parameter indicates the sequence number of the extents within this multi-extent file. The number of the first extent is 0, and this number is incremented by 1 on each following extent. The numbers may range from 0 to 255.

aaaaa represents the number of the track (relative to zero) where the extent is to begin. It should be replaced by the actual number, and may be from 1 to 5 digits long.

zzzzz gives the number of tracks the extent will use. It should be replaced by the actual number, and may be from 1 to 5 digits long.

Method 2

Using the filenames SORTWK1 to SORTWK9, one DLBL statement is required for each SD work file.

The symbolic unit names for the work files must be either user-created and referenced in the SORTWK parameter of the OPTION control statement, or the default names shown in Table 45 on page 5.4.

An example of Method 2 is given in Figure 302.

```
// ASSGN         SYS003,X'nnn'                          1
// DLBL          SORTWK1,'WORK AREA A',0,SD             2
// EXTENT        SYS003,vvvvvv,1,0,aaaaa,zzzzz          3
// ASSGN         SYS004,X'nnn'                          4
// DLBL          SORTWK2,'WORK AREA B',0,SD             5
// EXTENT        SYS004,vvvvvv,1,0,aaaaa,zzzzz          6
// ASSGN         SYS005,X'nnn'                          7
// DLBL          SORTWK3,'WORK AREA C',0,SD             8
// EXTENT        SYS005,vvvvvv,1,0,aaaaa,zzzzz          9
// ASSGN         SYS006,X'nnn'                         10
// DLBL          SORTWK4,'WORK AREA D',0,SD            11
// EXTENT        SYS006,vvvvvv,1,0,aaaaa,zzzzz         12
// ASSGN         SYS007,X'nnn'                         13
// DLBL          SORTWK5,'WORK AREA E',0,SD            14
// EXTENT        SYS007,vvvvvv,1,0,aaaaa,zzzzz         15
// ASSGN         SYS008,X'nnn'                         16
// DLBL          SORTWK6,'WORK AREA F',0               17
// EXTENT        SYS008,vvvvvv,1,0,aaaaa,zzzzz         18
```

Figure 302. Disk Work File JCL Method 2

Notes:

1 The ASSGN statement assigns the symbolic unit name SYS003 to the first work file. The file resides on a device at address X'nnn'.

2 The first DLBL statement uses the *required* filename SORTWK1 for the first work file. The file-id is 'WORK AREA A', there is to be no retention date, and the code is sequential disk (the SD designation). This last parameter *could* have been left out and received by default, as in the last DLBL statement in this example.

3 The EXTENT statement defines an area to be used for the SORTWK1 work file to which the symbolic name SYS003 has been assigned. *vvvvvv* represents the volume serial number; 1 indicates a data area; 0 shows a sequence number of zero; *aaaaa* represents the number of the track where the extent begins (relative to zero); and *zzzzz* represents the number of tracks used by the extent.

4-18 The instructions for statements 4-18 are the same as for statements 1-3, except that the SYSnumber is incremented by one for each work file.

Unlike Method 1, where the sequence number is always incremented by one for each extent, the sequence number in Method 2 EXTENT statements is always 0.

Calculating the Amount of Disk Work Space

As a general rule, 15 to 20 percent more disk work space should be planned for than the total input file size. If INREC or an E15 shorten or lengthen the input records, the disk work space required is also reduced or increased.

Whenever the initial allocation of SORTWK space is exhausted, SyncSort can request more work space. This space will be generated by a Disk Space Management package, if present, or may be supplied through operator communication.

Setting up Multiple SORTOUT Files

The multiple SORTOUT feature enables the user to specify a maximum of thirty-two SORTOUT files. The default names for the multiple SORTOUT files are SORTOUT, SORTOF2,...,SORTOF9, SORTO10,...SORTO32.

The DLBL and/or TLBL statements for filenames SORTOFx and SORTOxx are used to define the data sets used, in addition to SORTOUT, to receive sorted output. These values may be overridden by the FILNM parameter on the OPTION statement. SORTOFx and SORTOxx may be assigned to any disk, tape or unit record device.

The logical SYSnumbers default to SYS001,...,SYS032. These values may be overridden by the SORTOUT parameter on the OPTION statement. The number of output files is specified in the FILESOUT parameter on the SORT control statement.

Sample JCL/Control Statement Streams

One of the ways to initiate the sort/merge is by using job control language streams. This section illustrates sample JCL/control statement streams that you can use as models for your own applications.

Appendix C. “VSE/VSAM Space Management for SAM Files” explains how to set up job control language streams for VSAM managed SAM files. Information describing each line of JCL is provided.

A Standard-Labeled Tape Input and Output Sort

Figure 303 shows a sort with one tape input file, one tape output file, and a single extent disk work file. The default names have been used for the input, output and work files. The

control statements change the blocksize of the output file from that of the input file, sum a zoned decimal field when equal control fields are found, and request a dump in case the job does not run successfully.

```
// JOB          SCOFFLAW                                  1
// ASSGN        SYS001,X'280'                             2
// ASSGN        SYS002,X'281'                             3
// ASSGN        SYS003,X'142'                             4
// TLBL         SORTIN1,'MAY INPUT'                       5
// TLBL         SORTOUT,'MAY OUTPUT',31                   6
// DLBL         SORTWK1,,0,SD                             7
// EXTENT       SYS003,111111,1,0,1000,2000               8
// EXEC         SORT,SIZE=100K                            9
   SORT FIELDS=(5,22,CH,A),WORK=1                       10
   RECORD TYPE=V,LENGTH=500                             11
   INPFIL BLKSIZE=5000                                  12
   OUTFIL BLKSIZE=10000                                 13
   SUM FIELDS=(400,14,ZD)                               14
   OPTION DUMP                                          15
   END            UNPAID PARKING TICKETS* *MAY/*        16
/*                                                      17
/&                                                      18
```

Figure 303. Sample JCL/Control Statement Stream

Notes:

1 Creates a jobname.

2-4 Assigns symbolic unit name SYS001 to the output file at device address X'280', SYS002 to the input file at X'281', and SYS003 to the work file at X'142'.

5 Gives the filename of the input file and the file-id.

6 Gives the filename of the output file and the file-id, and requests a retention date of thirty-one days for the output file.

7 Gives the filename of the work file, places a comma for the omitted file-id, codes a 0 to specify no retention date, and identifies the type of file as a sequential disk.

8 Gives the symbolic unit name of the work file that occupies the extent, indicates that the volume serial number is 111111, codes 1 to specify a data area, codes 0 for the sequence number of the extent, indicates that the extent begins on track 1000, and that it runs for 2000 tracks.

9 Gives the name of the SyncSort program and indicates that the job will take up 100K space in virtual storage.

10 The SORT control statement shows that one field is to be sorted on. It begins on byte 5 of the record, has a length of 22 bytes, consists of character data, and is to be sorted in ascending sequence. There is to be one work file.

11 The RECORD statement shows that variable-length records are being sorted and that the maximum record length is 500 bytes.

12 The INPFIL statement shows that the blocksize of the records in the input file is 5000 bytes.

13 The OUTFIL statement shows that the blocksize of the records in the output file is 10,000 bytes.

14 The SUM statement requests that when two records with equal control fields are found, SyncSort is to sum a certain portion of the two records, place the total in one record, and delete the other record. The field to be summed begins on byte 400 of the record, is 14 bytes long, and has a zoned decimal format.

15 In case the job is not completed, a dump is to be processed.

16 The END statement tells SyncSort that the control statements for the job are now complete. The rest of the card image is a comment.

17 A standard job card image indicating the end of the JCL/control stream.

18 A standard job control statement indicating the end of the job stream.

An Unlabeled Tape Input and Output Sort

Figure 304 shows a sort with four unlabeled input tape volumes, five disk work files, and an unlabeled output tape file. In this case, by using the special INCLUDE feature, the sort has been used to build an output file containing unique information.

```
// JOB          FLYING                                                  1
// ASSGN        SYS001,X'280'                                           2
// ASSGN        SYS002,X'281'                                           3
// ASSGN        SYS003,X'282'                                           4
// ASSGN        SYS004,X'283'                                           5
// ASSGN        SYS005,X'284'                                           6
// ASSGN        SYS006,X'142'                                           7
// ASSGN        SYS007,X'142'                                           8
// ASSGN        SYS008,X'144'                                           9
// ASSGN        SYS009,X'144'                                           10
// ASSGN        SYSO10,X'145'                                           11
// DLBL         SORTWK1,,1                                              12
// EXTENT       SYS006,111111,1,0,200,50                                13
// DLBL         SORTWK2,,1                                              14
// EXTENT       SYS007,111111,1,0,2400,50                               15
// DLBL         SORTWK3,,1                                              16
// EXTENT       SYS008,444444,1,0,700,50                                17
// DLBL         SORTWK4,,1                                              18
// EXTENT       SYS009,444444,1,0,1800,50                               19
// DLBL         SORTWK5,,1                                              20
// EXTENT       SYS010,555555,1,0,3000,50                               21
// EXEC         SORT                                                    22
   SORT         FIELDS=(50,1,CH,A),FILES=4,WORK=5                       23
   RECORD       TYPE=F,LENGTH=100                                       24
   INPFIL       BLKSIZE=500,CLOSE=(UNLD,RALL)                           25
   OUTFIL       BLKSIZE=500                                             26
   INCLUDE      COND=(62,8,EQ,C'RETURNED',AND,                          27
                75,14,EQ,C'FEAR OF FLYING'),FORMAT=CH                   28
   OPTION       LABEL=(U,U,U,U,U),STORAGE=60K                           29
   END          NATIONAL RETURNS FEAR OF FLYING                         30
/*                                                                      31
/&                                                                      32
```

Figure 304. Sample JCL/Control Statement Stream

Notes:

1 Gives the name of the job.

2-11 Assigns symbolic unit name SYS001 to an output file, SYS002-SYS005 to 4 input files, and SYS006-SYS010 to 5 work files. (The distinction between input and work files is made in the SORT statement.)

12-21 Sets up the 5 work files according to "Method 2" on page 5.7. Each DLBL card image gives the filename, omits the file-id, specifies a one-day retention date (to protect the files from being overlaid with data from a job operating in another partition), and by omitting the last parameter, specifies a sequential disk file.

The EXTENT statements give the symbolic unit names of the files. They note that SYS006 and SYS007 are on volume 111111, SYS008 and SYS009 on volume 444444, and SYS010 on volume 555555. The 1 on each card image specifies a data area, and the 0 gives the required sequence number for sequential disk files. A beginning track number has been specified for each extent, and each extent will take 50 tracks.

22 Gives the SyncSort program name.

23 The SORT statement identifies one control field. It starts on byte 50 of the record, is 1 byte long, has character format, and is to be sorted in ascending sequence.

The FILES parameter gives four as the number of input files, and the WORK parameter gives five as the number of work files.

24 The RECORD statement shows fixed-length records that are 100 bytes long.

25-26 The INPFIL and OUTFIL statements specify a blocksize of 500 bytes. All four input tape volumes will be rewound, unloaded, and released at end-of-file. The Release option should only be used in the absence of a tape management system.

27-28 The INCLUDE statement tells the sort to select any record with 'RETURNED' in bytes 62-69 and 'FEAR OF FLYING' in bytes 75-88, and write them to the output file. Records that do not meet these requirements are not to be included. At the end of this sort, the programmer will have complete data on the returns of a book sold on consignment throughout the country. A report may be printed at a later time.

29 The OPTION statement shows that the output and four input files are unlabeled, and that SyncSort may use 60K in main storage.

30 The END control statement gives a comment.

31-32 The standard job control end card images.

A Disk Input and Output Sort

Figure 305 shows a sort with disk input, disk output and two disk work devices. The object of this sort is to create a paycheck file from the input payroll data. Later, paychecks will be printed from this output disk. The OUTREC feature has been used to select the pay, department, and name fields from the input record for the new, shortened output records.

```
// JOB           PAYROLL                                          1
// ASSGN         SYS001,X'142'                                    2
// ASSGN         SYS002,X'142'                                    3
// ASSGN         SYS003,X'144'                                    4
// ASSGN         SYS004,X'145'                                    5
// DLBL          SORTOUT,'PAYCHECKS',30                           6
// EXTENT        SYS001,111111,1,0,2500,400                       7
// DLBL          SORTIN1,'PAYFILE',30                             8
// EXTENT        SYS002,111111,1,0,500,2000                       9
// DLBL          SORTWK1,,0,DA                                   10
// EXTENT        SYS003,444444,1,0,200,500                       11
// EXTENT        SYS003,444444,1,1,1000,500                      12
// EXTENT        SYS003,444444,1,2,2300,500                      13
// EXTENT        SYS004,555555,1,3,450,500                       14
// EXTENT        SYS004,555555,1,4,1060,500                      15
// EXTENT        SYS004,555555,1,5,1800,500                      16
// EXEC          SORT                                            17
   SORT          FIELDS=(64,6,PD,A,2,3,CH,A)                     18
   RECORD        TYPE=F,LENGTH=190                               19
   INPFIL        BLKSIZE=3800                                    20
   OUTREC        FIELDS=(64,6,2,3,10,30)                         21
   OUTFIL        BLKSIZE=780                                     22
   OPTION        PRINT=CRITICAL,ROUTE=LOG                        23
   END           ***OCTOBER 15 PAYCHECKS***                      24
/*                                                               25
/&                                                               26
```

Figure 305. Sample JCL/Control Statement Stream

Notes:

1 Creates a jobname.

2-5 Assigns symbolic unit names to the output, input, and two work files.

6 The DLBL statement gives the filename SORTOUT to the output file, gives 'PAYCHECKS' as the file-id, asks for a 30-day retention period, and specifies (by omitting the parameter) a sequential disk.

7 The EXTENT statement gives SYS001 for the symbolic unit name of the output file, gives 111111 as the volume serial number, codes 1 to specify a data area, codes 0 to show the sequence number, gives 2500 as the beginning track number, and gives 400 as the number of tracks it will use.

8-9 This pair of DLBL-EXTENT statements for the input file follows the same format as the pair for the output file; however, the input file is much larger (2000 tracks) than the output file (400 tracks). This is because the input record will be

reformatted on the OUTREC statement so that at the end of the sort, when it becomes output, it will be 151 bytes shorter.

10 This DLBL statement gives the filename for the work file as SORTWK1, omits a file-id, specifies no retention date, and codes DA to indicate a direct access file.

11-16 The EXTENT statements give the locations and lengths of the six extents of the work file described on the DLBL statement (item 10). The sequence number is incremented by one for each extent.

17 The EXEC statement gives SORT as the SyncSort program name.

18 The SORT statement shows two control fields. The first, or major field, starts on byte 64 of the input record, is 6 bytes long, has packed decimal format, and is to be sorted according to ascending sequence. The second, or lesser field, begins on byte 2 of the record, is 3 bytes long, has character data, and is to be sorted according to ascending sequence.

19 The RECORD statement codes an F for fixed-length records and indicates that the length of the input record is 190 bytes. This is the l_1 value. The l_2 and l_3 values will be calculated by the sort and the correct values will be used when it is time to change the length of the record for the output file.

20 The INPFIL statement gives the blocksize of the input file as 3800 bytes. This is twenty records to a block.

21 The OUTREC statement shows how the input record is to be reconstructed to form the output record. The FIELDS parameter shows that three fields will be picked up. The first starts on byte 64 of the input record and is 6 bytes long. The second starts on byte 2 of the input record and is 3 bytes long. The third starts on byte 10 of the input record and is 30 bytes long. These are the pay, department number, and employee name fields, respectively, that will later be reproduced on paychecks. They will now run consecutively from byte 1 through byte 39 of the output record, and will be edited with spacing later when they are printed.

22 The OUTFIL statement gives the blocksize of the output file as 780 bytes. Although this is still twenty records to a block, the output file blocksize is considerably shorter than that of the input file.

23 The OPTION statement requests that only critical messages be issued and that they appear on the console.

24 The END statement indicates the end of the control statements. A comment is given.

25-26 The standard job control end statement.

A Tape Input and Output Merge

Figure 306 shows a merge with 4 standard-labeled tape files as input and one standard labeled tape file as output. The input files are already in sequence, as is required for all merge programs. In this case, a membership file is being updated. The ALTSEQ statement is used so that any small size numbers that appear in the member code field will be collated as equal to their matching large size numbers in the EBCDIC series. Also, the numbers are to be collated in descending order so that 9, 8, 7,... will collate before 0, 1, 2,...

```
// JOB          JOINERS                                              1
// ASSGN        SYS001,X'230'                                        2
// ASSGN        SYS002,X'231'                                        3
// ASSGN        SYS003,X'282'                                        4
// ASSGN        SYS004,X'283'                                        5
// ASSGN        SYS005,X'284'                                        6
// TLBL         MERGOUT,'MASTER-25',92                               7
// TLBL         MERGIN1,'MASTER-24'                                  8
// TLBL         MERGIN2,'APRIL JOINERS'                              9
// TLBL         MERGIN3,'MAY JOINERS'                               10
// TLBL         MERGIN4,'JUNE JOINERS'                              11
// EXEC         SORT                                                12
   MERGE FIELDS=(1,8,AQ,D,10,30,CH,A),FILES=4                      13
   RECORD TYPE=F,LENGTH=100                                        14
   INPFIL BLKSIZE=1000                                             15
   OUTFIL BLKSIZE=1000                                             16
   OPTION FILNM=(MERGOUT,MERGIN1,MERGIN2,                          17
     MERGIN3,MERGIN4)
   ALTSEQ CODE=(B0F0,B1F1,B2F2,B3F3,B4F4,B5F5,                     18
     B6F6,B7F7,B8F8,B9F9) COLLATE
*    SMALL NUMBERS EQUAL TO LARGE
   SUM FIELDS=NONE                                                 19
   END 3-MONTH MEMBERSHIP UPDATE                                   20
/*                                                                 21
/&                                                                 22
```

Figure 306. Sample JCL/Control Statement Stream

Notes:

1 Creates a jobname.

2-6 Assigns a symbolic unit name to the output file at device address X'230'. Assigns symbolic unit names to four input files on four separate devices and gives the device addresses.

7-11 The TLBL statements create filenames for the five tape files. Since these are user-supplied names, they must be referenced in the FILNM parameter of the OPTION control statement. The output file is given the updated file-id 'MASTER-25' and a retention date of 92 days (to cover July, August, and Sep-

tember) is requested. The MERGIN1 file is the old master and MERGIN2, MERGIN3, and MERGIN4 contain new data.

12 The EXEC statement gives SyncSort's program name and signals that the sort/merge control statements are about to begin.

13 The MERGE statement shows there are two control fields on which the merge will be based. The first control field starts on byte 1 of the record, is 8 bytes long, is to be merged according to an alternate collating sequence (this will be supplied in the ALTSEQ statement), and the sequence is to be referenced in descending order. The second control field starts on byte 10 of the record, is 30 bytes long, has character format data, and is to be merged according to ascending order. The FILES parameter specifies four input files.

14 The RECORD statement notes that the TYPE of records are fixed-length, and that their LENGTH is 100 bytes.

15-16 The INPFIL and OUTFIL statements both show a blocksize of 1000 bytes.

17 Since MERGOUT/MERGIN filenames were substituted for the default SORTOUT/SORTIN filenames, the user-supplied names must be referenced here in the FILNM parameter.

18 The ALTSEQ statement gives the alternate collating sequence that will be used for the first control field. In this case, the nine pairs of hex numbers specify that the small numbers represented in the EBCDIC series by B0-B9 are to be collated as identical to the large numbers represented by F0-F9. This statement includes a comment.

19 The SUM statement guarantees that the output membership file will have only one record per sort control key. Assuming the sort key identifies the member, this would mean that the updated membership file will mention each member exactly once. Since EQUALS is the default for a merge, when equal-keyed records are encountered, it is the record from the earliest input file that is retained; i.e., records from the old master file, then the April joiners, then the May joiners, and finally, the June joiners. The EQUALS default ensures that the updated master membership file identifies the member's earliest successful application for membership.

20 The END control statement indicates the end of the control statements and includes a comment.

21-22 Standard job control end statement.

A Sort Using Card Input

In Figure 307 and Figure 308, the SYSIPT parameter has been issued so that cards may be used as direct input to the sort. No work files are used for this sort. This is because the number of records is small enough to be sorted incore, and data can go directly to the output phase without ever needing to be written out to an intermediate work device. An approximate guide for determining whether a job is small enough to be sorted completely incore is:

Total bytes of input data + output BLKSIZE + 20K ≤ Sort program's storage.

The input in the following examples consists of the 200 report cards with the highest averages for one high school semester. They will be sorted by class and student names so that the disk output can later be printed as "The Dean's List." Either example will produce the same results.

Note: Some older programs use E15 programs to read card input. Since SyncSort supports direct reading of card input, these programs should be converted to use SYSIPT for increased efficiency.

```
// JOB         TOPGRADE                                              1
// ASSGN       SYS001,X'142'                                         2
// DLBL        SORTOUT,'DEAN'S LIST'                                 3
// EXTENT      SYS001,111111,1,0,100,4                               4
// ASSGN       SYS002,SYSIPT                                         4A
// EXEC        SORT                                                  5
   SORT        FIELDS=(79,2,CH,A,2,20,CH,A),WORK=0                   6
   RECORD      TYPE=F,LENGTH=80                                      7
*              NO INPFIL STATEMENT                                   8
   OUTFIL      BLKSIZE=1600                                          9
   END                                                              10
   Report Card 1                                                    11
   .
   .
   .
   Report Card 200                                                  12
/*                                                                  13
/&                                                                  14
```

Figure 307. Sample JCL/Control Statement Stream

```
// JOB          TOPGRADE                                              1
// ASSGN        SYS001,X'142'                                         2
// DLBL         SORTOUT,'DEAN'S LIST'                                 3
// EXTENT       SYS001,111111,1,0,100,4                               4
// EXEC         SORT                                                  5
   SORT         FIELDS=(79,2,CH,A,2,20,CH,A),WORK=0                   6
   RECORD       TYPE=F,LENGTH=80                                      7
   INPFIL       SYSIPT                                                8
   OUTFIL       BLKSIZE=1600                                          9
   END                                                               10
   Report Card 1                                                     11
   .
   .
   .
   Report Card 200                                                   12
/*                                                                   13
/&                                                                   14
```

Figure 308. Sample JCL/Control Statement Stream

Notes:

1 Creates a jobname.

2 Assigns a symbolic unit name SYS001 to the output file at device address X'142'.

3 Gives the filename and file-id of the output file.

4 Gives the symbolic unit name of the output file that occupies the extent, gives 111111 as the volume serial number, codes a 1 to indicate a data area, codes a 0 for the sequence number, gives 100 as the beginning track number, and states that the extent will use four tracks.

4A Assigns symbolic unit name SYS002 to the input file on the system input device.

5 Gives the program name.

6 The SORT control statement shows two fields to be sorted on. The first begins on byte 79 of the record, is 2 bytes long, has character data format, and is to be collated in ascending order. The second begins on byte 2, is 20 bytes long, has character data format and is to be collated in ascending order. The WORK=0 parameter states that no work files are going to be used. Since the default for omitting this parameter is 1 direct access work device, code a 0 here if no work device is to be used.

7 The RECORD statement shows that fixed-length, 80-byte records are being used.

8 The INPFIL statement is not required when SYSIPT is assigned to SYS002.

9 The OUTFIL statement indicates that the blocksize of the output file is to be 1600 bytes.

10 The END statement signals the end of the control statements and is required with card input.

11-12 The 200 input cards are inserted here.

13 The standard job control end card signals the end of the input.

14 The standard job control end card signals the end of the program.

A Disk Input and Output JOIN

Figure 309 shows a JOIN application with one fixed-length and one variable-length disk input file. The records in these two files are filtered based on INCLUDE parameters and then reformatted into variable-length output records to a disk file.

```
// JOB JOIN                                                          1
// ASSGN      SYS001,250                                             2
// ASSGN      SYS002,251                                             3
// ASSGN      SYS003,252                                             4
// ASSGN      SYS004,253                                             5
// ASSGN      SYS005,253                                             6
// ASSGN      SYS006,253                                             7
// DLBL       SORTOUT,'YTD.COMBINED',30                              8
// EXTENT     SYS001,111111,1,0,2500,100                             9
// DLBL       SORTIN1,'YTD.OLD',30                                  10
// EXTENT     SYS002,222222,1,0,2500,100                            11
// DLBL       SORTIN2,'YTD.NEW',30                                  12
// EXTENT     SYS003,333333,1,0,2500,100                            13
// DLBL       SORTWK1,'SORT.WORK.1',0                               14
// EXTENT     SYS002,222222,1,0,2500,100                            15
// DLBL       SORTWKA,'SORT.WORK.A',0                               16
// EXTENT     SYS005,222222,1,0,2600,100                            17
// DLBL       SORTWKB,'SORT.WORK.B',0                               18
// EXTENT     SYS006,222222,1,0,2700,100                            19
// EXEC       SORT                                                  20
 JOINKEYS     FILE=F1,FIELDS=(13,6,A),TYPE=F,LRECL=80,              21
              BLKSIZE=2400,INCLUDE=(19,2,CH,EQ,C' C')               22
 JOINKEYS     FILE=F2,FIELDS=(5,6,A),TYPE=V,LRECL=200,              23
              BLKSIZE=1700,INCLUDE=(11,10,ZD,GE,100000)             24
 REFORMAT     FIELDS=(F2:1,4,                                       25
                      F2:1,6,                                       26
                      F2:5,6,                                       27
                      F2:74,6,                                      28
                      F2:11,10,                                     29
                      F2:23,20,                                     30
                      F2:21)                                        31
 SORT        FIELDS=(5,6,BI,A) ,EQUALS,JOINWORK=2                   32
 OUTFIL      BLKSIZE=1700                                           33
 END                                                                34
/*                                                                  35
/&                                                                  36
```

Figure 309. A disk input and output JOIN

Notes:

1 Creates a job name.

2-7 Assigns symbolic unit names to the output, input, one sort work, and two join work files.

8-9 Gives DLBL and EXTENT information for the SORTOUT disk file 'YTD.COMBINED'.

10-11 Gives DLBL and EXTENT information for the SORTIN1 disk file 'YTD.OLD'.

12-13 Gives DLBL and EXTENT information for the SORTIN2 disk file 'YTD.NEW'.

14-15 Gives DLBL and EXTENT information for the SORTWK1 sort work disk file.

16-19 Gives DLBL and EXTENT information for SORTWKA and SORTWKB for the internal join work disk files.

20 The EXEC statement gives SORT as the SyncSort program name.

21-22 The JOINKEYS control statement is used to define the join key, record format, record length, and block size for file F1, or SORTIN2. The INCLUDE parameter is used to include all records with a 'C' in positions 19-20.

23-24 The JOINKEYS control statement is used to define the join key, record format, record length, and block size for file F2, or SORTIN2. The INCLUDE parameter is used to include all records that have a ZD value greater than 100000 in positions 11-20.

25-31 The REFORMAT control statement is used to combine data fields from both F1 and F2 based on the positions and lengths specified. Since the JOIN control statement is not specified, all equally-keyed records based on the keys specified in the JOINKEYS control statements will be kept and all unequally keyed records will be discarded. The reformatted output record will be variable-length because statement 25 copies the RDW from F2 and statement 31 copies the position (p) without the length (l) from F2.

32 The SORT control statement shows that all reformatted records coming out of the internal join sort will be sorted on the binary field starting in position 5 for a length of 6 in ascending sequence. The EQUALS parameter indicates that all equally keyed records will have their original order preserved through the entire sort process. JOINWORK=2 indicates that two join work files, SORTWKA and SORTWKB, will be used for the join sorting process. Since no WORK parameter is present, the default of one sort work is used, SORTWK1.

33 The OUTFIL control statement indicates the output disk block size for SORTOUT.

34 The END control statement indicates the end of the sort control statements.

35-36 The standard job control statements indicate the end of the job stream.

VSAM Input and Output

Figure 310 is a sample JCL/control statement stream that can be used to run an actual job with VSAM input and output files. This example illustrates the manner in which a job stream should be set up to facilitate the sorting of an entry-sequenced data set.

```
// JOB        SORTTEST                                                  1
// ASSGN      SYS001,X'250' OUTPUT FILE                                 2
// ASSGN      SYS002,X'251' INPUT FILE                                  3
// ASSGN      SYS003,X'252' SORT WORK                                   4
// ASSGN      SYS004,X'253' SORT WORK                                   5
// ASSGN      SYS005,X'254' SORT WORK                                   6
// DLBL       SORTOUT,'SORTOUT',,VSAM                                   7
// EXTENT     SYS001,PACKI                                              8
// DLBL       SORTIN1,'SORTIN1',,VSAM                                   9
// EXTENT     SYS002,PACK2                                             10
// DLBL       SORTWK1,,0                                               11
// EXTENT     SYS003,PACK3,1,0,4769,70                                 12
// DLBL       SORTWK2,,0                                               13
// EXTENT     SYS004,PACK4,1,0,4839,70                                 14
// DLBL       SORTWK3,,0                                               15
// EXTENT     SYS005,PACK,1,0,4909,70                                  16
// EXEC       SYNCSORT,SIZE=100K                                       17
   OPTION     PRINT=ALL,ROUTE=LST                                      18
   SORT       FIELDS=(11,4,BI,A),FILES=1,WORK=3                        19
   RECORD     TYPE=F,LENGTH=(0063,,0063)                               20
   INPFIL     VSAM,TOL                                                 21
   OUTFIL     ESDS,TOL                                                 22
   END                                                                 23
/*                                                                     24
/&                                                                     25
```

Figure 310. Sample JCL/Control Statement Stream

Notes:

1 Creates a jobname.

2-6 Assigns I/O disk devices for input, output and work files.

7-8 Specifies a VSAM disk file as output. The file should have been defined (by a DEFINE CLUSTER) earlier.

9-10 Specifies a VSAM disk file as input.

11-16 Specifies work files on disk.

17 Specifies the SIZE parameter to restrict the amount of virtual storage used by SyncSort. There must be sufficient virtual storage left in the partition for VSAM use.

18 The OPTION statement indicates the following:

- All sort/merge messages are to be printed.

- Messages are to be routed to SYSLST.

19 The SORT statement specifies the sort control field and one input file and three sort work files.

20 The RECORD statement specifies record information for an entry-sequenced data set. Fixed-length records are to be sorted. The input and output record lengths are 63 bytes.

21 The INPFIL statement specifies that the input file is VSAM and a warning code will be accepted when opening the VSAM input file.

22 The OUTFIL statement specifies that the output file is entry-sequenced and a warning code will be tolerated when opening a VSAM output data set.

23 The END statement signals the end of the control statements.

24-25 The standard job control end statements.

Using the FIELDS=COPY Parameter

The FIELDS=COPY parameter, when coupled with SyncSort's I/O techniques, may be used as a high performance file editing tool. By using this facility, the user may avoid writing file edit programs. Certain control statements can be input in a predetermined sequence to accomplish the desired file edit.

Figure 311 and the related record layout in Figure 312 show how to perform the file edit. In this example, the FIELDS=COPY parameter of the MERGE control statement is used in conjunction with the INCLUDE/OMIT and INREC/OUTREC control statements. For detailed descriptions, refer to "INCLUDE/OMIT Control Statement" on page 2.39, "INREC Control Statement" on page 2.83, and "OUTREC Control Statement" on page 2.174.

In this example, certain data concerning exployees in a fictitious department 44 are to be extracted from a master file and placed on an edited file.

```
// JOB          COPY AND EXIT                                          1
// ASSGN        SYS001,X'280'                                          2
// ASSGN        SYS002,X'281'                                          3
// TLBL         SORTOUT,'EDITED FILE'                                  4
// TLBL         SORTIN,'MASTER FILE'                                   5
// EXEC         SORT                                                   6
   MERGE        FIELDS=COPY                                           7
   RECORD       TYPE=F,LENGTH=200                                     8
   INCLUDE      COND=(24,7,EQ,C'DEPT 44'),FORMAT=CH                   9
   OUTREC       FIELDS=(110,40,6,10,C'$ ',42,4,PD,M2)                10
   INPFIL       BLKSIZE=2000                                         11
   OUTFIL       BLKSIZE=6200                                         12
   END                                                               13
/*                                                                   14
/&                                                                   15
```

Figure 311. Sample JCL/Control Statement Stream

Notes:

1 Creates a jobname.

2-3 Assigns symbolic unit name SYS001 to the output file at device address X'280' and SYS002 to the input file at X'281'.

4 Gives the filename of the output file and file-id.

5 Gives the filename of the input file and file-id.

6 Gives the name of the sort program.

7 The MERGE statement indicates that the FIELDS=COPY parameter will be used to copy the records.

8 The RECORD statement shows that fixed-length records are being processed and the record length is 200 bytes.

9 The INCLUDE statement indicates that SyncSort will process only records of employees in department 44. If the data beginning in byte 24 for a length of 7 bytes is equal to DEPT 44, the record will be copied.

10 For records that meet the criteria established by the INCLUDE statement, the OUTREC statement will output only the following data:

- Employee name (found at position 110 of a record for 40 bytes)
- Payroll code (found at position 6 of a record for 10 bytes)

- Payroll amount (the 4 byte PD field found at position 42 is converted to printable format and edited with SyncSort editing mask M2. Leading zeros will be suppressed and a minus sign will be included if the number is negative. A dollar sign and a blank will precede the number: $ II,IIT.TTS).

  The output records will be 62 characters in length.

11 The INPFIL statement gives the blocksize of the input file as 2000 bytes.

12 The OUTFIL statement gives the blocksize of the output file as 6200 bytes.

13 The END statement indicates the end of the control statements.

14-15 The standard job control end statements.

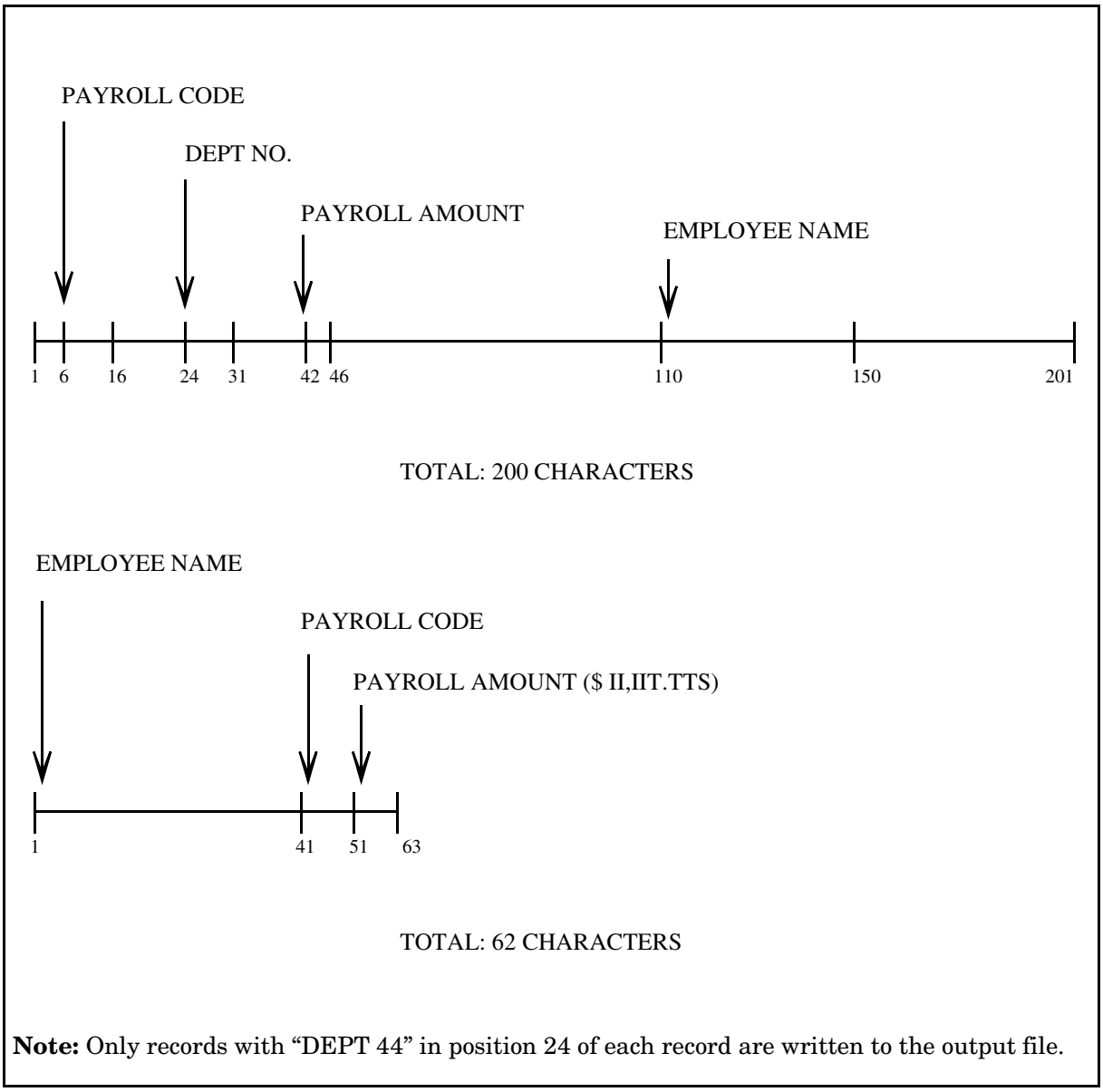


Figure 312. Sample Record Layout

Using the FIELDS=COMPARE Parameter

The FIELDS=COMPARE parameter of the MERGE statement provides a file identity test. In Figure 313, the parameter is used to test the functional equivalence of two routines by COMPAREing the output files they produced from the same input data set.

Suppose changes were made to a copy of the program that was responsible for the master file described in the previous problem (see Figure 311 and Figure 312)—changes that should not have affected the make-up of the master file. Therefore, the manner of performing the identity check on the master file will be as indicated in Figure 313. The JCL/control

statement stream is similar to that of the previous problem except for the COMPARE option and the absence of output file specifications. Because there is no SORTOUT with FIELDS=COMPARE, SORTIN files will start with SYS001. With FIELDS=COMPARE, the merge is not done. Moreover, it is not possible to use an option such as INCLUDE/OMIT with this parameter.

```
// JOB           COMPARE
// DLBL          SORTIN1,'FILEIN1',0
// EXTENT        SYS001,222222
// DLBL          SORTIN2,'FILEIN2',0
// EXTENT        SYS002,222222
// EXEC          SYNCSORT,SIZE=50K
   MERGE         FIELDS=COMPARE,ORDER=2
   RECORD        TYPE=F,LENGTH=200
   INPFIL        BLKSIZE = 200
   END
/*
/&
```

Figure 313. Sample JCL/Control Statement Stream

The files are processed record pair by record pair. If the input files are identical, the MERGE will print such a message. A second message exists for the case where the files are unequal (e.g., do not have the same number of records) but where the first n records of the larger file are identical to the smaller file in its entirety (i.e., up to the EOF flag); this message also gives the number of records processed from each file, clarifying which of the two has the extra record(s). The extra record(s) must all be at the end of the larger file in order to get this message.

In the two cases discussed thus far, all paired records compared as equal; in the first case, the files as well as the records compared as equal. In the second case, an unequal file condition was detected.

When an unequal record condition is detected, a different message is issued. An unequal record condition exists when two non-EOF records paired by the merge are not byte-for-byte identical such as, for example, when the RDWs of two variable-length records differ. After the "unequal records" message, 2 PDUMPs are immediately produced:

1. The first PDUMP shows each SORTIN DTF.

2. The second PDUMP displays the unequal records; the record from file one is followed by the corresponding file two record whose address appears in Register 8.

The MERGE then terminates with a message identifying the files as not equal and indicating the number of records processed.

All these messages are directed to SYSLST and no provision is made for SYSLST's being assigned to tape or disk. "A" messages generated by the FIELDS=COMPARE parameter do not appear on the console.

Chapter 6. EXEC PARM Options

EXEC PARM options can be specified to provide processing information for JCL-initiated and program-invoked applications. EXEC PARM options override both installation defaults (set by SYNCMAC) and parameters specified on the OPTION control statement. They can be useful when a parameter change is needed for one instance of the sort without changing the installation defaults for all sorts, or without recompiling a program that invokes the sort.

Specify the PARM options in the EXEC statement as follows:

```
// EXEC programname,...,PARM='option1,option2,...,optionn'
```

Figure 314. PARM Options in EXEC Statement

The maximum number of characters that can be specified in one PARM string is 100. To specify more than 100 characters, up to two additional PARM strings may be used on the EXEC statement.

Note: Since other program products also may be using the EXEC PARM, SyncSort will not perform its normal syntax validation on the PARM options. If SyncSort does not recognize a PARM option, the option will be ignored.

PARM Option Summary

The available SyncSort PARM options are summarized in Table 46.

PARM Option Name	PARM Option Function
CENTWIN/CW	Defines a 100-year window that determines the applicable century for 2-digit year data.
DIAG	Produces a listing of diagnostic messages and PDUMPs.
EQUALS/NOEQUALS	Indicates whether or not the original order of records with equal control fields needs to be preserved.
ERASE/ERASEWK	Erases contents of work files that were opened by the sort.
IGNRL4/NOIGNRL4	Indicates whether or not SyncSort should check that the variable-length input records are at least as large as the specified or computed l_4 value.
INCOR=OFF	Specifies that an incore sort is NOT to be performed.
PRINT	Determines which SyncSort messages are displayed.
ROUTE	Determines where SyncSort messages are displayed.
SF	Overrides the SORT FIELDS collating parameters for invoked sorts.
SYMNAMES	Specifies that the SyncSort dictionary feature is to be used to map symbolic names in control statements.
SYMNLIB	Specifies a library search chain used by the SyncSort dictionary feature.
SYMNOUT	Specifies where the dictionary listing is to be printed when using the SyncSort dictionary feature.
VL5L6L7	Analyzes a variable-length file to determine optimum values for l_5, l_6, and l_7, which can be used for subsequent sorts of the same file as LENGTH parameters on the RECORD control statement to improve performance.
VSCORE	Specifies the upper bound of virtual storage below the 16-megabyte line in which SyncSort will run.
VSCORET	Specifies the maximum amount of storage that SyncSort can use when it is running in a partition that has storage above the 16-megabyte line.
WORKNM	Changes the four-character prefix of all SORTWK names from the default, SORT.

Table 46. PARM Option Names and Functions.

SyncSort PARM Options

CENTWIN/CW

Figure 315. CENTWIN PARM Format

The CENTWIN PARM is an aid in overriding the CENTWIN=parameter on the OPTION control statement. It is most useful for applications that call the sort several times for which different values of the CENTWIN parameter are needed, or for adding a CENTWIN value to existing applications without altering the compiled program. For details on the CENTWIN parameter, see “CENTWIN Parameter (Optional)” on page 2.119.

CW=n is accepted as a synonym. The requirements for specifying “n” are the same as for OPTION CENTWIN.

The following is an example of the CENTWIN override for a program that invokes the sort only once.

```
// EXEC INVOKED,SIZE=512K,PARM='CENTWIN=1998'
```

Figure 316. Example CENTWIN Override

To allow for applications that call the sort several times (multiply-invoked sorts) for which some CENTWIN values require overrides and others do not, the CENTWIN=* can be inserted to denote a null override. The following example shows an EXEC PARM that was coded for an invoking program that calls the sort three times; the first and third sorts require CENTWIN overrides, and the second sort does not.

```
// EXEC INVOKED,SIZE=512K,PARM='CENTWIN=50,CENTWIN=*,CW=1985'
```

Figure 317. Example EXEC PARM

Note that the CENTWIN parameters may mix sliding and fixed century window values.

DIAG

```
DIAG
```

Figure 318. DIAG PARM Format

The DIAG PARM causes a listing of diagnostic messages and PDUMPs to be produced.

EQUALS/NOEQUALS

```
EQUALS
NOEQUALS
```

Figure 319. EQUALS/NOEQUALS PARMs Format

The EQUALS and NOEQUALS PARMs determine if the original order of records with equal control fields needs to be preserved. These PARMs use the same rules as the EQUALS and NOEQUALS parameters of the OPTION control statement. See "EQUALS/NOEQUALS Parameter (Optional)" on page 2.102.

ERASE/ERASEWK

```
ERASE
ERASEWK
```

Figure 320. ERASE/ERASEWK PARMs Format

ERASE or ERASEWK causes contents of the work files opened by the sort to be erased at the end of the sort. This PARM is equal to the ERASEWK parameter of the SORT control statement. See "ERASEWK Parameter (Optional)" on page 2.234.

IGNRL4/NOIGNRL4

```
IGNRL4
NOIGNRL4
```

Figure 321. IGNRL4/NOIGNRL4 PARMs Format

The IGNRL4 and NOIGNRL4 PARMs determine if the sort checks that each variable-length input record is at least as large as the specified or computed l_4 value. These PARMs use the same rules as the IGNRL4 and NOIGNRL4 parameters of the OPTION control statement. See "IGNRL4/NOIGNRL4/VLSHRT/NOVLSHRT Parameter (Optional)" on page 2.126.

The IGNRL4 overrides the INPFIL L4FILL and SYNCMAC L4FILL specifications for all files. No padding is done for any short record of length < l_4. The NOIGNRL4 PARM, allows a default L4FILL=X'nn' or C'c' specified by SYNCMAC used to pad short records.

INCOR=OFF

Figure 322. INCOR=OFF PARM Format

The INCOR=OFF PARM instructs SyncSort for z/VSE that an incore sort is NOT to be performed.

PRINT

Figure 323. PRINT PARM Format

The PRINT PARM controls which messages are displayed by SyncSort. This PARM uses the same rules as the PRINT parameter of the OPTION control statement. See “PRINT Parameter (Optional)” on page 2.131.

ROUTE

Figure 324. ROUTE PARM Format

The ROUTE PARM controls where messages are displayed by SyncSort. This PARM uses the same rules as the ROUTE parameter of the OPTION control statement. See “ROUTE Parameter (Optional)” on page 2.132.

SF

Figure 325. SF PARM Format

The SF PARM allows a SyncSort customer to override an invoking program's SORT FIELDS control statement by using the SF=(p, l, f, o) PARM on the EXEC statement. Only the (p, l, f, o) part of the invoking program's SORT control statement will be replaced. All other parameters on the invoking program's SORT control statement will remain unchanged. It should be noted that the syntax of (p, l, f, o) must follow the syntax required for the FIELDS parameter of the SORT control statement or collating errors or syntax errors will occur. The following is an example of the SORT FIELDS override for a program that invokes the sort only once.

```
// EXEC INVOKED,SIZE=512K,PARM='SF=(5,7,CH,A)'
```

Figure 326. Example SORT FIELDS Override

To allow for applications that call the sort several times (multiply-invoked sorts) for which some SORT FIELDS require overrides and others do not, SF=* can be inserted to denote a null override. The following example shows an EXEC PARM that was coded for an invoking program that calls the sort three times; the first and third sorts require SORT FIELDS overrides, and the second sort does not.

```
// EXEC INVOKED,SIZE=512K,PARM='SF=(5,7,CH,A),SF=*,SF=(9,4,BI,A)'
```

Figure 327. Example EXEC PARM for Invoking Program

In addition to the SF=(....) and SF=* override parameters, the SF=PRT parameter facilitates printing the sort control statements. The SF=PRT parameter allows the customer to print the invoking program's sort control statements prior to modifying them via the SF=(....) parameter.

The specification of the SF=PRT parameter will cause the sort to print the sort control statements depending on where ROUTE= is pointing. If ROUTE=LOG, the sort control statements will be printed on SYSLOG. If ROUTE=LST, the sort control statements will be printed on SYSLST. This only allows printing of the sort control statements to SYSLST as long as SYSLST is assigned to a printer device; it will not print the control statements to any other device. If the SF=PRT parameter is used with SyncSort's DIAG parameter, DIAG will be turned off. The SF=PRT parameter can be placed anywhere in the EXEC PARM and will not affect the assignment of the SF=(....) or SF=* override parameters. The following is

an example of the SF=PRT parameter to print the invoking program's sort control statements.

```
// EXEC INVOKED,SIZE=512K,PARM='SF=PRT'
```

Figure 328. Example SF=PRT Parameter

SYMNAMES

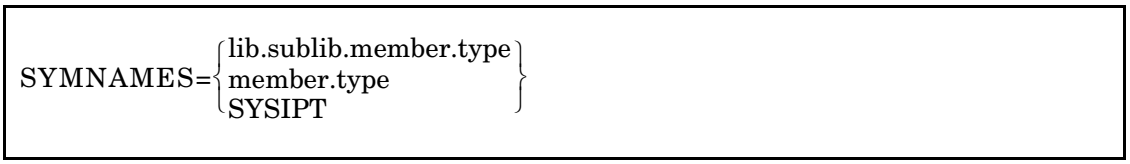


Figure 329. SYMNAMES PARM Format

The SYMNAMES PARM indicates that the SyncSort data dictionary feature is to be used to map symbolic names in control statements to proper position, length, and format values or constants. This PARM uses the same rules as the SYMNAMES parameter of the OPTION control statement. See "SYMNAMES Parameter (Optional)" on page 2.136.

SYMNLIB

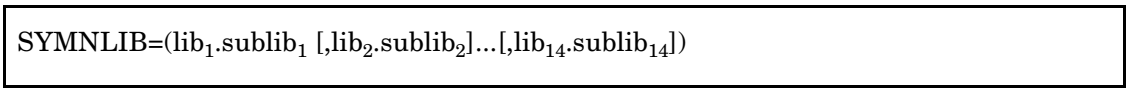


Figure 330. SYMNLIB PARM Format

The SYMNLIB PARM specifies a set of library.sublibrary pairs that constitutes a library search chain used in library searches when the SYMNAMES parameter specifies only a member name and member type. As many as 14 library/sublibrary pairs may be specified.

SYMNOUT

Figure 331. SYMNOUT PARM Format

The SYMNOUT PARM specifies where the SyncSort dictionary listing is to be printed. This PARM uses the same rules as the SYMNOUT parameter of the OPTION control statement. See "SYMNOUT Parameter (Optional)" on page 2.136.

VL5L6L7

VL5L6L7

Figure 332. VL5L6L7 PARM Format

The VL5L6L7 PARM is used to calculate and display values for a variable-length file that can be used to improve sort performance of future sorts using that file. VL5L6L7 will produce the WER224I message that displays recommended values of l_5, l_6, and l_7. These values can be specified as LENGTH parameters on the RECORD control statement for another sort using that file.

The l_5, l_6, and l_7 values for a particular sort application depend on the distribution of record lengths in the file, the position of the sort key(s), and any reformatting done by an E15 or by INREC. The displayed l_5 is the most common record length of the records in the file after INPFIL INREC, E15, and INREC processing; l_6 is the average work space per record; and l_7 is the recommended segment size.

Since additional overhead is used to calculate these values when VL5L6L7 is specified, resulting in reduced performance when it is used, this parameter should be used only once for a particular sort application. The first time the sort is run, use VL5L6L7 to calculate l_5, l_6, and l_7 values for that file with a specific sort key position and, if needed, an E15 and/or INREC. For subsequent sorts using that data (either unchanged or with a relatively small number of updates), remove the VL5L6L7 PARM and add l_5, l_6, and l_7 values to the LENGTH parameter of the RECORD control statement to improve the performance of those sorts.

When the INREC control statement is used to reformat the input records, care should be used in specifying the value of l_5. The value of l_5 displayed by the WER224I message is the most common record length of the records as they are being sorted, i.e. after all INPFIL INREC, E15, and INREC processing has been done. However, the value of l_5, which should be specified on the RECORD control statement, is the most common length after only INPFIL INREC and E15 processing, but before processing by the INREC control statement; any record length adjustments to l_5 (and l_2 and l_4) due to the INREC control statement are performed internally by SyncSort. (See Figure 3 on page 2.18.) For example, if the INREC control statement expands the records by 50 bytes, and WER224I displays l_5=76, then the actual l_5 value to be specified on a subsequent RECORD control statement is 26.

Another way to determine l_5, l_6, and l_7 values for a file, without sorting it, is to use the separate SYNCHSTO utility program (see “Chapter 11. The SYNCHSTO Utility Program”). SYNCHSTO is useful when a sort is not going to reformat the records with an E15 or INREC. It also allows you to sample a limited number of records of a very large file to estimate these values.

This PARM does not have a corresponding installation default or OPTION control statement parameter to override. It is ignored for a MERGE or a COPY application.

VSCORE

Figure 333. VSCORE PARM Format

The VSCORE PARM establishes the upper bound of virtual storage below the 16-megabyte line in which SyncSort will run, specified as a decimal number of n, nK (K=1024), or nM (M=1024x1024) bytes.

VSCORET

Figure 334. VSCORET PARM Format

The VSCORET PARM sets the maximum amount of storage that SyncSort can use when it is running in a partition that has storage above the 16-megabyte line, specified as a decimal number of n, nK (K=1024), or nM (M=1024x1024) bytes. If the specified value of VSCORET is less than the specified or default value of VSCORE, SyncSort automatically raises the value of VSCORET to the VSCORE value.

WORKNM

Figure 335. WORKNM PARM Format

The WORKNM PARM changes the four-character prefix of all SORTWK names from the default, SORT. For example, if WORKNM=SYNC is specified, all sort work files will be named SYNCWKx instead of SORTWKx.

Chapter 7. Invoking SyncSort from a Program

Another way to initiate the sort/merge is to have it loaded by an executing program. This program may be written in a number of languages including Assembler, COBOL, and PL/1. The coding necessary to invoke the sort/merge from an Assembler and from a COBOL program is given in this chapter.

When the sort/merge is invoked from a program, you can direct messages to a specific logical unit. This feature is explained in "OPTION Control Statement" on page 2.114 and "ROUTE Parameter (Optional)" on page 2.132.

Invoking SyncSort from an Assembler Program

When an Assembler program is used and SyncSort is to be called somewhere within that program, use the LOAD or CALL macro to establish linkage. (SyncSort will not execute as a subtask, so the FETCH and ATTACH macros cannot be used.) If the LOAD or CALL macro is used, SyncSort should be loaded on a doubleword boundary at the end of the program. To insure proper linkage, registers 13, 14, 15, and 1 must contain the following information:

Register 13 Must contain the address of an 18-word save area where SyncSort can store the contents of the user's registers when the sort/merge gains control. When the sort/merge program is over, these registers will be automatically restored to the user.

Register 14 Must contain the address of the instruction in the invoking program to which SyncSort is to return control when it finishes the sort/merge.

Register 15 Must contain the address of SyncSort's entry point.

Register 1 Must contain the address of a variable-length parameter list of address constants.

Addressing and Residence Modes For Invoking Programs

SyncSort may be invoked from an Assembler program located above or below 16 MB of virtual storage and may execute in either 24-bit or 31-bit addressing mode.

SyncSort will process the invoking parameter list in the addressing mode that is current when it receives control, and it will return control to the invoking program in that same mode.

- If SyncSort receives control in 24-bit mode, it treats all addresses as 24-bit addresses. SyncSort will ignore the high order byte of these addresses, and all user exits preloaded by the invoking program must be located below 16 MB of virtual storage and must be able to execute in AMODE 24.

- If SyncSort receives control in 31-bit mode, it treats all addresses as 31-bit addresses, and these addresses may be located either above or below 16 MB of virtual storage. If the addresses reside below 16 MB of virtual storage, the invoking program must provide "clean" 24-bit addresses, that is, bits 1 through 7 must be 0. SyncSort ignores the high order bit (bit 0) unless it is part of the address of a user exit.

When user exit addresses are included in the parameter list, bit 0 of the address signals the addressing mode of the exit. When the bit is set to 0, SyncSort calls the user exit in AMODE 24. When the bit is set to 1, SyncSort calls the user exit in AMODE 31.

When it loads a user exit, SyncSort always calls the exit in the AMODE assigned by the linkage editor, and provides "clean" 24-bit addresses in the parameter list. All addresses returned by the user exit must be valid 31-bit addresses for AMODE 31 exits and valid 24-bit addresses for AMODE 24 exits.

A user exit may return to SyncSort in any addressing mode.

The Parameter List of Address Constants

An Assembler program that invokes the sort/merge must include a 10-word or an 18-word parameter address constant list. These address constants will point to the images of SyncSort control statements, to a halfword set aside for a return code, and to three-branch tables for exit programs.

The first 10 parameters in the parameter list are obligatory and must be coded in the correct sequence, with A(0) coded for any control statement image or branch table entry not used. The final 8 parameters (address constants for ALTSEQ through REFORMAT) are

optional. The sequence of these last 8 addresses is not important and unused parameters can be omitted.

```
LINKLIST DC  A(LSORT)     * ADDRESS OF SORT CONTROL STATEMENT IMAGE
         DC  A(LREC)      * ADDRESS OF RECORD CONTROL STATEMENT IMAGE
         DC  A(LINPFIL)   * ADDRESS OF INPFIL CONTROL STATEMENT IMAGE
         DC  A(LOUTFIL)   * ADDRESS OF OUTFIL CONTROL STATEMENT IMAGE
         DC  A(LOPTION)   * ADDRESS OF OPTION CONTROL STATEMENT IMAGE
         DC  A(LMODS)     * ADDRESS OF MODS CONTROL STATEMENT IMAGE
         DC  A(PHASE1)    * ADDRESS OF PHASE 1 BRANCH TABLE
         DC  A(PHASE2)    * ADDRESS OF PHASE 2 BRANCH TABLE
         DC  A(PHASE3)    * ADDRESS OF PHASE 3 BRANCH TABLE
         DC  A(RETCODE)   * ADDRESS OF HALFWORD FOR RETURN CODE
         DC  A(LALTSEQ)   * ADDRESS OF ALTSEQ CONTROL STATEMENT IMAGE
         DC  A(LOUTREC)   * ADDRESS OF OUTREC CONTROL STATEMENT IMAGE
         DC  A(LSUM)      * ADDRESS OF SUM CONTROL STATEMENT IMAGE
         DC  A(LINCOM)    * ADDRESS OF INCLUDE/OMIT CONTROL STATEMENT IMAGE
         DC  A(LANALYZE)  * ADDRESS OF ANALYZE CONTROL STATEMENT IMAGE
         DC  A(LINREC)    * ADDRESS OF INREC CONTROL STATEMENT IMAGE
         DC  A(LJOIN)     * ADDRESS OF JOIN CONTROL STATEMENT IMAGE
         DC  A(LREFORMAT) * ADDRESS OF REFORMAT CONTROL STATEMENT IMAGE
```

Figure 336. Sample Parameter List of Address Constants

The Control Statement Images

The actual control statement images must use DC instructions and follow these specific rules:

1. Each control statement image must be delimited at the beginning by the constant C followed immediately by a single quote.

2. The operator (SORT, RECORD, etc.) must immediately follow the single quote.

3. The operator must be followed by exactly one blank.

4. The operator and parameters of the control statement images must be coded exactly as the control statements described in “Chapter 2. SyncSort Control Statements”.

5. The last parameter is followed by exactly one blank and then a final delimiting single quote.

6. The parameter for the INPFIL control statement image addresses may specify more than one control statement image to accommodate multiple input files. If more than one INPFIL control statement image is to be specified, each subsequent specification should immediately follow the final blank of the prior specification. A parameter list containing multiple INPFIL control statement images is illustrated in Figure 339 on page 7.7.

7. The parameter for the OUTFIL/XSUMFIL/XDUPFIL control statement image addresses may specify more than one control statement image to accommodate multiple output files. If more than one OUTFIL control statement image is to be specified, each subsequent specification should immediately follow the final blank of the prior specification. A parameter list containing multiple OUTFIL/XSUMFIL/XDUPFIL control statement images is illustrated in Figure 339 on page 7.7.

8. If both the INREC and the OUTREC control statement images are specified, each control statement image address may be specified separately. Or, the parameter for either the INREC control statement image address or the OUTREC control statement image address may be used to specify both control statement images. In this case, the second control statement image should immediately follow the final blank of the first specification.

9. The parameter for the SUM control statement image may be used to specify the DUPKEYS control statement image.

10. In a JOIN application, the parameter for the INPFIL control statement image would be used to specify the JOINKEYS control statement image.

11. To continue a control image, place a single quote with no embedded blank after the last character on the first card image. The successive card image should code a DC instruction in columns 10 and 11, a C' in columns 16 and 17, start the continuation in column 18, and leave exactly one blank followed by a single quote after the last parameter.

12. Embedded blanks are not allowed on control statement images, except within literal strings.

13. Comments are not allowed on the control statement images.

An example of a coded parameter list using the addresses from the sample parameter list of address constants is given in Figure 337 on page 7.5.

The Return Code Parameter

The return code address constant points to a halfword that must be set up somewhere in the user's program. After a successful run of the sort/merge program, SyncSort will store a 0 in this halfword. If the sort/merge was not successful, SyncSort will place a 16 in this halfword.

An example of a coded return code parameter is given in Figure 337 on page 7.5.

```
LSORT    DC  C'SORT FIELDS=(4.1,8.0,BI,D),WORK=2 '
LREC     DC  C'RECORD TYPE=V,LENGTH=(16,,,56,98),'
         DC  C'DELBLANK=(42,$) '
LINPFIL  DC  C'INPFIL BLKSIZE=3500 '
LOUTFIL  DC  C'OUTFIL BLKSIZE=6000 '
LOPTION  DC  C'OPTION PRINT=ALL,DUMP,STORAGE=120K '
LMODS    DC  C'MODS PH1=(,,E15) '
* PHASE 1 BRANCH TABLE
           USING PHASE 1,15
PHASE1   DC  A(0)
EXIT15   B   E15
EXIT17   DC  A(0)
EXIT18   DC  A(0)
         .
         .
         .
* END CONTROL STATEMENT IMAGES
E15      DS  0H
* EXIT 15 ROUTINE FOLLOWS HERE
         .
         .
         .
RETCODE  DS  H'0' *HALFWORD FOR RETURN CODE
```

Figure 337. Sample Code Parameter List Including Some Control Images and One Branch Table

```
LALT     DC  C'AQTT'
         DC  A(TTAB)
TTAB     DC  X'00010203FF09FF7FFFFFFF0B0C0D0E0F '
             X'10111213FFFF08FF18191A1B1C1D1E1F '
                          .
                          .
                          .
             X'30313233343536373839FFFFFFFFFFFF '
```

Figure 338. Coding an AQTT Table to Translate EBCDIC to ASCII

The ALTSEQ Control Statement Alternative

Although the code parameter on the ALTSEQ control statement image can be used to pass information for SyncSort to build a 256-byte translate table, you can build your own 256-byte table when invoking the sort/merge from a program. In this case, include your table in the program and SyncSort will reference it for all control fields specified with AQ as the data format code.

To create a translate table, follow these steps:

1. In place of the ALTSEQ address constant, specify the address of a fullword aligned location containing C'AQTT' in the parameter address list.

2. Program a constant C with the value 'AQTT'.

3. Immediately following this entry, place the address of the table.

4. Construct a 256-byte hexadecimal translate table.

An example of the coding necessary for the AQTT table is given in Figure 338 on page 7.5. Note that the entire table is not given.

A Sample Assembler Program Invoking SyncSort

Figure 339 on page 7.7 illustrates how to invoke the sort/merge from an Assembler Program. When SyncSort is invoked from a program, it will return control to the user's program upon completion, giving a code to indicate either a successful or an unsuccessful sort/merge program. For this reason, the sample program in Figure 339 on page 7.7 includes a process for a successful sort as well as a process for an unsuccessful sort. It checks the sort's return code with a compare instruction and either branches to or falls into the routine to process the results of the sort.

Invoking SyncSort from a COBOL Program

The COBOL programming language includes a sort feature so that the COBOL programmer has complete opportunity to access the SyncSort program. The COBOL program must include an SD (Sort-File-Description) in the File Section and a SORT statement in the Procedure Division. Control fields, called sort-keys in COBOL, are defined in the SD and, as specified in the SORT verb, sort-keys are either sorted in ascending or descending order according to the EBCDIC collating sequence for COBOL characters. (Consult American National Standard COBOL texts for complete details.)

A Sample COBOL Program Invoking SyncSort

The program in Figure 342 on page 7.9 was designed to produce three output reports from a single pass of the master file. By using the SyncSort sort/merge program together with the COBOL sort feature, three distinct records are created from one master record, sorted and later written out on the appropriate reports.

```
         *LOAD SORT,ESTABLISH LINKAGE,BRANCH AND LINK

         LOAD  SORT,SENTRY     *LOAD SORT AT SENTRY
         LR    R15,R1          *LOAD ADDRESS OF SENTRY
         LA    R1,PADLIST      *LOAD ADDRESS OF PARAMETER ADDRESS LIST
         LA    R13,REGSAVE     *LOAD ADDRESS OF REGISTER SAVE AREA
         BALR  R14,R15         *LINK TO R15 WITH A RETURN TO R14

         *SORT RETURNS CONTROL AT COMBACK, RETURN CODE IS CHECKED

COMBACK  CLC   RETCODE(2),=H'0'*COMPARE FOR SUCCESSFUL SORT
         BNE   BADSORT         *NO GOOD, GO TO ERROR ROUTINE
PROCESS  DS    0H
         .
         .                     *PROCESS ROUTINE FOR SUCCESSFUL SORT
         .
PADLIST  DC    A(SORT1)        *SORT CONTROL IMAGE ADDRESS
         DC    A(REC1)         *RECORD CONTROL IMAGE ADDRESS
         DC    A(INPUT1)       *INPFIL CONTROL IMAGE ADDRESS
         DC    A(OUTPT1)       *OUTFIL CONTROL IMAGES ADDRESS
         DC    A(OPT1)         *OPTION CONTROL IMAGE ADDRESS
         DC    A(0)            *MODS CONTROL IMAGE (UNUSED) ADDRESS
         DC    A(0)            *PHASE 1 BRANCH TABLE (UNUSED) ADDRESS
         DC    A(0)            *PHASE 2 BRANCH TABLE (UNUSED) ADDRESS
         DC    A(0)            *PHASE 3 BRANCH TABLE (UNUSED) ADDRESS
         DC    A(RETCODE)      *RETURN CODE HALFWORD ADDRESS
         DC    A(0)            *ALTSEQ CONTROL IMAGE (UNUSED) ADDRESS
         DC    A(INREC1)       *INREC CONTROL IMAGE ADDRESS
         DC    A(SUM1)         *SUM CONTROL IMAGE ADDRESS
         DC    A(OMIT1)        *OMIT CONTROL IMAGE ADDRESS
         DC    A(OUTREC1)      *OUTREC CONTROL IMAGE ADDRESS

*THE PARAMETER ADDRESS LIST IS GIVEN ABOVE, THE PARAMETERS ARE REFERENCED
*BELOW

SORT1    DC    C'SORT FIELDS=(24,8,BI,A),WORK=3,FILESOUT=2,FILES=2 '
REC1     DC    C'RECORD TYPE=F,LENGTH=148 '
INPUT1   DC    C'INPFIL BLKSIZE=1480,FILES=1 '
         DC    C'INPFIL BLKSIZE=2960,FILES=2 '
OUTPT1   DC    C'OUTFIL BLKSIZE=2500,FILES=1 '
         DC    C'OUTFIL BLKSIZE=5000,FILES=2 '
         DC    C'XSUMFIL BLKSIZE=2200 '
OPT1     DC    C'OPTION STORAGE=100K '
INREC1   DC    C'INREC FIELDS=(1,100,139,10) '
SUM1     DC    C'SUM FIELDS=(96,4,PD),XSUM '
OMIT1    DC    C'OMIT COND=(71,5,ZD,GT,5000) '
OUTREC1  DC    C'OUTREC FIELDS=(1,8,5X,9,10,5X,19,4,5X,23,88) '

REGSAVE  DC    18F'0'          *REGISTER SAVE AREA OF 18 FULLWORDS
RETCODE  DC    H'0'            *HALFWORD FOR RETURN CODE
         LTORG
BADSORT  DS    0H
         .
         .                     *ROUTINE FOR UNSUCCESSFUL SORT
         .
SENTRY   DC    D'0'            *SYNCSORT ENTRY POINT
         END
```

Figure 339. Assembler Program Invoking SyncSort

Problem Description

A master payroll file is in order according to social security number. Each record would appear as in Figure 340.

1	10	15	19	25 45
SOCIAL SECURITY NUMBER #	EMPLOYEE #	DEPT #	SALARY	PAYROLL DATA

Figure 340. A Record from a Master Payroll File

It is necessary to produce three reports containing these records, each with the records in order according to a different field.

For the first report, the records are to be in order according to employee number.

For the second report, the records are to be in order according to employee number within department number.

For the third report, the records are to be in order according to salary, but only including those records with salaries over $10,000.

A 14-byte key is generated to become the new sort field. This is given in Figure 341.

1	5	9 14
DEPT #	EMPLOYEE #	SALARY

Figure 341. 14-Byte Key As New Sort Field

The key will be appended to the beginning of each record generated from the master file.

After the fields in the appended keys are properly filled in and the records are sorted, the fields in the keys will be tested so that the record may be written on the appropriate report.

The appended key will not be picked up, so only the original record will appear in the reports.

```
IDENTIFICATION DIVISION
PROGRAM-ID.    REPGEN
AUTHOR.    SYNCSORT INCORPORATED
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE COMPUTER.    IBM-370-158.
OBJECT COMPUTER.    IBM-370-158.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT EMP-MASTER ASSIGN TO SYS007-UR-2540R-S.
    SELECT REP-MASTER ASSIGN TO SYS009-UT-2400-S-SORTOUT.
    SELECT SORT-FILE ASSIGN TO SYS001-UT-3330-S-SORTWK1.
DATA DIVISION.
FILE SECTION.
SD  SORT-FILE
    SORT-OPTION IS WORK-OPTION
    DATA RECORD IS REP-REC.
01  REP-REC.
    05   GEN-KEY       PIC X(14).
    05   REC           PIC X(45).
FD  EMP-MASTER
    DATA RECORD IS EMPLOYEE.
01  EMPLOYEE.
    05   $$            PIC X(9).
    05   EMP           PIC 9999.
    05   DEPT          PIC 9999.
    05   MONEY         PIC 9(6).
         88  REC-OK    VALUE 10001 THRU 999999.
    05   NAME          PIC X(20).
    05   FILLER        PIC X(37).

FD  REP-FILE
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS REPORT-REC.
01  REPORT-REC.
    05   REP-CODE      PIC X(14).
    05   REP-RECD      PIC X(45).

WORKING STORAGE SECTION
01  WORK-OPTION        PIC X(20)  VALUE   'OPTION SORTWK=003'.
01  WORK-RECORD
    05   WORK-KEY      PIC X(14) VALUE SPACE.
    05   WORK-REC      PIC X(45) VALUE SPACE

01  SORT-KEY.
    05   S-DEPT        PIC 9(4)  VALUE ZERO.
    05   S-EMP         PIC 9(4)  VALUE ZERO.
    05   S-SAL         PIC 9(6)  VALUE ZERO.
```

Figure 342. Invoking SyncSort from a COBOL Program

```
PROCEDURE DIVISION.
INITIAL.
    OPEN INPUT EMP-MASTER.
START-SORT.
    SORT SORT-FILE
         ASCENDING GEN-KEY
         INPUT PROCEDURE BUILD
         GIVING REP-FILE.
REPORT-WRITE.
    OPEN INPUT REP-FILE.
R-W.
    READ REP-FILE AT END GO TO EOJ-STEP.
    GO TO R-W.
EOJ-STEP.
    CLOSE REP-FILE.
EOJ-EXIT.
    STOP RUN.
BUILD SECTION.
BUILD-KEY.
    READ EMP-MASTER AT END GO TO CLOSE-MST.
    MOVE ZEROS TO SORT-KEY.
    MOVE DEPT TO S-DEPT.
    MOVE SORT-KEY TO WORK-KEY.
    MOVE EMPLOYEE TO WORK-REC.
    RELEASE REP-REC FROM WORK-RECORD.
    MOVE ZEROS TO SORT-KEY.
    MOVE EMP TO S-EMP.
    MOVE SORT-KEY TO WORK-KEY.
    MOVE EMPLOYEE TO WORK-REC.
    RELEASE REP-REC FROM WORK-RECORD.
    IF REC-OK
          MOVE ZEROS TO SORT-KEY,
          MOVE MONEY TO S-SAL,
          MOVE SORT-KEY TO WORK-KEY,
          MOVE EMPLOYEE TO WORK-REC,
          RELEASE REP-REC FROM WORK-RECORD.
    GO TO BUILD-KEY.
CLOSE-MST.
    CLOSE EMP-MASTER.
MST-EXIT.
    EXIT.
```

Figure 343. Invoking SyncSort from a COBOL Program (cont.)

Chapter 8. User Exit Programs

What Is an Exit?

User-supplied routines are known as exits. Different types of exits can be written to do a variety of tasks at different stages of the sort/merge. The point in the sort/merge at which an exit program takes control to do a specific task is also known as an exit, and this point has the same exit number as the program that is entered to do the task.

The number of an exit provides certain information: the first digit in the exit number indicates whether the exit occurs in Phase 1, 2, or 3 of the sort/merge. The second digit refers to the type of task(s) the exit can perform.

Table 47 on page 8.2 lists all available exits. It also illustrates in which phase each exit occurs and provides an overall view of the tasks that can be performed by each exit.

Loading an Exit Program into Main Storage

There are two ways to get exit programs into main storage: (1) place them in the core image library and have the sort/merge load them; (2) place them in main storage and tell the sort/merge where they are to be found.

When SyncSort Loads a User-Supplied Program

Group all exit routines that occur in a particular phase of the sort/merge together and catalogue each group under a unique name in the core image library. The group name(s), loading information, and names of the exits being included are supplied on the MODS control

statement. This informs SyncSort when to get the exit programs, where to find them, and how much space to set aside.

SyncSort can load the exits (routines) whether the sort is initiated by JCL or by another program.

TASKS	PHASE 1				PHASE 2			PHASE 3					
	E11	E15	E17	E18	E21	E25	E27	E31	E32	E35	E37	E38	E39
Checkpoint								*					
Process labels	*	*	*					*			*		
Open files	*							*					
Close files	*	*	*					*			*		
Read sort input		*											
Count input records		*											
Insert records		*								*			
Delete records		*				*				*			
Lengthen and shorten records		*								*			
Alter noncontrol fields of records		*				*			*	*			
Read merge input									*				
Sum records		*				*				*			
Substitute records (merge only)									*				
Suppress sequence checking										*			
Write output										*			
Process read errors				*								*	
Process direct access write errors													*
Provide VSAM passwords				*								*	*

Table 47. Exit Task Chart

Note: E21 and E27 are not currently supported.

If the exit programs are self-relocating or can be relocated by the system loader program, it is advisable to supply the length of the programs in the MODS statement rather than give an absolute address. If the exit programs cannot be relocated, they should be placed in the highest address available within the sort/merge partition.

When Programs Are Preloaded

If the sort/merge program is initiated from another program, exit routines may be placed in storage by the user. The initiating program must then contain the address constants of three exit branch tables, one for each of the three exit phases. These are part of a fixed-length parameter list that is to be placed in register 1. (See "Chapter 7. Invoking SyncSort from a Program".) Exit routines may not be preloaded by the user if the sort/merge is initiated by JCL.

High Level Language User Exit

In addition to assembler language, user exits may be coded in one of the supported High Level Languages (HLL). SyncSort supports user exits coded in COBOL/VSE, C/VSE, and REXX/VSE.

Note: COBOL/VSE and C/VSE exits require activation of the SyncSort LOCALE and COBOL/C Exit Facility as described in the *SyncSort for z/VSE Installation Guide*.

HLL exits cannot be pre-loaded by a user program. They must be loaded by SyncSort.

COBOL/VSE exits and C/VSE exits must be compiled and link-edited into phases and placed in core image libraries in the active search chain.

REXX/VSE exits must be catalogued in a library that is in the active search chain for PROC. SyncSort does not support compiled REXX exits.

Use the MODS control statement to indicate an HLL exit. In the loading information field, code COBOL for COBOL/VSE exit, C for C/VSE exit, and EXEC or REXX for REXX/VSE exit. SyncSort will use the appropriate method to load the HLL exit.

Linking Exit Programs to the Sort/Merge

Two procedures must be followed to secure linkage between exit programs and the sort/merge. First, for every phase in which exits are issued, the sort/merge must be given a branch table with the exits to be used within that particular phase. Second, exit programs must make use of the parameters that the sort/merge sets up for them. The information passed by SyncSort is placed in registers 1, 13, 14, and 15, and is accessed by standard programming conventions.

The Branch Table

All exit routines for a particular phase must be placed together in ascending numerical sequence and loaded as a single module. Immediately preceding this module must be a branch table that lists every branch (including those not supported) in the phase in ascending numerical sequence.

The format for each of the three branch tables is given in Figure 344, Figure 345 and Figure 346, respectively.

```
                USING PHASE1,15
   PHASE1       B     E11
                B     E15
                B     E17
                B     E18
```

Figure 344. PHASE1 Branch Table Format

```
           USING PHASE2,15
PHASE2     DC    A(0)       *E21 NOT SUPPORTED
           B     E25
           DC    A(0)       *E27 NOT SUPPORTED
```

Figure 345. PHASE2 Branch Table Format

```
           USING PHASE3,15
PHASE3     B     E31
           B     E32
           B     E35
           B     E37
           B     E38
           B     E39
```

Figure 346. PHASE3 Branch Table Format

Code a 4-byte branch instruction for every exit used, and code 4 bytes of zeros for every exit not used. An example of a coded Phase 3 branch table is Figure 347.

```
PHASE3   B     E31      *BRANCH TO E31
         DC    A(0)     *NO BRANCH TO E32
         DC    A(0)     *NO BRANCH TO E35
         B     E37      *BRANCH TO E37
         B     E38      *BRANCH TO E38
         DC    A(0)     *NO BRANCH TO E39
```

Figure 347. Sample PHASE3 Branch Table

The Parameters the Sort/Merge Sets Up

The sort/merge places addresses in registers 1, 13, 14, and 15 so that exit routines can access the information they need when they take control. The registers contain addresses of the following information.

Register 1 Contains the address of a parameter list of address constants. The parameter list format will vary, depending on which exit is being used. (The parameter list format for each exit is given later in the discussion for the exits.)

Register 13 Contains the address of a save area of eighteen words. Store the contents of general registers 14 through 12 at this address, starting at the fourth fullword. In other words, the save area begins 12 bytes off of register 13.

At the end of the routine, remember to restore the contents of registers 14 through 12 before returning control to SyncSort.

Register 14 Contains SyncSort's return address. At the end of the exit program, return to SyncSort at this address.

Register 15 Contains the address of the first instruction of the branch table for whatever phase is being executed.

Addressing and Residence Modes of User Exits

SyncSort supports both AMODE 24 and AMODE 31 user exits, and user exits loaded above or below 16 MB of virtual storage.

When it loads a user exit, SyncSort always calls the exit in the AMODE assigned by the linkage editor to the exit phase. When the address of a preloaded user exit is passed to SyncSort by an invoking parameter list, SyncSort determines the entry AMODE of the exit according to the following rules.

- If SyncSort is invoked with AMODE 24, SyncSort ignores the high order byte of the address of the user exit and calls the exit with AMODE 24.

- If SyncSort is invoked with AMODE 31, SyncSort uses the high order bit of the user exit address to determine the AMODE of the exit. If the high order bit is set to 1, the exit will be called with AMODE 31. If the high order bit is set to 0, the exit will be called with AMODE 24.

An exit entered with 24-bit addressing mode can only pass 24-bit addresses back to SyncSort. An exit entered with 31-bit addressing mode can only pass 31-bit addresses back to SyncSort, and bits 1 through 7 must be set to 0 in addresses below 16 MB of virtual storage.

A user exit may return to SyncSort in any addressing mode.

EXITS E11 and E31 Checkpointing and Label Processing

These exits are used for opening and closing files, initialization or termination processing, and the processing of labels. E31 is additionally used for checkpointing. Since the only time the opening and closing of files would be done by the user is while assuming the responsibility for processing labels, the opening and closing of files will be treated in this section, along with label processing.

Note: Label processing must be done whenever the user's files have nonstandard or user standard labels. The LABEL parameter of the OPTION control statement must be specified whenever label processing is done.

Exits E15, E32, and E35 may also be used to process labels and should do so if EXIT has been specified in the INPFIL or OUTFIL control statements. In addition, *final* trailer labels (those on the last input and output volumes) should be processed by E17 and E37, respectively.

Parameter and Entry Lists

For each of these exits, SyncSort will place the address of a parameter list of full word addresses in register 1. The information passed in these parameter lists will vary according to the point the user has reached in the sort/merge. An exit is approached many times during the SyncSort program and there is an opportunity to enter at each of these times; these opportunities are called *entries* and do various things. Table 48 and Table 49 on page 8.7 show the parameter to which each word points and the entry lists for each of the exits, respectively, showing what is done at the various entry points of each exit.

POINTER TO	USE
Checkpoint action word Checkpoint device list	checkpoint (E31 only)
Previous volume unit (symbolic unit number) Next volume unit (symbolic unit number) Block count (of trailer tape) SYSnumber table	label processing

Table 48. Parameter List for E11 and E31

	E11 ENTRY LIST	E31 ENTRY LIST
ENTRY 1	Open volume 1 of input file 1 Process header labels for input file 1	Open volume 1 of all input files (merge only) Open volume 1 of output file (sort or merge) Process header labels for input and output files above Checkpoint work files (sort
LATER ENTRIES	Open additional input volumes or files Process header labels for these files Process trailer labels for the preceding input volume or file	Open additional merge input and sort output volumes Process their header labels Process trailer labels for preceding input or output volume

Table 49. Entry Lists for E11 and E31

Checkpointing

When the CKPT option is specified on the SORT statement, SyncSort will take checkpoints of sort data in main storage and on work devices at the start of the final merge pass. However, if checkpoints of data are to be taken from a user-supplied program, this should be done at Phase 3 using E31. If CKPT is also a specified option, SyncSort will not take checkpoints.

E11 Programs

Since sorting has not yet begun at this phase, it is not necessary to pass any information from the exit routine to the sort through parameters.

E31 Programs

At this exit it is possible to checkpoint both user devices and sort work files. Since only one merge occurs in Phase 3, only one checkpoint is taken. This occurs at Entry 1 when the exit routine first gets control.

The only parameter that applies to an E31 checkpoint program is the *Checkpoint device list*, which is the second parameter in the list. The device list for disk files is given below in Figure 348.

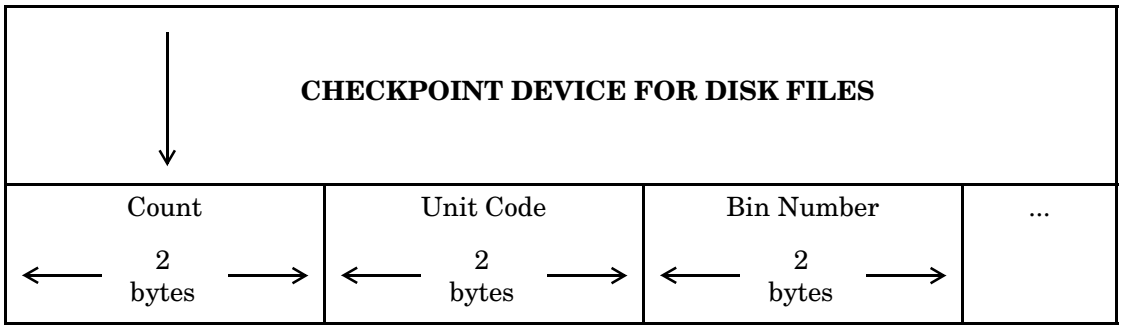


Figure 348. Checkpoint Device List for Disk Files

The 2-byte **Count** area contains the total number of devices to be checkpointed. The **Unit code** is the numeric portion of the symbolic unit name of the device, and the **Bin number** is always 0. (All numbers will be in the low-order bytes.) There will be only one count area at the start of the checkpoint device list, but a unit code and bin number will be given for every device checkpointed.

Label Processing

Nonstandard and user standard labels are not processed by the sort/merge program. Disk and tape files with these types of labels will need to have label processing routines written for headers, and the tape files will need routines for trailers as well. Since sort file labels are not processed in Phase 2, only E11 and E31 routines will be used, and if it is necessary to write a header label on the output file, exit 31 must be used. (Exit 37 must be used to write the trailer label on the last volume of the output file.) For exits E11 and E31, the last four words of the parameter list are relevant. (They are: 3. **Previous volume** (symbolic) **unit** (number), 4. **Next volume** (symbolic) **unit** (number), 5. **Block count** (of trailer tape), and 6. **SYSnumber table**.) The **Previous volume unit** points to the numeric portion of the symbolic unit name of the volume just processed. This is contained in a halfword in the low-order bytes. The **Next volume unit** points to the numeric portion of the symbolic unit name of the volume to be processed next. This is also in a halfword in the low-order bytes.

The **Block count** points to a 4-byte area with the number of blocks of the tape receiving a trailer label contained in the lower-order byte(s).

The **SYSnumber table** points to a 20 to 74 byte area containing the number portion of the symbolic unit name of every file used for the sort. Each byte contains the SYSnumber (in binary) of one file. The following describes the contents of each byte of the **SYSnumber table** for "n" output files (FILESOUT=n):

- Bytes 1 to "n" contain the SYSnumbers for the output files. Byte 1 is always present for the first output file. There are only as many bytes in the **SYSnumber table** as there are output files present.

- Bytes "n+1" through "n+m" contain the SYSnumbers for each of m input files. The maximum value for m is 32. A minimum of 9 bytes for input files is always reserved in the **SYSnumber table**, regardless of the number of input files actually present. If less than 9 input files are used, the unused bytes are filled with zeros.

- Bytes "n+m+1" through "n+m+9" contain the SYSnumbers for work files. The value of m may not be less than 9, and a minimum of 9 bytes is always reserved for work files in the **SYSnumber table**, regardless of the number of work files actually present. If less than 9 work files are used, the unused bytes are filled with zeros.

- Byte "n+m+10" contains the SYSnumber for the checkpoint file.

E11 Programs

For Entry 1, the **Previous volume unit** and the **Next volume unit** parameters are both filled with zeros. This permits testing for Entry 1. After testing successfully for Entry 1, open the first volume of the input file, referencing the **SYSnumber table** in parameter 6, and then process its label. (It is assumed that the first input volume is a nonstandard or user labeled file. For this first entry, it is important to know the type of label of the first volume of input.)

Later Entries of this exit occur each time another input volume is needed, and when a volume has user or nonstandard labels. In order to tell the user when a volume has standard or no labels and, therefore, needs no processing, parameter 4 will be set to zero and parameter 3 will contain the trailer unit. That means that if the **Previous volume unit** parameter contains a nonzero number, the volume just read needs to be processed; therefore, process its trailer label using the block count number given in parameter 5.

To find out if the last volume of a file has nonstandard or user labels, compare parameters 3 and 4. If they contain different numbers and parameter 3 is not zero, it is necessary to close the last volume. Then, check the **Next volume unit** (parameter 4); if it does not contain zeros, open the next input volume and process its header label.

The very last trailer label - the one on the last volume of the last input file - must be processed by an E17 program.

E31 Programs

For Entry 1, parameters 3 and 4 are set to zero. Use this entry to open the first volume of every input file used for a merge, referencing the SYSnumber table in parameter 6. (It is assumed these have nonstandard or user labels.) Also, use this entry to open the first volume of the output file for either a sort or a merge. Then, process their header labels.

Later Entries of this exit occur whenever another merge input or sort or merge output volume is needed, and when any of these volumes have nonstandard or user labels. If a volume has standard or no labels, parameters 3 and 4 will be set to zero. To find out if the volume just read needs processing, test parameter 3 (**Previous volume unit**). If it contains a numeric value, process a trailer label for this volume using the block count number found in parameter 5.

To find out if the last volume of a file has nonstandard or user labels, compare parameters 3 and 4. If they are different values and parameter 3 is not zero, it is necessary to close the volume.

To find out if it is necessary to open the next volume, test parameter 4. If it is not zero, open the volume and process its header label.

The last volume of every merge input file and the last volume of the output file for both a sort and merge cannot be processed at this exit. They must be processed with an E37 routine.

EXITS E15, E25, and E35 - Changing Records and Files

These exits are used when there is a desire to change the data in individual records, insert or delete records from a file, have the program assume responsibility for reading the input file, or have the program assume responsibility for writing the output file.

E15 Programs

Exit 15 is entered before input records are processed in Phase 1. It is, therefore, possible to write a program that changes the length of the records, or inserts or deletes records before any sorting takes place. The new length of the records must agree with whatever value has been specified for l_2 in the RECORD control statement. (For variable-length records, the maximum new length will agree with the l_2 value.)

If the E15 program is going to read the input file, it must also open the file, process the header labels, read the file and pass one record at a time to the sort/merge, close the file, and process tape trailer labels. And for all read programs, the control statements must include an INPFIL statement specifying EXIT.

Parameter List

When using an E15, a pointer to the 4-word parameter address list is passed to the user in register 1. (See Figure 349.)

POINTER TO:	
1. ...	Record
2. ...	Action word
3. ...	Record length
4. ...	Record type

Figure 349. Parameter List for E15

The **Record** parameter will contain the address of the next record to be processed. However, if EXIT has been specified on an INPFIL statement, this parameter will be set to zeros, since the sort/merge now expects the appropriate record address to be placed in the Record parameter. Whether the user or the sort/merge is placing addresses, at end of file the Record parameter will be filled with zeros.

The **Action word** parameter is where a code must be placed to tell the sort/merge what to do when it receives control.

The **Record length** parameter will give the address of a 20-byte (five word) area containing the record length values l_1, l_2, l_3, l_4, and l_5, as described in the length parameter of the RECORD control statement.

The **Record type** parameter will give the address of a 1-byte area coding the input record type: X'80' for fixed-length input and X'40' for variable-length.

The following codes may be passed to the sort/merge in the Action word.

CODE (in hexadecimal)

00 **Process normally**. Place X'00' in the Action word whenever the sort/merge should process a user record, whether it has been changed or not. However, if the record is changed, it is necessary to move it first to a work area, make the changes, and then put the address of the changed record in the Record parameter. If the record has not been changed, the sort/merge will supply the correct record address in this parameter.

04 **Delete record.** Whenever a record is to be deleted from the input file, place X'04' in the Action word. The sort/merge will then delete that record and return to the exit with the address of the next record to be processed in the Record parameter.

08 **No return.** Once an E15 exit has been entered, the sort/merge will keep returning to it until it receives an X'08' in the Action word. In other words, it is necessary to tell the sort/merge when this exit is no longer required.

0C **Insert record** When a record is to be inserted by the user, replace the address in the Record parameter with the main storage address of the record that is to be added. Then, place X'0C' in the low order byte of the Action word. When the sort/merge receives this code, it will insert the user's record and return the address of the next record to be processed (the one replaced by the user) in the Record parameter.

If additions are to be made at the end of the input file, check for zeros in the Record parameter. Any number of records can be added at this point, since the sort/merge will continue to return to exit E15 until an X'08' is coded to signal no return.

10 **End sort/merge.** If the sort/merge is to be terminated, place an X'10' in the low-order byte of the Action word. This will cause the sort/merge to go directly to EOJ. A message will then be given indicating how many records were processed in the last phase before the EOJ.

Note: If EXIT has been specified on an INPFIL statement, only the last three codes may be used.

A Sample EXIT E15 Program

The E15 program in Figure 350 should be compared with the SYSIPT examples in "Chapter 5. Job Control Language and Sample Control Statement Streams" (see Figure 308 and Figure 310). For increased efficiency when using card input for a sort, the SYSIPT parameter should be used.

```
STMT   SOURCE STATEMENT

 1          START 0
 2          USING BTBL15,15     R15 BASE REG FOR BRANCH TABLE BTBL15
 3          PRINT NOGEN
 4 BTBL15   DC    A(0)          NO E11
 5          B     E15           BRANCH TO EXIT E15
 6          DC    A(0)          NO E17
 7          DC    A(0)          NO E18
 9 *        ESTABLISH ADDRESSABILITY AND LINKAGE
11          DROP  15            R15 TO BE USED BY LIOCS
12          USING BTBL15,11     R11 NOW BASE REGISTER
13 E15      STM   14,12,12(13)  STORE REGISTERS IN SORT SAVE AREA
14          LR    11,15         ESTABLISH ADDRESSABILITY
15          ST    13,SAVEAREA+4 STORE SORT SAVE ADDRESS IN EXIT SAVE
16          LR    12,13         SAVE REGISTER13
17          LA    13,SAVEAREA   POINT TO EXIT SAVE
18          ST    13,8(12)      STORE EXIT SAVE ADDRESS
19          B     LINKS         CONTINUE PROCESSING
20 SAVEAREA DC    18A(0)        EXIT SAVE AREA
22 *        EXIT IS NOW SET UP TO HAVE ITS OWN REGISTERS STORED IF NEEDED
24 *        GET SYNCSORT PARAMETER ADDRESS LIST
26 LINKS    LR    4,1           LOAD ADDRESS OF PARAMETER LIST
27          L     5,4(1)        LOAD ADDRESS OF ACTION WORD
28          LA    6,INCARD      LOAD ADDRESS OF DTF
30 *        INPUT PROCEDURES
32 CARDP    NOP   READC         OPEN FILE FIRST TIME ONLY
33          OPENR (6)           OPEN SYSIPT RELOCATABLE
34          MVI   CARDP+1,X'F0' CHANGE NOP TO BC 15
35 READC    GET   (6)           READ A CARD
36          LA    7,CRDBUF      POINT TO INPUT BUFFER
37          ST    7,0(4)        STORE POINTER IN SORT PARAM LIST
38          LA    7,12          LOAD CODE 12 IN REG 7 (INSERT A RECORD)
40 *        RESTORE REGISTERS AND RETURN TO SYNCSORT
42 GOBACK   ST    7,0(5)        PUT CODE IN ACTION WORD IN PARAM LIST
43          L     13,4(13)      LOAD ADDRESS OF SORT SAVE AREA
44          L     14,12(13)     RESTORE REGISTER 14 (RETURN REG)
45          LM    0,12,20(13)   RESTORE REGISTERS 0 THROUGH 12
46          BR    14            BRANCH TO RETURN CONTROL TO SYNCSORT
48 *        END OF FILE PROCESSING
50 ENDFIL   LA    7,8           LOAD CODE 8 IN REGISTER 7 (NO RETURN)
51          CLOSER (6)          CLOSE SYSIPT RELOCATABLE
52          B     GOBACK        BRANCH TO RESTORE AND RETURN ROUTINE
54 *        LIOCS
56 CRDBUF   DC    80C' '
57 INCARD   DTFCD DEVADDR=SYSIPT,IOAREA1=CRDBUF,EOFADDR=ENDFIL
58 IJFZIZO  CDMOD               LIOCS CARD LOGIC MODULE
59          END
```

Figure 350. Sample E15 Exit Program

Coding a COBOL E15 Exit Program

An E15 exit program can be coded in COBOL/VSE. A COBOL E15 exit program is indicated in the MODS control statement.

```
MODS PH1=(MYCOBE15,COBOL,E15)
```

Figure 351. Sample MODS control statement for a COBOL E15

A COBOL exit uses GETVIS space to load run-time library routines. Allow at least 500K of extra GETVIS when executing SyncSort with a COBOL exit. 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.

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.
```

Figure 352. Sample LINKAGE SECTION for Fixed-Length Records

- 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.
```

Figure 353. Sample LINKAGE SECTION for Variable-Length Records

- 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.
```

Figure 354. (Page 1 of 2) Sample COBOL E15, Fixed-Length Records

```
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.
```

Figure 354. (Page 2 of 2) Sample COBOL E15, Fixed-Length Records

Sample COBOL E15, Variable-Length Records

```
IDENTIFICATION DIVISION.
PROGRAM-ID. E15VL19C.

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 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'.
```

Figure 355. (Page 1 of 3) Sample COBOL E15, Variable-Length Records

```
    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
```

Figure 355. (Page 2 of 3) Sample COBOL E15, Variable-Length Records

```
       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.
```

Figure 355. (Page 3 of 3) Sample COBOL E15, Variable-Length Records

Coding a C E15 Exit Routine

An E15 exit program can be coded in C/VSE. A C E15 exit program is indicated in the MODS control statement.

```
MODS PH1=(MYCE15,C,E15)
```

Figure 356. Sample MODS control statement for a C E15

A C exit is called by SyncSort as a subroutine, so it must not contain a **main()** routine.

A C exit uses GETVIS space to load runtime library routines. Allow at least 4 MB of extra GETVIS when executing SyncSort with a C exit. 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.

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)
```

Figure 357. Fixed-Length Records - Function Definition

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 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** *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 or does not exist, 08 will be passed every time including the first time.

record_up 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.

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 E15. The struct_ins_rep data type represents a structure that describes the fields within the inserted or replaced record.

dummy1 - dummy5 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.

comm_len This parameter points to a variable that defines the communication area length (always 256 bytes).

communication_area The communication_area parameter contains a pointer to the communication area. The struct_ca data type represents a user-defined structure that describes the fields in the communication area. This area can be used to pass information between exits.

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)
```

Figure 358. Variable-Length Records - Function Definition

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 (always 256 bytes).

communication_area The communication_area parameter contains a pointer to the communication area. The struct_ca data type represents a user-defined structure that describes the fields in the communication area. This area can be used to pass information between exits.

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 *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. 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;}
```

Figure 359. (Page 1 of 2) Sample C E15, Fixed-Length Records

```
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);
}
```

Figure 359. (Page 2 of 2) Sample C E15, Fixed-Length Records

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;}
```

Figure 360. (Page 1 of 3) Sample C E15, Variable-Length Records

```
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);
               }
```

Figure 360. (Page 2 of 3) Sample C E15, Variable-Length Records

```
                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);
}
```

Figure 360. (Page 3 of 3) Sample C E15, Variable-Length Records

E25 Programs

Exit E25 programs can be used to sum or delete fixed-length records during Phase 2 of the sort. Exit E25 will be entered just before each record in a string of records is moved to the output buffer, except in the case of the first record of the string, which will already have been moved. Because you are not allowed to delete the record already in the output buffer, you are cautioned that you may miss deleting certain records if they turn up at the beginning of a string. You may also miss examining entire groups of records, if the sort does not need to process the entire input file in Phase 2. And, in the event that the input file can be processed without going through Phase 2 at all, your E25 program will not run.

Parameter List

The three-word parameter list of addresses (see Figure 361) is passed to the user in register 1.

POINTER TO:
1. ... Record 2. ... Previous record 3. ... Action word

Figure 361. Parameter List for E25

The **Record** parameter will contain the address of the record to be processed next.

The **Previous record** will give the address of the previous record in the series, which is now in the output buffer.

The **Action word** points to the word where the code is placed to signal the sort on what action to take. The following codes are used.

CODE (in hexadecimal)

00 **Process normally.** Place an X'00' in the Action word when the sort is to process the present record and the previous record normally.

04 **Delete record.** The record that is waiting to be processed can be deleted by placing X'04' in the Action word. This is the record whose address is given in parameter 1. (It is not possible to delete the record whose address is given in parameter 2.) The sort will delete the record and return control to the user with the address of a new record in parameter 1. (The address of the previous record in parameter 2 will remain unchanged.)

If a pair of records is to be summed, place the total in the record in the output buffer (but do not alter the length of that record) and delete the record about to be processed. Continue summing at this point with new records as many times as is necessary.

08 **No Return.** X'08' should be placed in the Action word when the exit has completed and it is no longer necessary to return to this exit. The sort will then continue, but will not return to this exit.

10 **End sort.** Place X'10' or decimal 16 in the Action word to end the sort program as well as the user's program. This will cause the sort to go to EOJ, and a message will be sent indicating the number of records that were processed in this phase.

E35 Programs

Use an exit E35 program to insert or delete records, to change their contents or their lengths, or to sum them. If records are lengthened or shortened, the new length must be referenced in the l_3 parameter of the RECORD control statement. (For variable-length records, the maximum new length should be given as the l_3 value.)

The E35 exit will be entered before every output record is written. If desired, write a program that writes the output on one or more devices. If this is done, it is necessary to include an OUTFIL control statement specifying EXIT and assume entire responsibility in the program for opening, writing, and closing the output file. Do not use X'00' and X'0C' codes with this type of program.

Parameter List

Figure 362 shows the six-word parameter list that is pointed to by the user in register 1.

POINTER TO:	
1. ...	Record
2. ...	Previous record
3. ...	Action word
4. ...	Out-of-sequence record
5. ...	Record length
6. ...	Record type

Figure 362. Parameter List for E35

The **Record** parameter contains the address of the record to be processed next. When all the records have been processed, this parameter will be filled with zeros.

The **Previous record** parameter contains the address of the previous record, which has been processed and has moved on to the output buffer. Before the first record has been processed, this parameter will be filled with zeros.

The **Action word** points to a word where a code has been placed that informs the sort/merge what action is to be taken.

The **Out-of-sequence record** parameter is used when a record is to be inserted whose control field is not in sequence with those of other records. It points to a word filled with zeros.

The **Record length** parameter will give the address of a 20-byte (5-word) area containing the record length values l_1, l_2, l_3, l_4, and l_5.

The **Record type** parameter will give the address of a 1-byte area coding the input record type: X'80' for fixed-length input, X'40' for variable-length.

The following codes may be passed to the sort/merge in the Action word.

CODE (in hexadecimal)

00 **Process normally.** (Cannot be used with OUTFIL EXIT.) This is to be used when the sort/merge is to process records normally. If a record is to be changed, code X'00', move the record to a work area, make the change, and place the address of the changed record in the Record parameter. If a control field is changed in any way (including if a part of it is removed when a record is shortened), it is necessary to place a nonzero value somewhere in the word addressed by the fourth parameter. The sort/merge will then accept this now out-of-sequence record.

04 **Delete record.** If the sort/merge is to delete a record whose address is in parameter 1, place X'04' in the Action word.

08 **No return** When the program is completed and there is no longer a need to return to this exit, place X'08' in the Action word. The sort/merge will then continue but will not return to this exit.

0C **Insert record.** (Cannot be used with OUTFIL EXIT.) When a record is to be inserted in the output file, put the address of this record in parameter 1 in place of the address of the record to be processed next, and store an X'0C' or decimal 12 in the Action word. The sort/merge will insert the record, and when control is returned to the user, the address of the inserted record will be in parameter 2, and the address of the record that was to have been processed next will again appear in parameter 1. Records can continue to be inserted at this point or normal processing can be resumed.

If the end of the output file has been reached (in this case, the first parameter will contain zeros), additional records can be inserted because the sort/merge will keep returning to this exit until a code X'08' in the Action word has been reached.

Since the records at this final output stage are all in order, be sure to place a nonzero value in the out-of-sequence record parameter if a record is added that is not in sequence. This fourth parameter is a fullword of zeros, and nonzero values may be placed anywhere in that word.

10 **End sort/merge.** Place an X'10'or decimal 16 in the Action word when the sort/merge program is to be terminated. This will cause the sort/merge to go to EOJ, and a message will be sent informing the user how many records were processed during this last phase.

Coding a COBOL E35 Exit Program

An E35 exit program can be coded in COBOL/VSE. A COBOL E35 exit program is indicated in the MODS control statement.

```
MODS PH3=(MYCOBE35,COBOL,E35)
```

Figure 363. Sample MODS control statement for a COBOL E35

A COBOL exit uses GETVIS space to load run-time library routines. Allow at least 500K of extra GETVIS when executing SyncSort with a COBOL exit.

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.

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.
```

Figure 364. Sample LINKAGE SECTION for Fixed-Length Records

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 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.
     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.
```

Figure 365. Sample LINKAGE SECTION for Variable-Length Records

- 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 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 *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 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 output 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-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 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
```

Figure 366. (Page 1 of 2) Sample COBOL E35, 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).

     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.

    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.
```

Figure 366. (Page 2 of 2) Sample COBOL E35, Fixed-Length Records

Sample COBOL E35, Variable-Length Records

```
IDENTIFICATION DIVISION.

PROGRAM-ID. E35VL101.

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 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.
```

Figure 367. (Page 1 of 3) Sample COBOL E35, 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.
```

Figure 367. (Page 2 of 3) Sample COBOL E35, Variable-Length Records

```
       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.
```

Figure 367. (Page 3 of 3) Sample COBOL E35, Variable-Length Records

Coding a C E35 Exit Routine

An E35 exit program can be coded in C/VSE. A C E35 exit program is indicated in the MODS control statement.

```
MODS PH3=(MYCE35,C,E35)
```

Figure 368. Sample MODS Control Statement for a C E35

A C exit is called by SyncSort as a subroutine, so it must not contain a **main()** routine.

A C exit uses GETVIS space to load runtime library routines. Allow at least 4 MB of extra GETVIS when executing SyncSort with C exit.

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.

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)
```

Figure 369. Fixed-Length Records - Function Definition

The following describes the parameters used in Figure 369.

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 (always 256 bytes).

communication_area The communication_area parameter contains a pointer to the communication area. The struct_ca data type represents a user-defined structure that describes the fields in the communication area. This area can be used to pass information between exits.

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)
```

Figure 370. Variable-Length Records - Function Definition

The following describes the parameters used in Figure 370.

exit_status This parameter points to a variable containing exit status codes. See the exit_status definition for a fixed-length C E35 exit for the code definitions.

record_up The record_up parameter contains a “universal” pointer to the record leaving Phase 3. 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 universal pointer.

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 E35. The void* pointer can be cast to point an appropriate structure to describe the work record.

in_buf The in_buf 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 void* 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 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.

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 user-defined structure that describes the fields in the communication area. This information can be used to pass information between exits.

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. 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 and E35 is the only "output," SyncSort terminates immediately, since any subsequent records will never be seen.

12 *Insert a record.* This instructs SyncSort to add the record in the work area to the output 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.

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;
                    }
```

Figure 371. (Page 1 of 2) Sample C E35, Fixed-Length Records

```
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);
}
```

Figure 371. (Page 2 of 2) Sample C E35, Fixed-Length Records

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)
{
```

Figure 372. (Page 1 of 3) Sample C E35, Variable-Length Records

```
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;}
```

Figure 372. (Page 2 of 3) Sample C E35, Variable-Length Records

```
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);

}
```

Figure 372. (Page 3 of 3) Sample C E35, Variable-Length Records

EXIT E32 - Merge Only - Changing and Substituting Records, Reading Input

An E32 exit can only be used while running a merge. It is used to change the contents of a record or to substitute a record. If an INPFIL control statement specifying EXIT is included, E32 is being used to process the input files. A one-word parameter list will be passed to the user if EXIT is not specified on the INPFIL control statement, and a five-word parameter list will be passed if it is included.

Changing Records

Exit E32 is entered before each record of the input files is processed. Control, as well as noncontrol, areas may be changed as long as the total length of the record is not altered and the change in the control field does not result in an out-of-sequence record. The address of the record to be processed will appear in a one-word parameter list pointed to by register 1.

POINTER TO:
1. ... Record area

Figure 373. Parameter List for E32 (EXIT Not Specified on INPFIL Statement)

Whenever it is necessary to change a record: move the record to a work area, make the change, and place the address of the changed record in the Record area parameter. Then, return control to the merge. (It is not necessary to pass a code telling the merge what action to take.)

Substituting Records

A record can be replaced with a new record by putting the address of the new record in the Record area parameter over the address of the record being replaced. Then, pass control to the merge. When control is returned to the exit, the old record will have been deleted and the new one inserted in its place, and the address of the next record to be processed will appear in the Record area parameter. This is a one-for-one exchange; one record must be removed for every record inserted. Also, the record inserted must be in sequence. (The merge will be automatically terminated if this is not so.) Again, it is not necessary to give the merge a code telling it what action to take.

Reading Input

It is necessary to take complete responsibility for opening the files and processing their labels if the program is to read the input files. Records may be changed, inserted or deleted, providing they are passed in their final form to the merge. The merge will then process the records and write them on an output file; or, the output can be written using an E35 exit program. The merge will return control to E32 each time an individual record has been processed.

An INPFIL control statement specifying EXIT must be included with every program that reads input. Figure 374 illustrates a five-word parameter list that will be pointed to by the user in register 1.

POINTER TO:
1. ... Record area 2. ... Input file number 3. ... Action word 4. ... Record length 5. ... Record type

Figure 374. Parameter List for E32 (EXIT Specified on INPFIL Statement)

Place the address of the record to be passed to the merge in the **Record area** parameter.

Subsequently, find out from which input file to select the next record. This will be determined by the merge and the number of the file passed in the area pointed to by word 2, the **Input File Number** parameter. The merge will place a hexadecimal code in this fullword, using 0 to indicate the first file and incrementing by four for each successive file; (0=file 1, 4=file 2, 8=file 3 ...).

A code must be placed in the Action word parameter telling the merge what action to take each time a record is passed.

The **Record length** parameter will give the address of a 20-byte (five-word) area containing the record length values l_1, l_2, l_3, l_4, and l_5.

The **Record type** parameter will give the address of a 1-byte area coding the input record type: X'80' for fixed-length input, X'40' for variable-length.

The following codes may be passed to the merge in the Action word.

CODE (in hexadecimal)

08 **End of file.** Whenever the end of a file is reached place X'08' in the Action word. (There will be a different DTF for each input file so each file will have a different end-of-file address.)

0C **Accept this record.** Place this code in the Action word every time a record is passed to the merge.

10 **End merge.** If, for any reason, the merge is to be terminated, place this code in the Action word.

Coding a COBOL E32 Exit Program

An E32 exit program can be coded in COBOL/VSE. A COBOL E32 exit program is indicated in the MODS control statement.

```
MODS  PH3=(MYCOBE32,COBOL,E32)
```

Figure 375. Sample MODS control statement for a COBOL E32

A COBOL exit uses GETVIS space to load runtime library routines. Allow at least 500K of extra GETVIS when executing SyncSort with a COBOL exit.

Like any other E32 exit program, the COBOL E32 exit program is called each time an input record is processed in a MERGE or COPY. Communication between SyncSort and the COBOL E32 exit takes place in the LINKAGE SECTION of the COBOL program.

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.

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  FILE-NUMBER            PIC 9(8) COMPUTATIONAL.
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.

01  EXIT-ID                PIC X(3).
```

Figure 376. Sample LINKAGE SECTION for Fixed-Length Records

- 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 and sixth through eighth definitions define dummy areas.
- For the fifth definition (FILE-NUMBER) specify PIC 9(8) COMPUTATIONAL. SyncSort places the input file number from which the next record is to be read. (0=file 1, 4=file 2, 8=file 3, ...)
- 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.

- For the eleventh definition (EXIT-ID) specify PIC X(3). SyncSort places the literals 'E32' here.

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  DUMMY1                  PIC X.
01  FILE-NUMBER             PIC 9(8)    COMPUTATIONAL.
01  LEN-RU                  PIC 9(8)    COMPUTATIONAL.
01  LEN-WK                  PIC 9(8)    COMPUTATIONAL.
01  DUMMY2                  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.


01  EXIT-ID                 PIC X(3).
```

Figure 377. Sample LINKAGE SECTION for Variable-Length Records

- 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 area.
- For the fifth definition specify (FILE-NUMBER) specify PIC 9(8) COMPUTATIONAL. SyncSort places the input file number from which the next record is to be read. (0=file 1, 4=file 2, 8=file 3, ...)
- 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 E32 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.
- For the eleventh definition (EXIT-ID) specify PIC X(3). SyncSort places the literals 'E32' here.

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.

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.

This return code is valid only when EXIT is not specified on the INPFIL control statement.

8 *Do not return to this exit.* This instructs SyncSort to close the exit for the remainder of the sort application.

12 *Accept this record.* This instructs SyncSort to accept the record in the WORK area. This return code is valid only when EXIT is specified on the INPFIL control statement.

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.

This return code is valid only when EXIT is not specified on the INPFIL control statement.

Coding a C E32 Exit Program

An E32 exit program can be coded in C/VSE. A C E32 exit program is indicated in the MODS control statement.

```
MODS PH3=(MYCE32,C,E32)
```

Figure 378. Sample MODS control statement for a C E32

A C exit is called by SyncSort as a subroutine, so it must not contain a **main()** subroutine.

A C exit uses GETVIS space to load runtime library routines. Allow at least 4 MB of extra GETVIS when executing SyncSort with a C exit.

Like any other E32 exit program, the C E32 exit program is called each time an input record is processed in a MERGE or COPY. SyncSort and the C exit communicate 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.

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 E32exit (  int *exit_status,
           struct_ru *record_up,
           struct_ins_rep *work,
           int *dummy1,
           int *file_number,
           int *dummy2, int *dummy3, int *dummy4,
           int *comm_len,
           struct_ca *communication_area,
           char exit_id [3])
```

Figure 379. Fixed-Length Records - Function Definition

The following describes the parameters used in Figure 379.

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 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** *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 or does not exist, 08 will be passed every time including the first time.

record_up The record_up parameter contains a pointer to the record being passed to the E32 from the SORTIN. The struct_ru data type represents a structure that describes the fields within the SORTIN 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 E32. The struct_ins_rep data type represents a structure that describes the fields within the inserted or replaced record.

dummy1 - dummy4 These parameters define unused place holders. They are used with variable-length E32 and E35 communication. Their definition here allows a common parameter list for fixed-length and variable-length C E32 and E35 exits.

file_number This parameter points to an integer number that indicates which input file SyncSort is requesting input from. The number starts from 0 and increments by 4. (i.e. 0=file 1, 4=file 2, 8=file 3, etc.) When E32 encounters end-of-file on the indicated file, it should return with code 8, and SyncSort will no longer request input from that file.

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.

exit_id The exit_id parameter gives the address of a three character array that contains the literals 'E32'. This parameter may be used to identify an E32 call when the exit handles both E32 and E35 calls.

Variable-Length Records - Function Definition

```
int E32exit (  int *exit_status,
           struct_ru *record_up,
           struct_ins_rep *work,
           int *dummy1,
           int *file_number,
           int *len_ru,
           int *len_wk,
           int *dummy2,
           int *comm_len,
           struct_ca *communication_area,
           char exit_id [3])
```

Figure 380. Variable-Length Records - Function Definition

The following describes the parameters used in Figure 380.

exit_status This parameter points to a variable containing exit status codes. See the exit_status definition for a fixed-length C E32 exit for the code definitions.

record_up The record_up parameter contains a “universal” pointer to the record being passed to the E32 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 E32. The void* pointer can be cast to point an appropriate structure to describe the work record.

dummy1 - dummy2 These parameters define unused place holders. They are used with C E35 communication. Their definition here allows a common parameter list for C E32 and E35 exits.

file_number This parameter points to an integer number that indicates which input file SyncSort is requesting input from. The number starts from 0 and increments by 4. (i.e. 0=file 1, 4=file 2, 8=file 3, etc.) When E32 encounters end-of-file on the indicated file, it should return with code 8, and SyncSort will no longer request input from that file.

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.

exit_id The exit_id parameter gives the address of a three character array that contains the literals 'E32'. This parameter may be used to identify an E32 call when the exit handles both E32 and E35 calls.

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 *Accept this record.* This instructs SyncSort to accept the (unaltered) record in the record_up area.

This return code is valid only when EXIT is not specified on the INPFIL control statement.

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 *Accept this record.* This instructs SyncSort to accept the record in the WORK area. This return code is valid only when EXIT is specified on the INPFIL control statement.

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. When replacing a variable-length record, insure that its length is indicated in the len_wk parameter.

This return code is valid only when EXIT is not specified on the INPFIL control statement.

EXITS E17 and E37 - Writing and Processing Labels

These exits are used to process nonstandard or user standard trailer labels and to close files, and at exit 37 a trailer label can be written on the output file. The address of the following one-word parameter list is passed in register 1 for both exits.

POINTER TO:
1. ... Block count

Figure 381. Parameter List for E17 and E37

E17 Programs

This exit is used to process nonstandard or user standard tape trailer labels on the last input volume. (All preceding input volumes with nonstandard or user standard labels must be processed by an E11 program.)

The sort/merge will pass the address of the block count of the last input volume in the **Block count** parameter. After processing the trailer label, the input file must be closed. (If any files from the user program are used during Phase 1 of the sort, they can be closed at this exit.)

E37 Programs

This exit is used to process nonstandard and user standard tape trailer labels on merge input files and on the output file for both sort and merge programs. In addition, you may write and process nonstandard and user standard trailer labels on merge input files and on sort and merge output files. Also, if the user program uses any tape files during Phase 3, trailer labels for them can be processed at this exit.

After you have written and processed trailer labels, the files can be closed.

For a merge program, SyncSort provides a list of 4-byte block counts for the output file and for each input file. If any of these files are unlabeled or have standard labels, these 4-byte entries will be set to zero. The address of the list will be passed in the Block count parameter in register 1.

For a sort program, the address of the block count for the output volume just processed will be placed in register 1. This will point to a 4-byte entry.

Exits E18, E38, and E39—VSAM Exits

When VSAM files are sorted or merged, exit routines can be entered at E18, E38, or E39 for the following reasons:

1. To supply passwords.

2. To supply an exit list for VSAM input and output files.

The following sections describe VSAM specific coding instructions for exits E18, E38, and E39.

Passwords for VSAM Files

SyncSort for z/VSE allows password-protected VSAM files to be used for input and output. When a password is needed for a VSAM file, the appropriate user routine is entered at exit E18 (for sort input), E38 (for merge input), and E39 (for sort/merge output) before the files are opened by SyncSort. In the routine, the user can supply the required password(s). If the VSAM file is password-protected and a sort exit does not provide a password, SyncSort for z/VSE will either request that password to be entered at the operator's console or ABEND depending on the ATTEMPTS parameter of the DEFINE CLUSTER for the data set.

When a user activates the E18, E38, and E39 exits, register 1 will contain the address of a 12-byte parameter list. Refer to Figure 382 for the format of a parameter list.

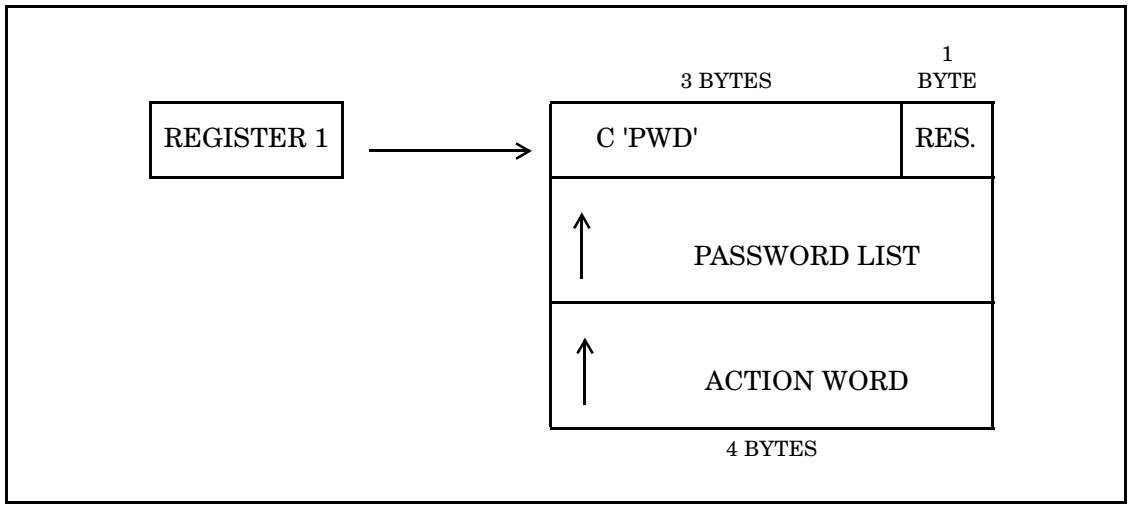


Figure 382. E18, E38, and E39 VSAM Password Parameter List

The parameter list contains a 3-byte flag that indicates the purpose for which the exit routine will be used. In this case, 'PWD' stands for password retrieval.

Note: If the user does not want a specific situation, the user routine should return to SyncSort leaving a return code of "0" in the Action Word. This code tells SyncSort to ignore the entry.

When the VSAM exit routine gets initial control, the pointer to the password list will be zeros. To supply passwords, the user must create a password list and ensure that the parameter list pointer is set to the password list address.

The format of the password list is shown in Figure 383.

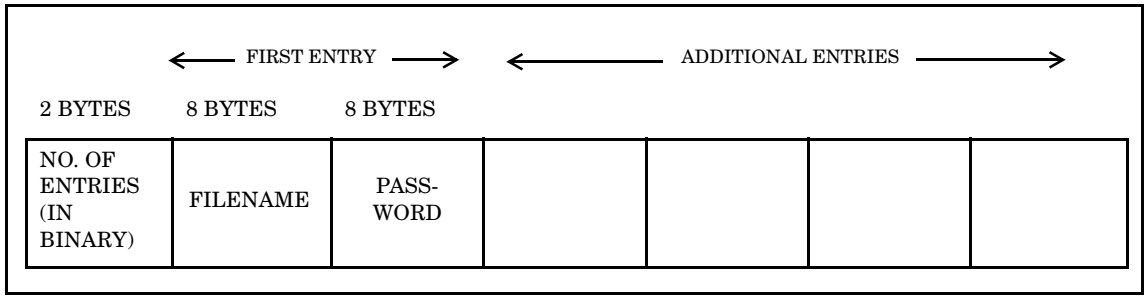


Figure 383. E18, E38, and E39 Password List Format

As soon as the user routine returns control after the password list is created, the Action Word will be checked for one of the following hexadecimal codes:

0 Ignore this entry.

4 Accept passwords.

8 Do not return to this exit for the exit list.

A user's routine is entered only once at an exit for passwords. All necessary passwords must be supplied at that point.

Exit List for VSAM Files

An exit list for VSAM input and output data sets can be supplied using exits E18, E38, or E39. The exit list must be constructed using the VSAM EXLST macro. All VSAM exit routines pointed to by this macro must return to VSAM by standard VSAM linkage. All conventions governing VSAM exit lists must be observed. Consult IBM's *VSE/ESA System Macro User's Guide* or *VSE/VSAM User's Guide and Application Programming*.

SyncSort comes to the user's sort/merge exit (containing the VSAM exit list and routines) once for each exit. The parameter list that the sort/merge passes to the exit has the format shown in Figure 384.

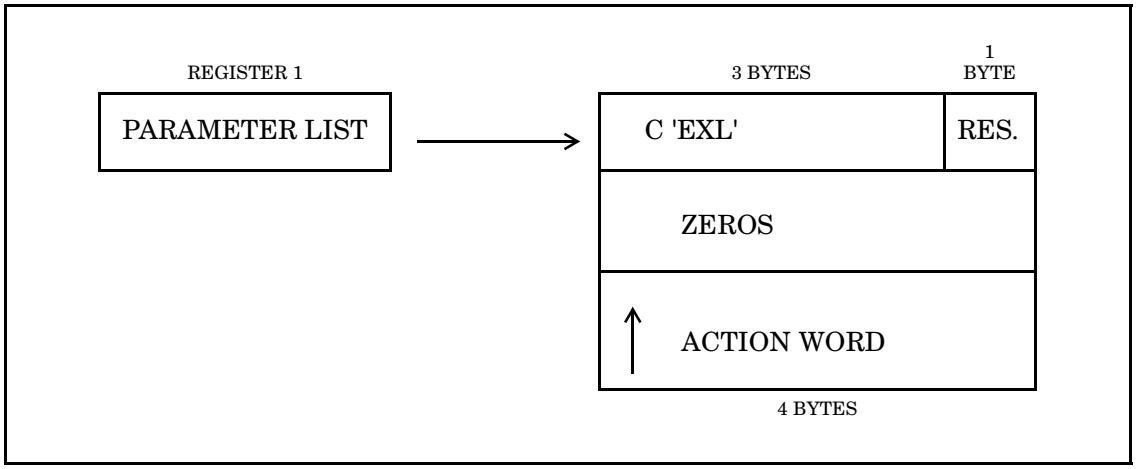


Figure 384. Parameter List Passed to the VSAM EXLST Exit

After saving any registers that were used, the 3-byte flag should be checked. In this case, 'EXL' stands for exit list. If it is not the proper flag, return to the sort/merge leaving return code "0" in the Action Word.

The exit list address should be stored in the second word. In addition, a return should be made to sort/merge with one of the following return codes in the Action Word; it is pointed to by the third word of the parameter list:

0 Ignore this entry.

4 Accept the exit list address.

8 Do not return to this sort/merge exit.

Exit Coding Instructions

To assist the user in coding exits E18, E38, and E39, the following instructions are provided. Four areas of information will be discussed for each exit. They include the building of an exit's parameter, password, and exit lists, and a specific coding procedure.

E18 Parameter List Contains the following data:

- Type indicator (C'PWD', which denotes that passwords are being requested, or C'EXL', which denotes that an exit list is being requested).
- Zeros (cleared for address of password list).
- Address of Action Word.

Note: The return code must be placed in the rightmost byte of the Action Word. Valid codes are: 0 - no reply, 4 - reply provided, and 8 - do not return.

E18 Password List Begins with a 2-byte entry count. It continues with a 16-byte entry for each password-protected sort input file. Refer to Figure 383 for an E18 password list format.

E18 Exit List Must adhere to rules found in IBM's *VSE/ESA System Macro User's Guide* or *VSE/VSAM User's Guide and Application Programming*. Those routines to which the list points must use standard VSAM linkage to return to VSAM.

Note: VSAM EXLST routines, indicated by the EXLST entry to E18/E38/E39, must not use the SAVE area indicated in register 13. It is in use by VSAM. The user is responsible for providing a SAVE area.

E18 Routine Procedure E18 is entered twice, as follows:

- The first time, parameter 1 contains C'PWD' (plus a padding byte).
- The second time, parameter 1 contains C'EXL' (plus a padding byte).

If no reply is desired, place a code of "0" in the Action Word. If not, place the address of the password or exit list (whichever is requested) in parameter 2. Return with a code of "4".

E38 Parameter List Contains the following data:

- Type indicator (C'PWD', which denotes passwords are being requested, or C'EXL', which denotes the exit list is being requested).
- Zeros (cleared for address of password list).
- Address of Action Word.

Note: The return code must be placed in the rightmost byte of the Action Word. Valid codes are: 0 - no reply, 4 - reply provided, and 8 - do not return.

E38 Password List Begins with a 2-byte entry count. It continues with a 16-byte entry for each password-protected merge input file. Refer to Figure 383 for an E38 password list format.

E38 Exit List Must adhere to rules found in IBM's *VSE/ESA System Macro User's Guide* or *VSE/VSAM User's Guide and Application Programming*. Those routines to which the list points must use standard VSAM linkage to return to VSAM.

Note: VSAM EXLST routines, indicated by the EXLST entry to E18/E38/E39, must not use the SAVE area indicated in register 13. It is in use by VSAM. The user is responsible for providing a SAVE area.

For end-of-file processing, the EODAD exit must not be used because the sort will not be able to detect the end-of-file. In its place, use a LERAD exit. In addition, test the FDBK code for "4", which indicates the end-of-file. Do not change the code as the sort also uses it.

Note: The same exit list must be valid for all SORTIN files.

E38 Routine Procedure E38 is entered twice, as follows:

- The first time, parameter 1 contains C'PWD' (plus a padding byte).
- The second time, parameter 1 contains C'EXL' (plus a padding byte).

If no reply is desired, ensure a code of "0" is placed in the Action Word. If not, place the address of the password or exit list (whichever is requested) in parameter 2. Return with a code of "4".

E39 Parameter List Contains the following data:

- Type indicator (C'PWD', which denotes passwords are being requested, or C'EXL', which denotes the exit list is being requested).
- Zeros (cleared for address of password list).
- Address of Action Word.

Note: The return code must be placed in the rightmost byte of the Action Word. Valid codes are: 0 no reply, 4 reply provided, and 8 do not return.

E39 Password List Begins with a 2-byte entry count. It is followed by output file name and associated password pairs. Refer to Figure 383 for an E39 password list format.

E39 Exit List Must adhere to rules found in IBM's *VSE/ESA System Macro User's Guide* or *VSE/VSAM User's Guide and Application Programming*. Those routines to which the list points must use standard VSAM linkage to return to VSAM.

Note: VSAM EXLST routines, indicated by the EXLST entry to E18/E38/E39, must not use the SAVE area indicated in register 13. It is in use by VSAM. The user is responsible for providing a SAVE area.

E39 Routine Procedure E39 is entered twice, as follows:

- The first time, parameter 1 contains C'PWD' (plus a padding byte).
- The second time, parameter 1 contains C'EXL' (plus a padding byte).

If no reply is desired, place a code of "0" in the Action Word. If not, place the address of the password or exit list (whichever is requested) in parameter 2. Return with a code of "4".

Coding REXX Exits

The exit programs E15, E32, and E35 can be coded in REXX/VSE. A REXX exit program is indicated in the MODS control statement. A LIBDEF search chain for PROCs must be established to include the library where the REXX exit program is catalogued.

```
MODS PH1=(MYE15,EXEC,E15),PH3=(REXXE35,EXEC,E35)
```

Figure 385. Sample MODS control statement for REXX exits

REXX Variables Provided by SyncSort

SyncSort 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 SyncSort.

SyncSort will automatically load these variables each time it calls a REXX exit. When the exit completes its work, it should use the following sequence of commands to return the variables to SyncSort:

```
ADDRESS SYNCREXX 'TAKE'
RETURN
```

The following table describes the special REXX variables:

VARIABLE	FUNCTION
SYRECORD	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. If SYRECORD is null, then SyncSort has no data remaining. When this happens, the exit can either CLOSE or can continue to INSERT new records.

Table 50. (Page 1 of 2) REXX Variables Provided by SyncSort

VARIABLE	FUNCTION
SYACTION	This variable should be set before the exit returns control to SyncSort. It describes the disposition of the current record. Possible values for SYACTION are as follows: ACCEPT: Retain the current record with no modification. REPLACE: Replace the current record with contents of the SYRECORD. DELETE: Delete the current record. INSERT: Insert the contents of the SYRECORD before the current record. CLOSE: Do not return to the exit. ABEND: Terminate SyncSort. If an E15/E32 is providing the input, the only valid values for SYACTION are INSERT, CLOSE, or ABEND. If an E35 is writing the output, the only valid values for SYACTION are DELETE, CLOSE, or ABEND.
SYOUTSEQ	For E35 exits only, this variable can be used in conjunction with an INSERT value for SYACTION to indicate that the new record is out of sequence. SYOUTSEQ=0 means the new record is in sequence; any other value indicates that it is not.
SYFILNUM	For E32 exits, this variable contains the file number of the file currently being processed. The numbering convention increments file numbers by 4 (e.g., FILE1=0, FILE2=4, ... FILE9=32).
SYEXITYP	This variable will automatically be set to E15, E32, or E35, depending on which type of exit is being called.
SYGBLN1... SYGBLN8	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 will insure that these variables are preserved across calls to the exit.
SYGBLSTR	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 will insure that this variable is preserved across calls to the exit.

Table 50. (Page 2 of 2) REXX Variables Provided by SyncSort

Sample REXX Exit

The following example illustrates a REXX exit that will count the number of records that are passed to the exit and convert the first 20 bytes of each record to upper case:

```
If syrecord='' Then
     Do
          syaction='CLOSE'
          Say 'REXX' syexityp 'counted' sygbln1 'records'
     End
Else
     Do
          sygbln1 = sygbln1 + 1
          Parse Var syrecord to_upper 21 no_change
          syrecord=Translate(to_upper) || no_change
          syaction='REPLACE'
     End
Address SYNCREXX 'TAKE'
Return
```

Figure 386. Sample REXX Exit

Chapter 9. Creating VSAM Alternate Index Files with SyncSort

Introduction

This chapter describes SYNCBIX, SyncSort's high performance replacement for the BLDINDEX process performed by Access Methods Services (IDCAMS). SYNCBIX interfaces with SyncSort for z/VSE to allow efficient reading of the base cluster, fast extracting of the record pointer information, primary and alternate keys, and high performance sorting of these records.

VSAM alternate index is one of VSAM's most useful features. It provides an alternate means to access records in a base cluster, through different keys. The base cluster, can be an Entry Sequenced Data Set (ESDS) or a Key Sequenced Data Set (KSDS). KSDS and ESDS alternate index files can be defined as either a UNIQUE key data set or as a NONUNIQUE key data set:

- UNIQUE indicates that every alternate key is in one and only one data record in the base cluster.
- NONUNIQUE indicates that an alternate key can be in several records in the base cluster.

The prime key of a KSDS file must be defined as a unique key data set.

Steps to Create an Alternate Index

In order to establish an alternate index to access a base cluster, the following steps are necessary, whether you are using BLDINDEX or SYNCBIX. Note that only the fifth item, building the index, differs between BLDINDEX and SYNCBIX.

1. Create the VSAM base file with the IDCAMS DEFINE CLUSTER command.

2. Load the base cluster with data.

3. Define the alternate index file with the IDCAMS DEFINE AIX command.

4. Define a path through the alternate index to the base cluster with a DEFINE PATH command for a KSDS base file.

5. Build the alternate index with the BLDINDEX process of IDCAMS or with SYNCBIX. Using SYNCBIX speeds the process because SYNCBIX uses the high-performance sorting of SyncSort for z/VSE. The process is conceptually divided into three steps: reading, sorting, and writing to output:

   - The entire base cluster is read and the record pointer information, primary key and alternate key are extracted.

   - The extracted data is sorted on the alternate key. This process may be completed entirely in virtual storage or for larger files require the use of disk work files.

   - The sorted information is formatted into alternate index data records that are loaded directly into the alternate index cluster.

Sample Alternate Index Definitions: IDCAMS and SYNCBIX

The figures below show JCL and control statements for creating alternative indexes with IDCAMS and SYNCBIX. The pairing highlights the difference between IDCAMS and SyncSort's SYNCBIX. Note that each pair is identical except for Step 4. Thus, only the actual index building step of IDCAMS (Step 4) requires changes in order to use SYNCBIX.

The following is a generalized Step 4 using SYNCBIX:

```
* Step 4: BUILD ALTERNATE INDEX USING SYNCBIX
// DLBL IJSYSUC,'catalog.fileid',,VSAM
// DLBL filename_i.,'base.cluster.fileid',,VSAM
// DLBL filename_o.,'aix.fileid',,VSAM
// DLBL SORTWK1,'sortwk.fileid',0,SD
// EXTENT SYS003,volser,1,0,strk,numtrk
// ASSGN SYS003,DISK,VOL=volser,SHR
/*
// LIBDEF PHASE,SEARCH=(lib.sublib)
// EXEC PGM=SYNCBIX,SIZE=512K
     INFILE(filename_i</pw_i>)            -
     OUTFILE(filename_o</pw_o>)           -
     PRINT(prt)                             -
     WORK(n)                                -
     SORT(sortname)
/*
```

Figure 387. Step 4 Using SYNCBIX

The following figures show the JCL and control statements for IDCAMS and SYNCBIX for four types of alternate index definition:

- Unique Key Alternate Index on a KSDS -- IDCAMS
- Unique Key Alternate Index on a KSDS -- SYNCBIX

- Non-Unique Key Alternate Index on a KSDS -- IDCAMS
- Non-Unique Key Alternate Index on a KSDS -- SYNCBIX

- Unique Key Alternate Index on a ESDS -- IDCAMS
- Unique Key Alternate Index on a ESDS -- SYNCBIX

- Non-Unique Key Alternate Index on a ESDS -- IDCAMS
- Non-Unique Key Alternate Index on a ESDS -- SYNCBIX

```
// JOB AIX1AMS
*
*       STEP 1. DEFINE KSDS BASE CLUSTER
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE CLUSTER( NAME(KSDS.BASE.CLUSTER)                    -
   FREESPACE(10 5)                                          -
   INDEXED                                                  -
   KEYS(4 28)                                               -
   VOLUMES(VOL001))                                         -
  DATA( NAME(KSDS.BASE.CLUSTER.DATA)                        -
   RECORDS(1000) RECORDSIZE(100 100))                       -
  INDEX( NAME(KSDS.BASE.CLUSTER.INDEX))                     -
  CATALOG(KSDS.USER.CATALOG)
/*
*       STEP 2. LOAD KSDS BASE CLUSTER
*
// EXEC PGM=LOADKSDS
             .
             .
             .
/*
*       STEP 3. DEFINE ALTERNATE INDEX AND PATH
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE ALTERNATEINDEX( NAME(KSDS.AIX.CLUSTER)              -
   FREESPACE(5 5)                                           -
   KEYS(8 12)                                               -
   RELATE(KSDS.BASE.CLUSTER)                                -
   UNIQUEKEY                                                -
   UPGRADE                                                  -
   VOLUMES(VOL001))                                         -
  DATA( NAME(KSDS.AIX.CLUSTER.DATA)                         -
   RECORDS(1000) RECORDSIZE(50 50))                         -
  INDEX( NAME(KSDS.AIX.CLUSTER.INDEX))                      -
  CATALOG(KSDS.USER.CATALOG)
 DEFINE PATH( NAME(PATH.TO.KSDS.BASE.CLUSTER)               -
  PATHENTRY(KSDS.AIX.CLUSTER))                              -
  CATALOG(KSDS.USER.CATALOG)
/*
*       STEP 4. BUILD ALTERNATE INDEX USING IDCAMS
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 BLDINDEX INDATASET(KSDS.BASE.CLUSTER)                      -
          OUTDATASET(KSDS.AIX.CLUSTER)                      -
          WORKVOLUMES(VOL001)                               -
          CATALOG(KSDS.USER.CATALOG)
/*
/&
```

Figure 388. Unique Key Alternate Index on a KSDS -- IDCAMS

```
// JOB AIX1SS
*
*       STEP 1. DEFINE KSDS BASE CLUSTER
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE CLUSTER( NAME(KSDS.BASE.CLUSTER)                    -
   FREESPACE(10 5)                                          -
   INDEXED                                                  -
   KEYS(4 28)                                               -
   VOLUMES(VOL001))                                         -
  DATA( NAME(KSDS.BASE.CLUSTER.DATA)                        -
   RECORDS(1000) RECORDSIZE(100 100))                       -
  INDEX( NAME(KSDS.BASE.CLUSTER.INDEX))                     -
  CATALOG(KSDS.USER.CATALOG)
/*
*       STEP 2. LOAD KSDS BASE CLUSTER
*
// EXEC PGM=LOADKSDS
           .
           .
           .
/*
*       STEP 3. DEFINE ALTERNATE INDEX AND PATH
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE ALTERNATEINDEX( NAME(KSDS.AIX.CLUSTER)              -
   FREESPACE(5 5)                                           -
   KEYS(8 12)                                               -
   RELATE(KSDS.BASE.CLUSTER)                                -
   UNIQUEKEY                                                -
   UPGRADE                                                  -
   VOLUMES(VOL001))                                         -
  DATA( NAME(KSDS.AIX.CLUSTER.DATA)                         -
   RECORDS(1000) RECORDSIZE(50 50))                         -
  INDEX( NAME(KSDS.AIX.CLUSTER.INDEX))                      -
  CATALOG(KSDS.USER.CATALOG)
 DEFINE PATH( NAME(PATH.TO.KSDS.BASE.CLUSTER)               -
  PATHENTRY(KSDS.AIX.CLUSTER))                              -
  CATALOG(KSDS.USER.CATALOG)
/*
*       STEP 4. BUILD ALTERNATE INDEX USING SYNCBIX
*
// LIBDEF PHASE,SEARCH=(lib.sublib)
// ASSGN SYS003,DISK,TEMP,VOL=VOL010,SHR
// DLBL IJSYSUC,'KSDS.USER.CATALOG',,VSAM
// DLBL BASE,'KSDS.BASE.CLUSTER',,VSAM
// DLBL AIX,'KSDS.AIX.CLUSTER',,VSAM
// DLBL    SORTWK1,'SORTWK1.FILE',0,SD
// EXTENT  SYS003,VOL010,1,0,strk,ntrk
// EXEC PGM=SYNCBIX,SIZE=2048K
 INFILE(BASE) OUTFILE(AIX) WORK(1)
/*
/&
```

Figure 389. Unique Key Alternate Index on a KSDS -- SYNCBIX

```
// JOB AIX2AMS
*
*       STEP 1. DEFINE KSDS BASE CLUSTER
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE CLUSTER( NAME(KSDS.BASE.CLUSTER)                    -
   FREESPACE(10 5)                                          -
   INDEXED                                                  -
   KEYS(8 12)                                               -
   VOLUMES(VOL001))                                         -
  DATA( NAME(KSDS.BASE.CLUSTER.DATA)                        -
   RECORDS(1000) RECORDSIZE(100 100))                       -
  INDEX( NAME(KSDS.BASE.CLUSTER.INDEX))                     -
  CATALOG(KSDS.USER.CATALOG)
/*
*       STEP 2. LOAD KSDS BASE CLUSTER
*
// EXEC PGM=LOADKSDS
            .
            .
            .
/*
*       STEP 3. DEFINE ALTERNATE INDEX AND PATH
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE ALTERNATEINDEX( NAME(KSDS.AIX.CLUSTER)              -
   FREESPACE(5 5)                                           -
   KEYS(4 28)                                               -
   RELATE(KSDS.BASE.CLUSTER)                                -
   NONUNIQUEKEY                                             -
   UPGRADE                                                  -
   VOLUMES(VOL001))                                         -
  DATA( NAME(KSDS.AIX.CLUSTER.DATA)                         -
   RECORDS(1000) RECORDSIZE(50 50))                         -
  INDEX( NAME(KSDS.AIX.CLUSTER.INDEX))                      -
  CATALOG(KSDS.USER.CATALOG)
 DEFINE PATH( NAME(PATH.TO.KSDS.BASE.CLUSTER)               -
  PATHENTRY(KSDS.AIX.CLUSTER))                              -
  CATALOG(KSDS.USER.CATALOG)
/*
*       STEP 4. BUILD ALTERNATE INDEX USING IDCAMS
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 BLDINDEX INDATASET(KSDS.BASE.CLUSTER)                      -
          OUTDATASET(KSDS.AIX.CLUSTER)                      -
          WORKVOLUMES(VOL001)                               -
          CATALOG(KSDS.USER.CATALOG)
/*
/&
```

Figure 390. Non-Unique Key Alternate Index on a KSDS -- IDCAMS

```
// JOB AIX2SS
*
*       STEP 1. DEFINE KSDS BASE CLUSTER
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE CLUSTER( NAME(KSDS.BASE.CLUSTER)                    -
   FREESPACE(10 5)                                          -
   INDEXED                                                  -
   KEYS(8 12)                                               -
   VOLUMES(VOL001))                                         -
  DATA( NAME(KSDS.BASE.CLUSTER.DATA)                        -
   RECORDS(1000) RECORDSIZE(100 100))                       -
  INDEX( NAME(KSDS.BASE.CLUSTER.INDEX))                     -
  CATALOG(KSDS.USER.CATALOG)
/*
*       STEP 2. LOAD KSDS BASE CLUSTER
*
// EXEC PGM=LOADKSDS
           .
           .
           .
/*
*       STEP 3. DEFINE ALTERNATE INDEX AND PATH
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE ALTERNATEINDEX( NAME(KSDS.AIX.CLUSTER)              -
   FREESPACE(5 5)                                           -
   KEYS(4 28)                                               -
   RELATE(KSDS.BASE.CLUSTER)                                -
   NONUNIQUEKEY                                             -
   UPGRADE                                                  -
   VOLUMES(VOL001))                                         -
  DATA( NAME(KSDS.AIX.CLUSTER.DATA)                         -
   RECORDS(1000) RECORDSIZE(50 50))                         -
  INDEX( NAME(KSDS.AIX.CLUSTER.INDEX))                      -
  CATALOG(KSDS.USER.CATALOG)
 DEFINE PATH( NAME(PATH.TO.KSDS.BASE.CLUSTER)               -
  PATHENTRY(KSDS.AIX.CLUSTER))                              -
  CATALOG(KSDS.USER.CATALOG)
/*
*       STEP 4. BUILD ALTERNATE INDEX USING SYNCBIX
*
// LIBDEF PHASE,SEARCH=(lib.sublib)
// ASSGN SYS003,DISK,TEMP,VOL=VOL010,SHR
// DLBL IJSYSUC,'KSDS.USER.CATALOG',,VSAM
// DLBL BASE,'KSDS.BASE.CLUSTER',,VSAM
// DLBL AIX,'KSDS.AIX.CLUSTER',,VSAM
// DLBL    SORTWK1,'SORTWK1.FILE',0,SD
// EXTENT  SYS003,VOL010,1,0,strk,ntrk
// EXEC PGM=SYNCBIX,SIZE=2048K
 INFILE(BASE) OUTFILE(AIX) WORK(1)
/*
/&
```

Figure 391. Non-Unique Key Alternate Index on a KSDS -- SYNCBIX

```
// JOB AIX3AMS
*
*       STEP 1. DEFINE ESDS BASE CLUSTER
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE CLUSTER( NAME(ESDS.BASE.CLUSTER)                    -
   NONINDEXED                                               -
   VOLUMES(VOL001))                                         -
  DATA( NAME(ESDS.BASE.CLUSTER.DATA)                        -
   RECORDS(1000) RECORDSIZE(100 100))                       -
  CATALOG(ESDS.USER.CATALOG)
/*
*       STEP 2. LOAD ESDS BASE CLUSTER
*
// EXEC PGM=LOADESDS
             .
             .
             .
/*
*       STEP 3. DEFINE ALTERNATE INDEX AND PATH
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE ALTERNATEINDEX( NAME(ESDS.AIX.CLUSTER)              -
   FREESPACE(5 5)                                           -
   KEYS(8 12)                                               -
   RELATE(ESDS.BASE.CLUSTER)                                -
   UNIQUEKEY                                                -
   UPGRADE                                                  -
   VOLUMES(VOL001))                                         -
  DATA( NAME(ESDS.AIX.CLUSTER.DATA)                         -
   RECORDS(1000) RECORDSIZE(50 50))                         -
  INDEX( NAME(ESDS.AIX.CLUSTER.INDEX))                      -
  CATALOG(ESDS.USER.CATALOG)
 DEFINE PATH( NAME(PATH.TO.ESDS.BASE.CLUSTER)               -
  PATHENTRY(ESDS.AIX.CLUSTER))                              -
  CATALOG(ESDS.USER.CATALOG)
/*
*       STEP 4. BUILD ALTERNATE INDEX USING IDCAMS
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 BLDINDEX INDATASET(ESDS.BASE.CLUSTER)                      -
          OUTDATASET(ESDS.AIX.CLUSTER)                      -
          WORKVOLUMES(VOL001)                               -
          CATALOG(ESDS.USER.CATALOG)
/*
/&
```

Figure 392. Unique Key Alternate Index on an ESDS -- IDCAMS

```
// JOB AIX3SS
*
*       STEP 1. DEFINE ESDS BASE CLUSTER
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE CLUSTER( NAME(ESDS.BASE.CLUSTER)                    -
   NONINDEXED                                               -
   VOLUMES(VOL001))                                         -
  DATA( NAME(ESDS.BASE.CLUSTER.DATA)                        -
   RECORDS(1000) RECORDSIZE(100 100))                       -
  CATALOG(ESDS.USER.CATALOG)
/*
*       STEP 2. LOAD ESDS BASE CLUSTER
*
// EXEC PGM=LOADESDS
           .
           .
           .
/*
*       STEP 3. DEFINE ALTERNATE INDEX AND PATH
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE ALTERNATEINDEX( NAME(ESDS.AIX.CLUSTER)              -
   FREESPACE(5 5)                                           -
   KEYS(8 12)                                               -
   RELATE(ESDS.BASE.CLUSTER)                                -
   UNIQUEKEY                                                -
   UPGRADE                                                  -
   VOLUMES(VOL001))                                         -
  DATA( NAME(ESDS.AIX.CLUSTER.DATA)                         -
   RECORDS(1000) RECORDSIZE(50 50))                         -
  INDEX( NAME(ESDS.AIX.CLUSTER.INDEX))                      -
  CATALOG(ESDS.USER.CATALOG)
 DEFINE PATH( NAME(PATH.TO.ESDS.BASE.CLUSTER)               -
  PATHENTRY(ESDS.AIX.CLUSTER))                              -
  CATALOG(ESDS.USER.CATALOG)
/*
*       STEP 4. BUILD ALTERNATE INDEX USING SYNCBIX
*
// LIBDEF PHASE,SEARCH=(lib.sublib)
// ASSGN SYS003,DISK,TEMP,VOL=VOL010,SHR
// DLBL IJSYSUC,'ESDS.USER.CATALOG',,VSAM
// DLBL BASE,'ESDS.BASE.CLUSTER',,VSAM
// DLBL AIX,'ESDS.AIX.CLUSTER',,VSAM
// DLBL    SORTWK1,'SORTWK1.FILE',0,SD
// EXTENT  SYS003,VOL010,1,0,strk,ntrk
// EXEC PGM=SYNCBIX,SIZE=2048K
 INFILE(BASE) OUTFILE(AIX) WORK(1)
/*
/&
```

Figure 393. Unique Key Alternate Index on an ESDS -- SYNCBIX

```
// JOB AIX4AMS
*
*       STEP 1. DEFINE ESDS BASE CLUSTER
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE CLUSTER( NAME(ESDS.BASE.CLUSTER)                    -
   NONINDEXED                                               -
   VOLUMES(VOL001))                                         -
  DATA( NAME(ESDS.BASE.CLUSTER.DATA)                        -
   RECORDS(1000) RECORDSIZE(100 100))                       -
  CATALOG(ESDS.USER.CATALOG)
/*
*       STEP 2. LOAD ESDS BASE CLUSTER
*
// EXEC PGM=LOADESDS
           .
           .
           .
/*
*       STEP 3. DEFINE ALTERNATE INDEX AND PATH
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE ALTERNATEINDEX( NAME(ESDS.AIX.CLUSTER)              -
   FREESPACE(5 5)                                           -
   KEYS(4 28)                                               -
   RELATE(ESDS.BASE.CLUSTER)                                -
   NONUNIQUEKEY                                             -
   UPGRADE                                                  -
   VOLUMES(VOL001))                                         -
  DATA( NAME(ESDS.AIX.CLUSTER.DATA)                         -
   RECORDS(1000) RECORDSIZE(50 50))                         -
  INDEX( NAME(ESDS.AIX.CLUSTER.INDEX))                      -
  CATALOG(ESDS.USER.CATALOG)
 DEFINE PATH( NAME(PATH.TO.ESDS.BASE.CLUSTER)               -
  PATHENTRY(ESDS.AIX.CLUSTER))                              -
  CATALOG(ESDS.USER.CATALOG)
/*
*       STEP 4. BUILD ALTERNATE INDEX USING IDCAMS
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 BLDINDEX INDATASET(ESDS.BASE.CLUSTER)                      -
          OUTDATASET(ESDS.AIX.CLUSTER)                      -
          WORKVOLUMES(VOL001)                               -
          CATALOG(ESDS.USER.CATALOG)
/*
/&
```

Figure 394. Non-Unique Key Alternate Index on an ESDS -- IDCAMS

```
// JOB AIX4SS
*
*       STEP 1. DEFINE ESDS BASE CLUSTER
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE CLUSTER( NAME(ESDS.BASE.CLUSTER)                    -
   NONINDEXED                                               -
   VOLUMES(VOL001))                                         -
  DATA( NAME(ESDS.BASE.CLUSTER.DATA)                        -
   RECORDS(1000) RECORDSIZE(100 100))                       -
  CATALOG(ESDS.USER.CATALOG)
/*
*       STEP 2. LOAD ESDS BASE CLUSTER
*
// EXEC PGM=LOADESDS
           .
           .
           .
/*
*       STEP 3. DEFINE ALTERNATE INDEX AND PATH
*
// EXEC PGM=IDCAMS,SIZE=AUTO
 DEFINE ALTERNATEINDEX( NAME(ESDS.AIX.CLUSTER)              -
   FREESPACE(5 5)                                           -
   KEYS(4 28)                                               -
   RELATE(ESDS.BASE.CLUSTER)                                -
   NONUNIQUEKEY                                             -
   UPGRADE                                                  -
   VOLUMES(VOL001))                                         -
  DATA( NAME(ESDS.AIX.CLUSTER.DATA)                         -
   RECORDS(1000) RECORDSIZE(50 50))                         -
  INDEX( NAME(ESDS.AIX.CLUSTER.INDEX))                      -
  CATALOG(ESDS.USER.CATALOG)
 DEFINE PATH( NAME(PATH.TO.ESDS.BASE.CLUSTER)               -
  PATHENTRY(ESDS.AIX.CLUSTER))                              -
  CATALOG(ESDS.USER.CATALOG)
/*
*       STEP 4. BUILD ALTERNATE INDEX USING SYNCBIX
*
// LIBDEF PHASE,SEARCH=(lib.sublib)
// ASSGN SYS003,DISK,TEMP,VOL=VOL010,SHR
// DLBL IJSYSUC,'ESDS.USER.CATALOG',,VSAM
// DLBL BASE,'ESDS.BASE.CLUSTER',,VSAM
// DLBL AIX,'ESDS.AIX.CLUSTER',,VSAM
// DLBL    SORTWK1,'SORTWK1.FILE',0,SD
// EXTENT  SYS003,VOL010,1,0,strk,ntrk
// EXEC PGM=SYNCBIX,SIZE=2048K
 INFILE(BASE) OUTFILE(AIX) WORK(1)
/*
/&
```

Figure 395. Non-Unique Key Alternate Index on an ESDS -- SYNCBIX

Syntax Rules for SYNCBIX

The following are the rules for creating card image control statements required by SYNCBIX:

- SYNCBIX scans columns 1 through 80 of all card images.
- An “*” in column 1 of a card image indicates that the card image is a comment statement. This statement will not be processed by SYNCBIX. Comment statements may be placed anywhere in the SYSIPT card image stream.
- One or more complete SYNCBIX parameters (separated by at least one blank) may be placed on a card image.
- SYNCBIX parameters may start in column 2 or later.
- Continuation is indicated by completing a parameter on a card image and then using either a "-" or "+" character. Any information after the "-" or "+" character will be treated as a comment.

SYNCBIX Parameters

The following are the parameters accepted by SYNCBIX:

INFILE Specifies the filename, $filename_i$ of the VSAM DLBL statement, and if required, the read password, pw_i of the VSAM base cluster to be used as input to the alternate index building process. This parameter is required. The read password of the INFILE parameter is only required if the VSAM file is password protected.

NOREUSE Specifies that the SYNCBIX default of REUSE is not to be applied to the VSAM alternate index cluster.

OUTFILE Specifies the filename, $filename_o$ of the VSAM DLBL statement, and if required, the write password, pw_o of the VSAM alternate index cluster or path that points to the alternate index cluster. This parameter is required. The write password of the OUTFILE parameter is only required if the VSAM file specified by the OUTFILE parameter is password protected.

PRINT Specifies the amount of information to be displayed on SYSLST. SYSLST may only be assigned to a printer. PRINT(ALL) specifies that all of the control statements and both informational and critical messages be displayed. PRINT(CRITICAL) specifies that only critical error messages be displayed. PRINT(NONE) specifies that no messages are to be displayed. PRINT(CRITICAL) is the default. The PRINT parameter must be specified first in order to affect the printing of the other SYNCBIX parameters. The PRINT parameter is optional.

SORT Specifies the phase name of SyncSort for z/VSE. The default name is SORT, which is almost always the correct name. This parameter is optional, and should not be specified without consultation with the system programmer who installed SyncSort for z/VSE.

WORK Specifies the total number in decimal of disk files to be used for SORTWK. The value of n must be between 1 and 9 and the default value of n is 1. This parameter is optional. You must provide n SORTWKx DLBLs in your JCL to be used if the data cannot be processed completely in virtual storage. See "Setting up Disk Work File Statements" on page 5.5 for more details.

SYNCBIX Job Control Statements

The following are the job control statements (JCL) required by SYNCBIX:

IJSYSUC DLBL Specifies the VSAM catalog that contains both the VSAM base cluster and the VSAM alternate index file. This statement is not needed if both files are cataloged in the master catalog or if the correct catalog is defined as IJSYSUC in standard labels. This statement is also not needed if both of the DLBLs for the base cluster and the alternate index cluster use the CAT=parameter on their DLBLs.

filename$_i$ DLBL Specifies the VSAM file that is the base cluster for the alternate index file. The INFILE parameter specifies the value of filename$_i$. This DLBL statement may use the CAT=parameter to indicate the VSAM catalog that owns this cluster.

filename$_o$ DLBL Specifies the VSAM file that is the alternate index cluster that is to created. The OUTFILE parameter specifies the value of filename$_o$. This DLBL statement may use the CAT=parameter to indicate the VSAM catalog that owns this cluster.

SORTWKx DLBL/EXTENT/ASSGN Specifies the SORTWK disk space to be used by SyncSort for z/VSE if the BLDINDEX process is too large to fit completely in virtual storage. There should be one set of SORTWK DLBL/EXTENT/ASSGN statements for each SORTWK specified by the WORK parameter. For more details on SORTWK JCL, see "Setting up Disk Work File Statements" on page 5.5. These statements may not be required if they already exist in the LABEL area.

LIBDEF

Defines the VSE library and sublibrary, (lib.sublib), which contains SYNCBIX and SyncSort for z/VSE. This statement may not be required if the SyncSort for z/VSE library is in the permanent LIBDEF chain.

EXEC

Specifies that the program to be executed is SYNCBIX and defines the virtual storage it should use. SYNCBIX will generally execute more efficiently in larger virtual storage, but reading and writing the VSAM files efficiently requires a significant amount of GETVIS storage. Using SIZE=512K or SIZE=1024K should be reasonable in most cases. However, when the maximum record length of the alternate index file is greater than 32K, the "SIZE=" value chosen must include the maximum record length (i.e., SIZE=512K+nnnn or SIZE=1024K+nnnn, where nnnn indicates the maximum record length defined for the alternate index files).

SYNCBIX Messages

SYNCBIX messages are written to SYSLST with a record length of 121 bytes and SYSLST must be ASSGNed to a print device.

SYNCBIX messages have the prefix SBIX. (See "SYNCBIX Messages" on page 12.35 for a description of SYNCBIX messages.)

Chapter 10. SyncSort Reentrant Access Method Operation

Overview

SyncSort Reentrant Access Method (SSRAM) is an interface between an invoking program and SyncSort for z/VSE. SSRAM allows a program to invoke up to sixteen concurrent sorts each of which can manipulate the same data differently.

Syntax Conventions

In this chapter, the following stylistic conventions apply. Lowercase characters in call syntax, parameter lists, and error messages indicate variable information or information that you must supply. Uppercase characters indicate literal information. Brackets in call syntax enclose optional parameters.

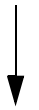
Linking to SSRAM

SSRAM has a language interface that must be linkedited to the invoking program in order to use SSRAM. The same language interface is used regardless of the language in which the invoking program is written. The language interface interprets the sort call routines.

Following is sample JCL to linkedit the interface to an invoking Assembler program.

```
//  JOB COMPILE
//  OPTION  CATAL
 PHASE USERPROG,*
//  EXEC ASSEMBLY, SIZE=500K
```

source program



```
/*
 INCLUDE SSSRAM
// EXEC LNKEDT
/*
/&
```

Figure 396. Sample JCL to linkedit to Invoking Assembler Program

Sort Call Overview

There are five calls that control the operation of sorts running under SSRAM:

- SRTCORE
- SRTOPEN
- SRTFILL
- SRTGETR
- SRTCLSE

SRTCORE *Reserves storage for all sorts.* SRTCORE allows the application program to specify the storage area which will be used by SSRAM and to limit the amount of storage SSRAM uses. This must be at least as large as the sum of the storage specified in the parameter lists for all concurrent lists. SRTCORE is required for PL/I and FORTRAN programs to avoid storage overlay problems. SRTCORE is optional for COBOL and Assembler programs.

SRTOPEN *Opens a sort.* SRTOPEN causes SSRAM to use the information in the parameter list, as well as the key definition table, return code table and duplicate record processing table (when used) to initialize a sort.

SRTFILL *Passes a record to a sort.* SSRAM uses the information in the parameter list to determine the sort to which the record should be moved.

SRTGETR *Retrieves a record from a sort.* The first SRTGETR call tells the sort to merge the sorted strings and moves a sorted record to the work area. Subsequent SRTGETR calls move a sorted record to the work area of the invoking program.

SRTCLSE *Ends a sort.* The CLOSE call terminates the sort for the specified parameter list.

Parameter List Overview

A unique parameter list is required for each sort that is initiated through SSRAM. The application program stores information into this area, which SSRAM uses to run the sort. SSRAM then sets a flag in the parameter list, which can be tested by the application program. The list must be fullword aligned. The fields of this list are described here; language-specific details are in later sections.

file id This identifies the SSRAM work file name, as specified on the DLBL job control statement. The work file must be on DASD, not on tape.

logical unit number This identifies the LUN for the work file, as specified in the SYSnnn values on the ASSGN and EXTENT job control statements. nnn must be between 000 and 221.

storage This sets the amount of storage that SSRAM will use for this sort. At least 64K is required; increased storage improves sort performance.

record length This specifies the record length (maximum record length for variable-length records).

end-of-file flag This is used by SSRAM to return a flag indicating when end-of-file has been reached.

key definition table This specifies the sort keys and the sorting order.

The Key Definition Table

Each parameter list contains a key definition table. The key definition table defines the fields or sort keys by which records will be sorted. The key definition table also determines the order, ascending or descending, in which the sort keys will be sorted.

The key definition table can be 256 characters long. Following is the format for the key definition table.

[V,] (p_1,l_1,f_1,o_1,p_2,l_2,f_2,o_2...p_n,l_n,f_n,o_n)

Figure 397. Key Definition Table Format

V V is required when the record is a variable-length record.

Variable-length records must begin with a standard 4-byte RDW (record descriptor word). The first two bytes of the RDW must contain the record length in binary format. The second two bytes of the RDW are ignored by SSRAM but, nonetheless, should contain binary zeros. If you wish to sort on record length, you may sort on the first two bytes. The first data position is byte 5.

When sorting variable-length records you may enter either the average record length or the maximum record length in the parameter table. However, you must enter the maximum record length when the key definition table contains either signed binary or floating-point data type.

p This is the starting position of the key field. The maximum value allowed is 4092, where 1 is the first position in the record.

l This is the length of the key field in bytes. The maximum value varies depending on the type of data. Following are the maximum values allowed for each data type.

- The maximum value for alphanumeric data is 256.
- The maximum value for packed decimal data is 16.
- The *valid* values for floating-point data are 4 and 8.
- The maximum value for unsigned binary data is 4092.
- The maximum value for fixed integer data is 256.
- The maximum value for zoned decimal data is 256.

f This is the data format code of the sort key. See the following table for allowed codes.

Code	Type of Data
AC	EBCDIC characters are translated to their ASCII equivalents
ASL	Leading separate sign (ASCII + or -)
AST	Trailing separate sign (ASCII + or -)
BI or B	Unsigned binary
CH or C	Alphanumeric EBCDIC characters
CLO	Leading overpunch sign
CSF	Floating sign
CSL	Leading separate sign
CST	Trailing separate sign
CTO	Zoned decimal, trailing overpunch
FI	Signed fixed point
FL	Signed floating point
FS	Floating sign
LS	Leading separate sign
OL	Leading overpunch sign
PD or P	Signed packed decimal
PD0	Packed decimal, first digit and trailing sign ignored
TS	Trailing separate sign
ZD or Z	Zoned decimal, trailing overpunch
The following 2-digit formats are treated as 4-digit years by CENTWIN (century window) processing	
Y2B	Binary 2-digit, 1 byte year
Y2C	Character 2-digit year
Y2D	Packed decimal 2-digit, 1 byte year
Y2P	Packed decimal 2-digit, 2 byte year
Y2S	Character or zoned decimal 2-digit, 2-byte year
Y2T Y2V Y2W Y2X Y2Y	Full date formats; see Table 37 on page 2.228
Y2Z	Character 2-digit year

Table 51. Data Format Codes

For details on each data type, see the Format Code Chart in " Valid Formats for Control Fields" on page 2.228.

o This specifies the order in which the records will be sorted. Enter A for ascending order. Enter D for descending order.

Sample Key Definition Table

The following figure shows two ways to show the same key definition table.

```
(1,23,C,A,36,2,B,D,32,4,P,A)
or
(1,23,CH,A,36,2,BI,D,32,4,PD,A)
```

Figure 398. Sample of Equivalent Key Definition Tables

In the preceding example, the record is sorted in ascending order on the 23-character field (alphanumeric), which starts in position 1. The minor sort keys are the 2-byte, binary data starting in position 36, which is sorted in descending order and a 4-byte packed-decimal field, which starts in position 32.

Return Code Table Overview

In addition to the required parameter list, you have the option of using a return code table for each sort. The advantage of using a return code table is that it offers greater control when an error occurs. Typically, when SSRAM encounters an error, it cancels the program.

If SSRAM is passed a return code table during the OPEN call, SSRAM updates the return code table after each subsequent call. It is recommended that the program check the return code table each time it is updated by SSRAM. The program can take appropriate action based on SSRAM's response.

The return code table contains binary values and must be fullword aligned. The values in the table are in binary format.

DISPLACEMENT	TABLE ENTRY
+0	**return code (halfword)**
+2	**error code (halfword)**
+4	**FILL record count (fullword)**
+8	**GET record count (fullword)**

Table 52. Return Code Table

return codes Following are the possible return codes:

0 The last call ran successfully.

4 There is an error in the parameter list.

8 SSRAM detected an end-of-file condition as a result of the last GET call. Check the error code.

12 The CLOSE call completed successfully.

16 An error occurred during the last call. Check the error code.

error code This is the binary representation of the error message number.

FILL record count This counts the records passed as a result of FILL calls. SSRAM updates the record count when the work file is closed or if an error occurs.

GET record count This counts the records passed as a result of GET calls. SSRAM updates the record count when the work file is closed or if an error occurs.

Refer to Figure 400 on page 10.9, Figure 410 on page 10.14, Figure 420 on page 10.18, or Figure 429 on page 10.23 for the format of the return code table for each respective language.

Duplicate Record Processing Overview

In addition to the required parameter list, you have the option of using a duplicate record code for a sort. The advantage of using a duplicate record code is that it offers the option to delete records that have duplicate keys during sort execution. The other fields in the duplicate key records can be unequal.

To delete duplicate key records, enter the value 8 in the variable for duplicate record processing specified in the SRTOPEN parameter list. To retain all records, enter the value 0; this is the default.

If you are deleting duplicate keyed records and the records are variable-length records, be sure to enter the maximum record length in the parameter table. If you use the average record length instead of the maximum record length, the sort may terminate and issue error message WER123A SUM FIELD BEYOND RECORD.

ASSEMBLER Parameter List and Return Code Table

Each sort that you initiate through SSRAM must have a unique parameter list. The parameter list must be fullword aligned. Figure 399 shows the format for an Assembler parameter list.

```
PARMLIST  DS  0F                          Fullword alignment required
FILEID    DC  CL5'filename'               File name
LOGUNIT   DC  CL3'logical unit #'         Logical Unit Number
STORAGE   DC  F'storage'                  Storage Size
RECLEN    DC  H'record length'            Record Length
EOFFLAG   DC  CL2'XX'                     End-of-file indicator
KEYDFTBL  DC  C'key definition table'     Sort keys
```

Figure 399. Assembler Parameter List

Following is the description of each of the required parameters.

FILEID Enter the 1-5 character filename for the SSRAM disk work area as specified on the DLBL job control statement.

LOGUNIT Use this to set the logical unit number (LUN) for the DASD work file. This value corresponds to the SYSnnn value on both the ASSGN and EXTENT control statements for the work file in the JCL. The LUN must be a 3-digit number between 000 and 221.

STORAGE Use this to set the amount of main storage that SSRAM will use for this sort task. The minimum amount of storage required by SSRAM for each sort is 64K. However, SSRAM will accept a minimum value of 10K. If a value less than 64K is specified, SSRAM automatically uses 64K instead. Increasing storage improves performance.

RECLEN For fixed-length records, enter the actual record length. For variable-length records, enter the maximum record length.

EOFFLAG SSRAM sets this to 99 when EOF occurs on the SSRAM work file.

KEYDFTBL Enter the key definition table.

The following figure shows the format for an Assembler return code table.

```
RCODETBL  DS  0F        FULLWORD ALIGNMENT REQUIRED
RTNCODE   DC  H'0'      RETURN CODE
ERRCODE   DC  H'0'      ERROR CODE
FILLCNT   DC  F'0'      FILL RECORD COUNT
GETCNT    DC  F'0'      GET RECORD COUNT
```

Figure 400. Assembler Return Code Table

ASSEMBLER Calls

SRTCORE

```
CALL  SRTCORE,([starea,]stlength[,sys,usr,ops,str,pfx,dsm,ccw,centwin])
```

Figure 401. SRTCORE Format

The optional SRTCORE call defines a storage length; it may also define a storage area and century window. Only one SRTCORE call may be defined in the application program and it must precede all other SSRAM calls in the program.

starea Enter the variable name of the storage area within the program which will be used by SSRAM.

stlength Enter the variable name of the storage length, stored as a fullword value. A minimum of 64K of storage is required for each sort.

When **starea** is not specified, SSRAM will be loaded at the end of the last program phase loaded.

When **starea** is specified within the problem program area, a placeholder value of X'FFFFFFFF' may be used as the length parameter to specify that storage will be allocated from the **starea** location to the end of the problem program area.

sys, **usr**, **ops**, **str**, **pfx**, **dsm**, and **ccw** are optional positional parameters which are accepted but ignored. **centwin** is an optional positional parameter which specifies the century window to which 2-digit year data belongs when being processed by the sort. For details on using **centwin**, see “ CENTWIN Parameter (Optional)” on page 2.101.

If **centwin** is specified, the preceding 7 parameters must be specified using fullword placeholders. The value X’FFFFFFFF’ may be used for each. A value must also be specified for **starea**.

Following are examples of SRTCORE usage:

```
CALL SRTCORE,(AREA,LENGTH)
CALL SRTCORE,(AREA,DUMMYLEN)  Use space to end of problem program
*                                 area (AREA must not be in
*                                 GETVIS storage)
CALL SRTCORE,(LENGTH)         Load at end of last phase
CALL SRTCORE,(AREA,LENGTH,FILL,FILL,FILL,FILL,FILL,FILL,FILL,WINDOW)
*                             Century window specified
LENGTH   DC  F'128000'        Length of storage area in bytes
DUMMYLEN DC  F'-1'            Special positional value for "stlength"
AREA     DS  XL128000         Storage where SSRAM will be loaded
WINDOW   DC  F'2007'
FILL     DC  F'-1'
```

Figure 402. Examples of SRTCORE Usage

SRTOPEN

```
CALL SRTOPEN,(parmlist[,rcodetbl[,duprecps[,userexit]]])
```

Figure 403. SRTOPEN Format

The SRTOPEN call starts a SSRAM sort.

parmlist Enter the variable name of the parameter list. The parameter list is required.

rcodetbl Enter the variable name of the return code table.

duprecps Enter the variable name of the duplicate record processing field. This is a fullword binary value. The duplicate record processing field is optional.

userexit This function is not supported. However, if a variable name is entered, SSRAM will ignore the value and continue processing normally.

Because the variables are positional, if you wish to specify a duplicate record processing table and a user exit, you must enter a variable name for the return code table even if you are not using return codes. In this case, define a placeholder return code table as follows:

```
rcodetbl  DC   F'-1'
```

Figure 404. Placeholder Return Code Table

SRTFILL

```
CALL SRTFILL,(parmlist,record-area)
```

Figure 405. SRTFILL Format

The SRTFILL call tells SSRAM to pass a record from the specified record area to the sort defined by the specified parameter list.

parmlist Enter the variable name of the parameter list of the sort to which you wish to pass a record.

record-area Enter the variable name of the area that contains the record to be passed to the sort.

SRTGETR

```
CALL SRTGETR,(parmlist,record-area)
```

Figure 406. SRTGETR Format

The SRTGETR call retrieves a record from the sort defined by the parameter list and puts it in the specified record-area. If SSRAM detects an end-of-file condition, SSRAM sets EOFFLAG to 99. If the program expects a return code, SSRAM will return code 8 at EOF.

parmlist This is the variable name of the parameter list that identifies the sort from which the record is being retrieved.

record-area This variable name identifies the name of the record area to which the record is to be moved.

SRTCLSE

```
CALL SRTCLSE,(parmlist[,rcodetbl])
```

Figure 407. SRTCLSE Format

Use the SRTCLSE call to terminate sort processing for an individual sort. If return codes have been requested, SSRAM supplies the record count and return code 12 if the close was successful.

parmlist This is the variable name of the parameter list of the sort you wish to close.

rcodetbl This is the variable name of the return code table, which is updated and returned to the program. This is optional.

COBOL Parameter List and Return Code Table

Each sort that you initiate through SSRAM must be accompanied by a unique parameter list. Since the parameter list must be fullword aligned, the parameter list must be defined as SYNCHRONIZED. Figure 408 shows the format for a COBOL parameter list.

```
01   SRTTBLE    SYNCHRONIZED.
   02  FILE-ID       PICTURE X(5)   VALUE  'filename'.
   02  LOG-UNIT      PICTURE X(3)   VALUE  'Logical unit number'.
   02  STORAGE       PICTURE S9(6)  VALUE  main storage size COMP.
   02  REC-LENGTH    PICTURE S9(4)  VALUE  record-length COMP.
   02  EOF-FLAG      PICTURE XX     VALUE  SPACES.
   02  KEYDEF-TBL    PICTURE X(nn)  VALUE  'key definition table'.
```

Figure 408. COBOL Parameter List

FILE-ID Enter the 1-5 character filename for the SSRAM disk work area as specified on the DLBL job control statement.

LOG-UNIT Use this to set the logical unit number (LUN) for the DASD work file. This value corresponds to the SYSnnn value on both the ASSGN and EXTENT control statements for the work file in the JCL. The LUN must be a 3-digit number between 000 and 221.

STORAGE Use this to set the amount of main storage that SSRAM will use for this sort task. The minimum amount of storage required by SSRAM for each sort is 64K. However, SSRAM will accept a minimum value of 10K. If a value less than 64K is specified, SSRAM automatically uses 64K instead. Increasing storage improves performance.

REC-LENGTH For fixed-length records, enter the actual record length. For variable-length records, enter the maximum record length.

EOF-FLAG SSRAM sets this to 99 when EOF occurs on the SSRAM work file.

KEYDEF-TBL Enter the key definition table.

There are special considerations for sorting variable-length records from a COBOL program. Since COBOL does not typically use the RDW, you must ensure that an RDW that meets the previously described specifications is included as part of the record. This can be done in the record-area definition in WORKING STORAGE. The following figure shows a sample record-area definition that includes the required RDW. REC-LEN must be adjusted to include the record length and the length of the RDW for each variable-length record. The record should be moved to USER-DATA prior to being passed to the sort.

```
01  RECORD-AREA.
    05  REC-LEN   PIC S9(4)  COMPUTATIONAL.
    05  FILLER    PIC XX.
    05  USER-DATA PIC X(...).
```

Figure 409. Sample COBOL Record Area Definition

Figure 410 shows the format for a COBOL return code table. Since the return code table must be fullword aligned, the return code table must be defined as SYNCHRONIZED.

```
01 RETURN-CODE-TABLE       SYNCHRONIZED.
   05  RETURN-CODE  PIC     S9(4) COMPUTATIONAL  VALUE  ZERO.
   05  ERROR-CODE   PIC     S9(4) COMPUTATIONAL  VALUE  ZERO.
   05  FILL-COUNT   PIC     S9(6) COMPUTATIONAL  VALUE  ZERO.
   05  GET-COUNT    PIC     S9(6) COMPUTATIONAL  VALUE  ZERO.
```

Figure 410. COBOL Return Code Table

COBOL Calls

SRTCORE

```
CALL 'SRTCORE' USING [starea,]stlength,[sys,usr,ops,str,pfx,dsm,ccw,centwin].
```

Figure 411. SRTCORE Format

The optional SRTCORE call defines a storage area and storage length; it may also define a century window. Only one SRTCORE call may be defined in the application program and it must precede all other SSRAM calls in the program.

starea Enter the variable name of the storage area within the program which will be used by SSRAM.

stlength Enter the variable name of the storage length. A minimum of 64K of storage is required for each sort.

When **starea** is not specified, SSRAM will be loaded at the end of the last program phase loaded.

sys, **usr**, **ops**, **str**, **pfx**, **dsm**, and **ccw** are optional positional parameters which are accepted but ignored. **centwin** is an optional positional parameter which specifies the century window to which 2-digit year data belongs when being processed by the sort. For details on using **centwin**, see " CENTWIN Parameter (Optional)" on page 2.101.

If **centwin** is specified, the preceding 7 parameters must be specified using placeholder variables. The value PLACEHOLDER PIC X(-1) may be used for each.

Following is an example of SRTCORE usage:

```
CALL 'SRTCORE' USING STORAREA,STORLEN.

WORKING-STORAGE SECTION.
01 STORAREA     PIC    X(128000).
01 STORLEN      PIC    S9(6) COMPUTATIONAL VALUE +128000.
```

Figure 412. Example of SRTCORE Usage

SRTOPEN

```
CALL 'SRTOPEN' USING parmlist[,rcodetbl[,duprecps]].
```

Figure 413. SRTOPEN Format

The SRTOPEN call starts a SSRAM sort.

parmlist Enter the variable name of the parameter list. The parameter list is required.

rcodetbl Enter the variable name of the return code table. The return code table is optional.

duprecps Enter the code that indicates how duplicate records are processed. 0 indicates that duplicate records are to be retained; this is the default. 8 tells SSRAM to delete records with duplicate keys. The following figure shows how to specify deletion of records with duplicate keys.

```
DUPRECPS   PIC S9(6) VALUE +8 COMPUTATIONAL.
```

Figure 414. Format to Specify Deletion of Records with Duplicate Keys.

Because the variables are positional, if you wish to specify duplicate record processing, you must enter a variable name for the return code table even if you are not using return codes. In this case, define a placeholder return code table as follows:

```
rcodetbl   PIC S9(6) VALUE HIGH-VALUES.
```

Figure 415. Placeholder Return Code Table

SRTFILL

```
CALL 'SRTFILL' USING parmlist,record-area.
```

Figure 416. SRTFILL Format

The SRTFILL call tells SSRAM to pass a record from the specified record area to the sort defined by the specified parameter list.

parmlist Enter the variable name of the parameter list of the sort to which you wish to pass a record.

record-area Enter the variable name of the area that contains the record to be passed.

SRTGETR

```
CALL 'SRTGETR' USING parmlist,record-area.
```

Figure 417. SRTGETR Format

The SRTGETR call retrieves a record from the sort defined by the parameter list and puts it in the specified record-area. If SSRAM detects an end-of-file condition, SSRAM sets EOF-FLAG to 99. If the program expects a return code, SSRAM will return code 8 at EOF.

parmlist This is the variable name of the parameter list that identifies the sort from which the record is being retrieved.

record-area This variable name identifies the name of the area to which the record is to be moved.

SRTCLSE

```
CALL 'SRTCLSE' USING parmlist[,rcodetbl].
```

Figure 418. SRTCLSE Format

Use the SRTCLSE call to terminate sort processing for an individual sort. If return codes have been requested, SSRAM supplies the record count and return code 12 if the close was successful.

parmlist This is the variable name of the parameter list of the sort you wish to close.

rcodetbl This is the variable name of the return code table, which is updated and returned to the program. This is optional.

Note: With COBOL (D), ENTER LINKAGE and ENTER COBOL must be used with each call.

FORTRAN Parameter List and Return Code Table

Each sort that you initiate through SSRAM must be accompanied by a unique parameter list. The parameter list in FORTRAN is defined using the DIMENSION and the DATA statements. Figure 419 shows the format for a FORTRAN parameter list.

```
DIMENSION parmlist(n),data-area(nnnnn)
DATA parmlist / 5Hbyte1-5,3Hbyte6-8,byte9-12,byte13-16,sH(byte17-n)
```

Figure 419. FORTRAN Parameter List

Refer to the IBM VS FORTRAN documentation for a full description of the DIMENSION and DATA statements syntax. Following is a description of each of the required fields.

parmlist(n) This identifies the variable name of the parmlist defined in the DATA statement. n is the number of words in the accompanying DATA statement; its value depends on the size of the key definition table.

Byte 1-5 Use this to specify the 5-character filename of the SSRAM disk work area as specified on the DLBL job control statement.

Byte 6-8 Use this to identify the Logical Unit Number of the device that contains the work file. This value corresponds to the SYSnnn value on both the ASSGN and EXTENT control statements for the work file in the JCL. The LUN must be a 3-digit number between 000 and 221.

Byte 9-12 Use this to set the amount of main storage that SSRAM will use for a sort task. The minimum amount of storage required by SSRAM for each sort is 64K. However, SSRAM will accept a minimum value of 10K. If a value less than 64K is specified, SSRAM automatically uses 64K instead. Increasing storage improves performance.

Byte 13-16 Use this to specify the record length. For variable-length records, specify the maximum record length.

Byte 17-n Use this for the key definition table. It must be enclosed within paretheses. *s* is the size of the table in bytes.

Figure 420 shows the format for a FORTRAN return code table.

```
DIMENSION     rcodetbl (3)
```

Figure 420. FORTRAN Return Code Table

The return code table contains the following:

Byte 1-2 return code

Byte 3-4 error code

Byte 5-8 FILL record count

Byte 9-12 GET record count

FORTRAN Calls

SSCOR

```
CALL SSCOR (iarea,ilength[,isys,iusr,iops,istr,ipfx,idsm,iccw,icentwin])
```

Figure 421. SSCOR Format

The required SSCOR call defines a storage area and storage length. Optionally, it may also define a century window. Only one SSCOR call may be defined in the application program and it must precede all other SSRAM calls in the program.

iarea Enter the variable name of the storage area within the program which will be used by SSRAM.

ilength Enter the variable name of the storage length. A minimum of 64K of storage is required for each sort.

isys, **iusr**, **iops**, **istr**, **ipfx**, **idsm**, and **iccw** are optional positional parameters which are accepted but ignored. **icentwin** is an optional positional parameter which specifies the century window to which 2-digit year data belongs when being processed by the sort. For details on using **icentwin**, see " CENTWIN Parameter (Optional)" on page 2.101.

If **icentwin** is specified, the preceding 7 parameters must be specified using placeholder variables. The value DATA IHOLDER/-1/ may be used for each.

Following is an example of SSCOR usage:

```
DIMENSION IAREA(32768)
DATA ILENGTH/131072/
CALL SSCOR (IAREA,ILENGTH)
```

Figure 422. Example of SSCOR Usage

SSOPN

```
CALL SSOPN (parmlist[,rcodetbl[,duprecps]])
```

Figure 423. SSOPN Format

The SSOPN call starts a SSRAM sort.

parmlist Enter the variable name of the parameter list. The parameter list is required.

rcodetbl Enter the variable name of the return code table.

duprecps Enter the variable name of the duplicate record processing field. The duplicate record processing field is optional.

Because the variables are positional, if you wish to specify a duplicate record processing field, you must enter a variable name for the return code table even if you are not using one. In this case, define a placeholder return code table as follows:

```
DATA rcodetbl  /-1/
```

Figure 424. Placeholder Return Code Table

SSFIL

```
CALL SSFIL (parmlist,irec)
```

Figure 425. SSFIL Format

The SSFIL call tells SSRAM to pass a record from the specified integer array to the sort defined by the specified parameter list.

parmlist Enter the variable name of the parameter list of the sort to which you wish to pass a record.

irec Enter the variable name of the integer array that contains the record to be sorted.

SSGET

```
CALL SSGET (parmlist,irec,&stmt)
```

Figure 426. SSGET Format

The SSGET call retrieves a record from the sort defined by the parameter list and puts it into a specified integer array. If SSRAM detects an end-of-file condition, SSRAM passes control to the statement specified by &stmt. If the program expects a return code, SSRAM will return a code of 8 at EOF.

parmlist This is the variable name of the parameter list that identifies the sort from which a record is to be retrieved.

irec This variable name identifies the integer array into which the record is to be moved.

&stmt This is the statement number of the statement to which SSRAM passes control when an end-of-file condition occurs.

SSCLS

```
CALL SSCLS (parmlist[,rcodetbl])
```

Figure 427. SSCLS Format

Use the SSCLS call to terminate sort processing for an individual sort. If return codes have been requested, SSRAM supplies the FILL record count, the GET record count, and return code 12 if the close was successful.

parmlist This is the variable name of the parameter list of the sort you wish to close.

rcodetbl This is the variable name of the return code table.

PL/I Parameter List and Return Code Table

Each sort that you initiate through SSRAM must be accompanied by a unique parameter list. Since the parameter list must be fullword aligned, the parameter list must be defined as a STATIC ALIGNED structure. Do not use other structures such as AUTOMATIC for SSRAM storage areas. Figure 428 shows the format for a PL/I parameter list.

```
DECLARE 1 SORTTAB STATIC ALIGNED,
 2 FILEID     CHAR (5)          INITIAL ('file name'),
 2 LOGUNIT    CHAR (3)          INITIAL ('logical unit #'),
 2 STORAGE    BINARY FIXED (31) INITIAL (storage size),
 2 RECLEN     BINARY FIXED (15) INITIAL (record length),
 2 EOFFLAG    CHAR (2)          INITIAL ('XX'),
 2 KEYDFTBL   CHAR (nn)         INITIAL ('key definition table');
```

Figure 428. PL/I Parameter List.

Following is the description of each of the required parameters.

FILEID Enter the 5-character filename for the SSRAM disk work area as specified on the DLBL job control statement.

LOGUNIT Use this to set the logical unit number (LUN) for the DASD work file. This value corresponds to the SYSnnn value on both the ASSGN and EXTENT control statements for the work file in the JCL. The LUN must be a 3-digit number between 000 and 221.

STORAGE Use this to set the amount of main storage that SSRAM will use for a sort task. The minimum amount of storage required by SSRAM for each sort is 64K. However, SSRAM will accept a minimum value of 10K. If a value less than 64K is specified, SSRAM automatically uses 64K instead. Increasing storage improves performance.

RECLEN For fixed-length records, enter the actual record length. For variable-length records, enter the maximum record length.

EOFFLAG SSRAM sets this to 99 when EOF occurs on the SSRAM work file.

KEYDFTBL Enter the key definition table. It must be enclosed within parentheses.

Figure 429 shows the format for a PL/I return code table. Since the return code table must be fullword aligned, the return code table must be defined as a STATIC ALIGNED structure.

```
DECLARE 1 RETURN_CODE_TABLE   STATIC ALIGNED,
 2 RETURN_CODE  BINARY FIXED(15) INIT(0),
 2 ERROR_CODE   BINARY FIXED(15) INIT(0),
 2 FILL_COUNT   BINARY FIXED(31) INIT(0),
 2 GET_COUNT    BINARY FIXED(31) INIT(0);
```

Figure 429. PL/I Return Code Table.

PL/I Calls

SRTCORE

```
CALL SRTCORE (starea,stlength[,sys,usr,ops,str,pfx,dsm,ccw,centwin]);
```

Figure 430. SRTCORE Format

The required SRTCORE call defines a storage area and storage length. Optionally, it may also define a century window. Only one SRTCORE call may be defined in the application program and it must precede all other SSRAM calls in the program.

starea Enter the variable name of the storage area within the program which will be used by SSRAM.

stlength Enter the variable name of the storage length. A minimum of 64K of storage is required for each sort.

sys, **usr**, **ops**, **str**, **pfx**, **dsm**, and **ccw** are optional positional parameters which are accepted but ignored. **centwin** is an optional positional parameter which specifies the century window to which 2-digit year data belongs when being processed by the sort. For details on using **centwin**, see " CENTWIN Parameter (Optional)" on page 2.101.

If centwin is specified, the preceding 7 parameters must be specified using placeholder variables. The value PLACEHOLDER CHAR(-1) may be used for each.

Following is an example of SRTCORE usage:

```
    DCL1 STOR_AREA  STATIC ALIGNED,
       2 SRAM_AREA  CHAR(256000);
    DCL  STOR_LEN   BIN FIXED(31) INIT(256000);
```

Figure 431. Example of SRTCORE Usage

SRTOPEN

```
CALL  SRTOPEN (parmlist[,rcodetbl[,duprecps]]);
```

Figure 432. SRTOPEN Format

The SRTOPEN call starts a SSRAM sort.

parmlist Enter the variable name of the parameter list. The parameter list is required.

rcodetbl Enter the variable name of the return code table.

duprecps Enter the variable name of the duplicate record processing field. The duplicate record processing field is optional.

Because the variables are positional, if you wish to specify a duplicate record processing field, you must enter a variable name for the return code table even if you are not using one. In this case, define a placeholder return code table as in the following example:

```
DCL rcodetbl  FIXED BIN(31) INIT(-1);
```

Figure 433. Placeholder Return Code Table

SRTFILL

```
CALL SRTFILL (parmlist,recptr);
```

Figure 434. SRTFILL Format

The SRTFILL call tells SSRAM to pass a record from the specified record area to the sort defined by the specified parameter list.

parmlist Enter the variable name of the parameter list of the sort to which you wish to pass a record.

recptr Enter the variable name of an address constant pointer to a one-dimensional structure or elementary item that contains the input record to be sorted.

SRTGETR

```
CALL SRTGETR (parmlist,recptr);
```

Figure 435. SRTGETR Format

The SRTGETR call retrieves a record from the sort defined by the parameter list and puts it in the area specified by the address constant pointer. If SSRAM detects an end-of-file condition, SSRAM sets EOFFLAG to 99. If the program expects a return code, SSRAM will return a code of 8 at EOF.

parmlist This is the variable name of the parameter list that identifies the sort from which the record is being retrieved.

recptr This variable name identifies the name of the address constant pointer that specifies the area to which the record is to be moved.

SRTCLSE

```
CALL SRTCLSE (parmlist[,rcodetbl]);
```

Figure 436. SRTCLSE Format

The SRTCLSE call terminates sort processing for an individual sort. If return codes have been requested, SSRAM supplies the FILL record count, the GET record count, and the return code 12 if the close was successful.

parmlist This is the variable name of the parameter list of the sort you wish to close.

rcodetbl This is the variable name of the return code table, which is updated with the appropriate return code and record counts. This is optional.

Chapter 11. The SYNCHSTO Utility Program

What is SYNCHSTO?

SYNCHSTO is a separate program that is used to determine information about variable-length record files. The program uses SyncSort to scan a variable-length record file and provides information about the records in the file, which then can be used to run more efficient sorts of that file.

SYNCHSTO will report the:

- minimum and maximum record lengths in the file;
- average record length;
- total number of records in the file;
- total number of bytes in the file;
- measured l_5 value (most common record length) for the file;
- recommended l_6 value (average work space per record) for the file; and
- recommended l_7 value (optimum segment length) for the file.

SYNCHSTO will also produce a histogram that displays the distribution of the lengths of the records in the file.

The l_5, l_6, and l_7 values can be used for the LENGTH parameter on the RECORD control statement for subsequent sorts using the file to help improve the performance of those sorts. Since the l_7 value depends on the record length and key position after INREC and E15 processing, the value calculated by SYNCHSTO is an estimate assuming a specified (or default) key position. If a sort application reformats the records using an E15 or INREC, a more accurate value can be calculated for that application by using the VL5L6L7 PARM with that sort (see "VL5L6L7" on page 6.8).

The output from SYNCHSTO is written to SYSLST, which must be ASSGNed to a print device.

Control Parameters for SYNCHSTO

The parameter statement (or statements) describes the input file and controls the characteristics of the SYNCHSTO output. Parameter statements are read into SYNCHSTO through SYSIPT. A parameter statement contains parameters and their associated values (if any), which may be specified in any order.

The following rules are used for parameter statements:

- The first parameter on a statement begins in column 1 and extends up to column 71.
- Each parameter statement ends with a blank to end that statement, or a comma to continue the parameter list onto the next statement.
- Parameters are separated by commas, with no intervening blanks.
- A blank or a comma may not be used within a parameter.
- A continuation statement begins in column 1.

The control parameters are described below; defaults are underlined. For all numerical values, only a significant number of digits is needed (i.e. "nnnnn" can be replaced by "80"; "00080" is not required). LRECL is the only parameter that is <u>required</u> in all cases.

BLKSIZE

```
BLKSIZE=nnnnn
```

Figure 437. BLKSIZE Format

This parameter specifies the block size of the input file. (It is required if the file is of type VB or VBS.)

The maximum value of nnnnn is 65528.

CLOSE

Figure 438. CLOSE Format

For tape files, this parameter indicates what action should be taken after the file has been read. CLOSE=RWD, the default, specifies that the last tape input volume will be rewound at end-of-file. CLOSE=NORWD specifies that input volumes should not be rewound at end-of-file. CLOSE=UNLD specifies that the last tape input volume should be rewound and unloaded at end-of-file.

This parameter is ignored for non-tape files.

KEYEND

Figure 439. KEYEND Format

This parameter specifies the ending position of the last sort key field that will be used when sorting this file. Remember that the first 4 bytes of the record are used by the Record Descriptor Word, so the first byte of the data portion of the record is byte 5.

LRECL (required)

Figure 440. LRECL Format

This parameter specifies the maximum record length of the variable-length records in the file. Remember to include 4 bytes for the Record Descriptor Word.

The maximum value of “nnnnn” is 65524.

NRECS

Figure 441. NRECS Format

If specified, this parameter specifies the number of records to be read from the file. If not specified, the entire file is read. The file characteristics measured and calculated by SYNCHSTO will be based on only the first "nnnnnnnnnn" records of the file. (If the record lengths are randomly distributed in a very large file, this can provide a useful estimate that avoids the long processing time required for reading the entire file.)

The maximum value of "nnnnnnnnnn" is 2 gigabytes - 1 (i.e. 2147483647).

OPEN

Figure 442. OPEN Format

For tape files, this parameter indicates what action should be taken when the file is opened.

OPEN=RWD, the default, specifies that the first volume of the tape input file is to be rewound when the file is opened. OPEN=NORWD specifies that the first volume of the tape input file is not to be rewound when the file is opened.

This parameter is ignored for non-tape files.

SORTNAME

Figure 443. SORTNAME Format

This parameter, if specified, indicates the name of SyncSort as installed on your system to be used by SYNCHSTO.

Specify the "sortname" value as 1 to 8 alphanumeric characters. The first character must be alphabetic.

SPAN

Figure 444. SPAN Format

This parameter indicates that the input file may contain spanned records.

VOLUME

```
VOLUME=nnn
```

Figure 445. VOLUME Format

This parameter indicates the number of volumes for an unlabeled input tape file.

This parameter is ignored for disk input files.

VSAM

```
VSAM
```

Figure 446. VSAM Format

Specify this parameter when the input file is a VSAM file.

WIDTH

```
WIDTH=nnnnn
```

Figure 447. WIDTH Format

This parameter indicates the range between the minimum and maximum record lengths in each group of the SYNCHSTO histogram display. The number specified must be a multiple of 4.

If not specified, SYNCHSTO will dynamically calculate the value of WIDTH to use for the display based on the minimum and maximum record lengths of the records read from the file.

The maximum value of "nnnnn" is 65524.

Job Control Language

Use the filename SORTIN1 and symbolic unit name SYS002 to define the input file on the ASSGN, DLBL or TLBL, and EXTENT JCL statements, followed by one or more control statements. A sample SYNCHSTO job stream is shown below.

```
          // JOB TESTHSTO
          // ASSGN SYS001,SYSLST
1.        // ASSGN SYS002,DISK,VOL=mydisk,SHR
2.        // DLBL SORTIN1,'VARIABLE.DATA.FILE',0,SD
1.        // EXTENT SYS002,mydisk
          // EXEC SYNCHSTO,SIZE=128K
3.        LRECL=2900,KEYEND=75
4.        WIDTH=100,
4.        NRECS=5000
          /*
          /&
```

Figure 448. Sample SYNCHSTO Job Stream

Notes:

1. Symbolic unit name SYS002 is used.

2. Filename SORTIN1 is used.

3. The required LRECL parameter is on this control statement.

4. The second control statement ends with a comma, indicating that it is continued on the next control statement. Since multiple control statements are allowed, and a parameter cannot be “split” in the middle by a comma, there is no difference between specifying control statements with or without continuations.

Sample SYNCHSTO Output

Figure 449 shows the output resulting from running SYNCHSTO with the JCL example shown above.

```
1.
                 * * * * * * * * * * * * *   SYNCHSTO PARAMETER STATEMENTS * * * * * * * * * * * * *
                        LRECL=2900,KEYEND=75
                        WIDTH=100,
                        NRECS=5000
2.
  SHS001I  NUMBER OF RECORDS . . . . . . .         5000
  SHS002I  TOTAL LENGTH OF ALL RECORDS . .      8169820
  SHS003I  AVERAGE RECORD LENGTH . . . . .         1633
  SHS004I  KEY END POSITION  . . . . . . .           75
  SHS005I  HISTOGRAM LINE WIDTH  . . . . .          100
  SHS006I  LONGEST RECORD  . . . . . . . .         2900
  SHS007I  SHORTEST RECORD . . . . . . . .          402
  SHS008I  MOST COMMON RECORD LENGTH (L5).         1100
  SHS009I  AVG. SPACE PER RECORD (L6). . .         1651
  SHS010I  RECOMMENDED SEGMENT SIZE (L7) .          220
3.
 RECORD                                                                                                        RECORD LENGTH
  COUNT                                                                                                            MIN    MAX
    207 *********************************************************************                                      400    499
    205 ********************************************************************                                       500    599
    212 **********************************************************************                                     600    699
    193 ****************************************************************                                           700    799
    214 ***********************************************************************                                    800    899
    179 ***********************************************************                                                900    999
    199 ******************************************************************                                        1000   1099
    224 **************************************************************************                                1100   1199
    210 **********************************************************************                                    1200   1299
    201 *******************************************************************                                       1300   1399
    228 ****************************************************************************                              1400   1499
    200 ******************************************************************                                        1500   1599
    195 *****************************************************************                                         1600   1699
    193 ****************************************************************                                          1700   1799
    200 ******************************************************************                                        1800   1899
    196 *****************************************************************                                         1900   1999
    192 ****************************************************************                                          2000   2099
    179 ***********************************************************                                               2100   2199
    195 *****************************************************************                                         2200   2299
    207 *********************************************************************                                     2300   2399
    194 ****************************************************************                                          2400   2499
    188 **************************************************************                                            2500   2599
    189 ***************************************************************                                           2600   2699
    188 **************************************************************                                            2700   2799
    210 **********************************************************************                                    2800   2899
      2                                                                                                           2900   2903
```

Figure 449. Sample SYNCHSTO Output

1. The parameters read by SYNCHSTO are printed at the top of the report. If there are any control statement errors, error messages are printed here.

2. The SYNCHSTO informational messages describe the characteristics of the records that have been read for the file, and the recommended values of l_6 and l_7 that have been calculated. These are described in “SYNCHSTO Messages” on page 12.37.

3. The histogram gives a graphic display of the distribution of the lengths of the records in the file. RECORD COUNT gives the number of records falling within the minimum and maximum numbers shown under RECORD LENGTH. The range is the line WIDTH value that has been specified and displayed. The asterisks represent the number of records in each range of record lengths.

Chapter 12. Messages

SyncSort for z/VSE Messages

This chapter identifies SyncSort for z/VSE messages. A section at the end of this chapter describes the most common errors and how to correct them.

There are three main types of messages for the sort/merge program:

1. **WERnnnA** messages generally report a critical error (WER226A, for example, is an exception). The sort/merge has terminated for the indicated reasons.

2. **WERnnnI** and **WERnnnB** messages are informational, although some of them can be taken as warnings. In any case, processing continues.

3. **WERnnnC** messages are automatically printed on the console as well as the printer unless ROUTE=LST has been specified. In that case, they will only appear on the printer.

Installation-related messages are documented in the *SyncSort for z/VSE Installation Guide*.

The messages for SyncSort sort/merge that are not installation-related are given below.

Note: These messages are available in the VSE console EXPLAIN function if your systems programmer installed SyncSort with this option.

WER002A **EXCESS CONTROL STATEMENTS**
EXPLANATION: Over 80 control statements were used.

ACTION: If excess occurred because of comments on control statement continuations, use the special comment statements for remarks. (These will not be counted in the 80 statement total).

WER003A **CONTINUATION STATEMENT ERROR FOUND**
EXPLANATION: This message is generated when an improperly specified continuation statement is encountered.

WER004A **CALCAREA REQUESTED; NO SIZE=D**
EXPLANATION: Whenever the CALCAREA parameter is included in the OPTION statement, a SIZE value must be provided in the SORT statement. The SIZE value must be a positive integer.

WER007I **CENTURY WINDOW FROM xxxx TO yyyy**
EXPLANATION: One or more Y2x data formats have been used for a SORT/MERGE control statement or an OUTREC edit field. The starting year is xxxx and the ending year is yyyy for the century window used to process the fields.

WER008A **INVALID CENTWIN/CW VALUE ON EXEC PARM**
EXPLANATION: The CENTWIN/CW PARM on the EXEC statement must be a number between 0 and 100 or a number between 1000 and 3000.
ACTION: Correct the CENTWIN/CW values in the EXEC statement PARM.

WER009A **INVALID ANSI CONTROL CHARACTER FOUND**
EXPLANATION: The ANSI control character found in the HEADER parameter of the OUTFIL control statement was invalid or illegal for the device assigned. Note that (1) SyncSort does not support X, Y, Z control characters for a card punch; and (2) even though &PAGE begins with a blank, which is a valid ANSI control character for a printer, it is not permitted to be the first field of a heading when LINES=ANSI or LINES=(ANSI,n).

WER010A **xxxxx STATEMENT MISSING**
EXPLANATION: A required SORT, MERGE, or RECORD control statement was not specified.

WER012A **SUM INCOMPATIBLE WITH COPY OR COMPARE**
EXPLANATION: The SUM or DUPKEYS control statement cannot be specified when FIELDS=COPY or FIELDS=COMPARE is specified.

WER014A **DUPLICATE PARM FOUND ON xxxx STATEMENT**
EXPLANATION: One of the allowable parameters has been specified more than once on a single control statement. The message will reflect the name of the statement containing the duplicated parameter.

WER015A **EXTRA ESDS/KSDS/RRDS ON OUTFIL STMT**
EXPLANATION: More than one of the ESDS, KSDS, or RRDS keywords was specified on the OUTFIL control statement.

WER017A **ERROR IN DISPLACEMENT/LENGTH VALUE**
EXPLANATION: The sum of the lengths of all control fields is over 4092 bytes. (If the EQUALS parameter is used on the OPTION statement, the sum of the lengths is limited to 4088 bytes.)

WER018A **CONTROL FIELD ERROR**
EXPLANATION: A value specifying the type of field format was left out of the FIELDS parameter in the SORT or MERGE statement.

WER019A **SIZE/SKIPREC ERROR**
EXPLANATION: The number of records in the input files was incorrectly specified in the SIZE parameter on the SORT statement.

WER021A **TYPE PARAMETER MISSING ON RECORD STMT**
EXPLANATION: The TYPE parameter was not included on the RECORD statement as required.

WER022A **RECORD TYPE INVALID**
EXPLANATION: An invalid value was given for the TYPE parameter on the RECORD statement. (Only F, V, D, VS, or VBS is valid.) Or, TYPE=V, VS or VBS was specified when DATA=A was specified on the INPFIL statement. Or, TYPE=D was specified when DATA=E was specified on the INPFIL statement or was the default.

WER023A **LENGTH PARAM. MISSING ON RECORD STMT**
EXPLANATION: The LENGTH parameter was not included on the RECORD control statement as required.

WER024A **ERROR IN LENGTH VALUE**
EXPLANATION: An invalid l value appeared on the RECORD statement.
EXAMPLES: $l_5 > l_1$ $l_4 > l_2$
Or, any of the following may have occurred:

1. Output was to tape and a record length of fewer than 18 bytes was defined. The system does not guarantee that it can correctly read this record back if the last block has only one record. To correct this problem, increase the minimum record length to at least 18 bytes.

2. Input was an EPIC-cataloged data set. The record length for this file stored in the EPIC catalog does not match the record length specified on the RECORD control statement.

3. Input was a VSAM file and the record length in the VSAM catalog differs from the record length on the RECORD control statement.

4. The BUFOFF parameter was specified incorrectly. The BUFOFF value can be 0-99 for input but only 0 or 4 for output. (Only 0 should be specified for fixed-length output records.) The BUFOFF parameter must not be specified for EBCDIC data.

5. The record length (l_1-l_5) for ASCII data exceeds 9,999 bytes.

6. The length specified on the RECORD statement conflicts with the length of the record after INPFIL INREC processing. Remember that the LRECL parameter of the INPFIL control statement specifies the record length *before* INPFIL INREC processing, and the LENGTH parameter of the RECORD control statement specifies the record length *after* INPFIL INREC processing.

WER025A **JOINKEYS FIELD BEYOND RECORD**
EXPLANATION: The last byte of a control field as specified on the JOINKEYS control statement was beyond the last byte of a record.

WER026A **RECORD LENGTH (L1) NOT SPECIFIED**
EXPLANATION: The l_1 value was not included in the RECORD statement as required.

WER027A **CONTROL FIELD BEYOND RECORD**
EXPLANATION: The last byte of a control field as specified was beyond the last byte of a record.

WER028I **OPTION ADDROUT PARAMETER IGNORED**
EXPLANATION: ADDROUT can not be specified on the OPTION control statement when INREC, OUTREC, SUM, or DUPKEYS has been specified. The ADDROUT specification will be ignored and processing will continue.

WER029I ****END PHASE 0 NO ERRORS DETECTED****
EXPLANATION: The sort's phase 0 completed with no errors detected.

WER030A **UNIT RECORD MAXIMUM VALUE EXCEEDED**
EXPLANATION: One or more of the unit record parameters specified on the OUTFIL statement exceeded internal restrictions.

WER031A **FOR OUTFIL EXIT, E35 IS REQUIRED**
EXPLANATION: An OUTFIL statement specifying EXIT was found, but no E35 program was included.

WER032I **OUTFIL BYPASS PARAMETER IGNORED**
EXPLANATION: The BYPASS parameter was specified on the OUTFIL

statement. Since this may be specified only on the INPFIL statement, the parameter was ignored and processing continued.

WER033A **FILESOUT CONFLICTS WITH OUTFIL FILES**

EXPLANATION: The number of output files specified in the FILESOUT parameter on the SORT or MERGE control statement differs from the number of output files specified in the FILES parameter on the OUTFIL control statement.

ACTION: Correct either the FILESOUT parameter or the FILES parameter so that both parameters specify the same number of output files.

WER034A **ENDREC < STARTREC ON IN/OUTFIL STMT**
WER034A **ENDREC < STARTREC ON XSUMFIL STMT**
WER034A **ENDREC < STARTREC ON XDUPFIL STMT**

EXPLANATION: The values specified for STARTREC and ENDREC on an INPFIL/OUTFIL, XDUPFIL, or XSUMFIL control statement are inconsistent. The ENDREC value must be larger than or equal to the STARTREC value.

WER036I **XSUMFIL IGNORED, XSUM NOT SPECIFIED**
WER036I **XDUPFIL IGNORED, XDUP NOT SPECIFIED**

EXPLANATION: The sort has found and ignored either an XSUMFIL control statement because the XSUM parameter is not present, or an XDUPFIL control statement because the XDUP parameter is not present. To activate XSUM or XDUP processing (i.e., saving the eliminated records in a separate output file), the XSUM or XDUP parameter must be specified.

WER037I **NO KEYWORDS FOUND ON xxxxx STATEMENT**

EXPLANATION: One of the SyncSort control statements, xxxxx, has been specified without any of the allowable keywords. This message will reflect the name of the appropriate control statement.

WER039A **INSUFFICIENT STORAGE**

EXPLANATION: The sort started to run, but found insufficient storage during Phase 0 or Phase 1.

ACTION: Check to see that figures are accurate if the SIZE parameter was used on the JCL EXEC statement and/or STORAGE specified on the OPTION statement. (SyncSort requires at least 64K of storage to run.) If exit programs have been preloaded, check to see that they were loaded into the highest possible positions in the partition. Increase storage.

WER040A **VSAM OR KEYED FILE NOT ON DISK**

EXPLANATION: All keyed or VSAM files must be on disk.

WER042A **UNITS ASSIGN ERR: DVC=XX, LUN=YY**
EXPLANATION: In attempting to determine the assignment for LUN=YY (where YY is the SYSNBR in hexadecimal), the sort/merge found a discrepancy. Either (1) DVC=XX was not supported, where XX is the device type in hexadecimal; or (2) LUN=YY was not assigned, UA or IGN (in this case, DVC=FF).

WER043I **EXIT IGNORED, THIS PHASE NOT MERGE**
EXPLANATION: A Phase 2 exit, which can be used only with a sort, was specified for a merge. The exit was never processed, and the sort continued.

WER044A **ILLEGAL OVERLAPPING OF DECIMAL FIELDS**
EXPLANATION: Two control fields may not share the rightmost (signed) byte of a packed or zoned decimal field unless they both end at that byte. (A signed byte can not be embedded in a decimal field.)
EXAMPLE: FIELDS=(8,2,PD,A,9,4,PD,A,5,3,CH,D)

WER047A **xxxxx STATEMENT HAS SYNTAX ERROR**
EXPLANATION: One of the SyncSort control statements, xxxxx, contains a syntax error. The name of the statement containing the syntax error will be reflected in the message.

WER048I **E32 EXIT IGNORED WITH SORT STATEMENT**
EXPLANATION: An exit E32 was specified for a sort program. Since an E32 can be used only with a merge, the exit was ignored and the sort continued.

WER049A **CALCAREA INCOMPATIBLE WITH MERGE/COPY**
EXPLANATION: CALCAREA cannot be specified for a merge or copy application.

WER051A **OVERLAPPING SORTWK FILES ON 'nnn'**
EXPLANATION: Two (or more) of the SORTWK files within the jobstream have overlapping extents. The 'nnn' device address identifies the relevant ASSGN statements.

WER054A **MIXED CKD AND FBA SORTWKS ARE ILLEGAL**
EXPLANATION: When Fixed Block Architecture (FBA) devices (3310 and 3370) are used for SORTWK, they must be the only work device type. They can not be mixed with the standard Count Key Data (CKD) devices (3380, 3390, etc.).

WER056A **FILEIN CONFLICTS WITH INPFIL STMT**
WER056A **FILES CONFLICTS WITH JOINKEYS STMT**
EXPLANATION: The number of input files specified in the FILES parameter on the SORT, MERGE, or JOINKEYS control statement dif-

fers from the number of input files specified in the FILES/FNAMES parameter on the INPFIL control statement.
ACTION: Correct either the FILES parameter or the number of INPFIL control statements so that both parameters specify the same number of input files.

WER058I **ALTSEQ STMT IGNORED - NO AQ FIELD**
EXPLANATION: If an ALTSEQ statement is used, code AQ for the f value of at least one control field in the FIELDS parameter of the SORT or MERGE statement. The sort will run as if the ALTSEQ statement had not been coded.

WER059A **RECORD LEN/BLKSIZE INVALID FOR DEVICE**
EXPLANATION: The l_2 value on the RECORD statement exceeded the track capacity of the device used for SORTWK, or the BLKSIZE value specified on the INPFIL/OUTFIL statement exceeded the track capacity of the device used for SORTIN/SORTOUT.

WER063I **STORAGE AVAIL**
EXPLANATION: This is the amount of storage, given in decimal bytes, in which the sort/merge is running.

WER065A **PROBABLE DECK STRUCTURE ERROR**
EXPLANATION: An invalid operation definer was found.
ACTION: Check for misspelling of operation definers such as control statements.

WER066I **filename CONTROL INTERVAL SIZE=xxxxx**
EXPLANATION: filename indicates the filename (e.g. SORTOUT, SORTOF2). xxxxx indicates the control interval size SyncSort has selected for SAM SORTOUT on an FBA device. This value was determined by user DLBL specifications or by internal calculations.

WER067A **DUPLICATE xxxxx STATEMENT FOUND**
EXPLANATION: One of the allowable SyncSort control statements, xxxxx, was specified more than once. The name of the duplicated statement will be reflected in the message.

WER068A **INVALID CISIZE SPECIFIED**
EXPLANATION: The CISIZE value specified on the DLBL statement is insufficient for one block or is not a multiple of 512. Or, a disk space management package has changed the CISIZE via a catalogue entry to a value insufficient to accommodate one block of data.

WER070A **INVALID INPUT DEVICE FOR ADDROUT**
EXPLANATION: SORTIN is an FBA device and CKD ADDROUT format is not applicable.

WER072A **INVALID DATA CONVERSION REQUEST**

EXPLANATION: Data conversion has been requested 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 ZD, PD, BI, or FI.

WER075A **NO EXTENT FOR SORT WORK**

EXPLANATION: One or more extents for sort work files were not defined.

WER076A **INSUFFICIENT GETVIS AVAILABLE**

EXPLANATION: The LABEL macro has returned a code of X'1C', which has this meaning.
ACTION: Provide additional GETVIS area using the EXEC statement's SIZE parameter.

WER080I/A **INCONSISTENT LUNS: SYSxxx, SYSyyy**

EXPLANATION: There is an inconsistency between SyncSort defaults and label processing for an I/O file. xxx is the default; yyy is the label value.

Note: At the user's option, this message may result in critical error termination.

WER081A **KEY LENGTH EXCEEDS RECORD LENGTH**

EXPLANATION: The sum of the lengths of the control fields is greater than the length specified in one or more of the l values on the RECORD statement.

WER082A **INVALID BLKSIZE SPECIFIED**

EXPLANATION: The BLKSIZE value has been incorrectly specified. The following rules must be observed in specifying BLKSIZE:

1. For fixed-length input records, the blocksize specified in the INPFIL statement must be a multiple of the length of the input record (l_1). For ASCII records, the BUFOFF value, if specified, must be added.

2. For fixed-length output records, the blocksize specified in the OUTFIL statement must be a multiple of the length of the output record following E35, OUTREC, and OUTFIL processing.

3. For variable-length EBCDIC records, the blocksize specified in the INPFIL and OUTFIL statements must be at least the length of the longest record plus 4 bytes for the Block Descriptor Word.

4. For variable-length ASCII records, the block size specified in the INPFIL and OUTFIL statements must be at least the length of the longest record plus the BUFOFF value, if specified.

5. The blocksize cannot exceed 65,528 for EBCDIC records or 9,999 for ASCII records.

WER083A **VTOC FAILURE: R15=xx, LUN=yy**
EXPLANATION: VTOC processing for yy has resulted in a non-zero return code of xx. Check CVH Return Codes in the appropriate VSE LIOCS manual.

WER084A **CRDSIZE NOT EVENLY DIVISIBLE BY LRECL**
EXPLANATION: For SYSIPT input, CRDSIZE must be an integral multiple of the record length value, l_1.

WER085A **UNABLE TO PERFORM SORTWORK ERASE**
EXPLANATION: The sort was unable to acquire the storage needed to perform the requested erasing of the sortwork file(s).
ACTION: Provide additional GETVIS area using the EXEC statement's SIZE parameter.

WER090A **INPFIL EXIT REQUIRES AN E15 OR E32**
EXPLANATION: An INPFIL statement specifying EXIT was found, but no E15 or E32 exit was included.

WER091A **NUMERIC FIELD ERROR ON xxxxx STATEMENT**
A numeric value was specified improperly on a SyncSort control statement, xxxxx. The name of the statement containing the incorrect specification will be reflected in the message.

WER099A **EQUAL TAPE LUNS FOUND FOR MERGE**
EXPLANATION: This message is received when two or more files, being input to the same merge, are assigned to the same logical unit. This situation results in termination of the merge.

WER100A **INVALID STATEMENT BEFORE END STMT**
EXPLANATION: Either a misspelled operation definer was found in a control statement, or an extraneous statement was present.

WER101A **DATA=A INVALID**
EXPLANATION: A control statement or parameter was specified that is not permitted when DATA=A. Or, an input or output file other than a tape file was specified.

WER102A **LENGTH CHANGED, EXIT NOT SPECIFIED**
EXPLANATION: The l_2 or the l_3 values were not equal to the l_1 value

on the RECORD statement, but neither an exit nor an OUTREC statement was included.

WER103A **HEADER/TRAILER FIELD BEYOND RECORD**
EXPLANATION: A data field specified in a HEADER or TRAILER is beyond the length of the record.
ACTION: Make sure that the position plus the length of the data field specified in the HEADER or TRAILER does not exceed the length of the record.

WER105A **INVALID FORMAT FOR ASCII DATA**
EXPLANATION: An invalid value was specified for the code format (f). Only AC, ASL, or AST are valid format values for ASCII data.

WER106A **PHASE NAME TOO LONG**
EXPLANATION: A phase name specified on the MODS statement is more than the allowed eight characters.

WER108A **EXIT ADDRESS IS INVALID**
EXPLANATION: An absolute address that is not on a doubleword boundary was specified in the MODS statement (in the loading information parameter).

WER109A **REXX ENVIRONMENT UNAVAILABLE**
EXPLANATION: The current operating environment does not support the execution of a REXX exit. A REXX exit can only be run on an ESA mode machine with REXX/VSE installed.

WER110A **FILE NAME TOO LONG**
EXPLANATION: A file name of over eight characters was specified on the OPTION statement.

WER112A **NO NUMERIC FIELD ON MODS STATEMENT**
EXPLANATION: A MODS statement with a missing loading address was found.

WER113I **BLOCK SIZE EXCEEDS CI SIZE**
EXPLANATION: The BLKSIZE specified on either the INPFIL or OUTFIL control statement is greater than the size of the control interval specified for the corresponding input file.

WER114A **INVALID INTERMEDIATE STORAGE DEVICE**
EXPLANATION: This message is due to an error in the JCL specification of files. User SORTIN files may not be fully specified or there may be an inconsistency between the values of the WORK and FILES parameters - somehow a discrepancy has arisen between the number of files and the number of file labels in use.

WER115A **REQ'D PARM MISSING ON JOINKEYS STMT**
EXPLANATION: A parameter was not included on the JOINKEYS control statement as required.

WER116A **MISSING PARM FOR SORT OR MERGE**
EXPLANATION: A required parameter is missing from the SORT or MERGE statement.

WER117A **PHASE 0 HAS CRITICAL ERROR**
EXPLANATION: The sort/merge has found an error in Phase 0 that is too serious to allow the program to run.

WER118A **MINIMUM XXXX SORT WORK AREA nnnnnn**
EXPLANATION: Phase 0 has been completed and the CALCAREA parameter (requested explicitly using the OPTION statement or implicitly using ANALYZE) has terminated the sort with this report of the number of tracks (for CKD devices) or blocks (for FBA devices) necessary for the SORTWK files. XXXX indicates the device type, and nnnnnn indicates the number of tracks or blocks. If DEVWK is specified, SyncSort will display the number of tracks needed for the work area of the listed devices. Otherwise, it will display the number of tracks needed for each of the CKD and FBA devices.

WER119A **EXCESS ERRORS NOT DISPLAYED**
EXPLANATION: A very large number of errors have been generated. Some of the error messages will not be displayed.
ACTION: Correct all of the indicated errors.

WER120A **OVERLAPPING FIELDS IN HEADER/TRAILER**
EXPLANATION: A field specified in a header or trailer overlapped another field.

WER123A **SUM FIELD BEYOND RECORD**
WER123A **DUPKEYS FIELD BEYOND RECORD**
EXPLANATION: A field specified either on the SUM control statement or the DUPKEYS control statement is beyond the length of the record.

WER124A **ILLEGAL OVERLAPPING OF SUM FIELDS**
WER124A **ILLEGAL OVERLAPPING OF DUPKEYS FIELDS**
EXPLANATION: A field specified in the SUM or DUPKEYS statement overlapped with another field in the same statement or a SORT/MERGE control field.

WER127A **INREC/OUTREC FIELD BEYOND RECORD**
REFORMAT
BUILD/OVERLAY
EXPLANATION: A field specified on the INREC, OUTREC,

REFORMAT, BUILD, or OVERLAY control statement or parameter was either beyond the minimum length of the record or more than 4096 bytes long.

WER128A **ILLEGAL INREC/OUTREC DATA FIELD**
REFORMAT
BUILD/OVERLAY

EXPLANATION: Any of the following may have occurred:

1. A non-numeric value was given for a numeric field.
2. A zero value was specified.
3. A value was omitted.
4. A space value was greater than 256.
5. An alignment value other than H, F, or D was specified.
6. The last value specified was a p value indicating the variable portion of a record, but fixed-length records were specified.
7. Overlapping output fields were specified.
8. No data portion (past the RDW) was specified for variable-length records.

WER129A **INCLUDE/OMIT FIELD BEYOND RECORD**
IFTHEN

EXPLANATION: The length of the field specified on an INCLUDE, OMIT, or IFTHEN control statement or parameter for either inclusion or omission is longer than the length of the user's records.

WER130A **INCLUDE/OMIT INVALID COND**
IFTHEN **SELF DEF TERM**
LENGTH
FIELD POSITION
FORMAT
LOGICAL OPERATOR
WILDCARD SYNTAX

EXPLANATION: The indicated parameter on the INCLUDE, OMIT, or IFTHEN statement or parameter was incorrectly specified.

WER132A **INCLUDE/OMIT FORMATS INCOMPATIBLE**
IFTHEN

EXPLANATION: A format code in either a field-to-field comparison or field-to-constant comparison is illegal.

WER133A **INREC/OUTREC RDW NOT INCLUDED**
REFORMAT
BUILD/OVERLAY
EXPLANATION: The Record Descriptor Word was not included when the INREC, OUTREC, REFORMAT, BUILD, or OVERLAY control statement or parameter was coded for variable-length records.

WER134A **SORTOUT REC LEN INCOMPAT W/OUTREC**
EXPLANATION: The length specified as the l_3 value in the RECORD statement is not equal to the length of the output record defined by the OUTREC statement.

WER136I **INREC/OUTREC RECORD LENGTH=xxxxx**
EXPLANATION: This message displays the record length, xxxxx, after INREC processing (if only INREC is specified) or after OUTREC processing (if OUTREC is specified). The record length is displayed in decimal bytes.

WER137A **STORAGE LESS THAN 64K MINIMUM**
EXPLANATION: The value specified for the STORAGE parameter on the OPTION statement is less than the minimum required for SyncSort for z/VSE.
ACTION: Increase the STORAGE parameter value so that it is at least equal to the minimum amount of storage required.

WER138A **ILLEGAL KEY LENGTH SPECIFIED**
EXPLANATION: A key is over 255 bytes or > the l_1 or l_3 values in the RECORD statement or the input records are blocked or the output records are blocked and ADDROUT was not specified.

In addition, the user will receive this message if KEYLEN was specified and SORTIN was specified as VSAM.

WER139A **EXIT NOT SPECIFIED FOR N.S. LABEL**
EXPLANATION: The LABEL parameter on the OPTION statement indicates non-standard SORTIN and/or SORTOUT but the MODS statement does not specify an appropriate exit (E11 or E17 for non-standard SORTIN, E31 or E37 for non-standard SORTOUT).

WER140I **ROUTE=nnn IGNORED [INVALID DVC TYPE]**
EXPLANATION: Messages were not routed to the logical unit specified by 'nnn' because ROUTE=nnn was specified for a JCL sort/merge, or an invalid device type was specified for an invoked sort/merge.

WER141A **SORTWK DEFAULT/DEVWK DEVICE INVALID**
EXPLANATION: The SORTWK device specified as the installation default or specified as the 'nn' variable for DEVWK in the OPTION con-

trol statement is invalid. Check the table of valid codes for disk device types.

WER146A **CONVERT/FTOV/VTOF CONFLICTS WITH PARM**
EXPLANATION: The CONVERT/FTOV/VTOF parameter has been specified on the INPFIL or OUTFIL control statement and one of the following is also true:

1. One or more of the following have also been specified in the control statements: OPTION ADDROUT; RECORD TYPE=VS, VBS, F, or D; OUTFIL SPAN or EXIT; MERGE FIELDS=COMPARE; INPFIL DATA=A.

2. OUTREC was not specified in the OUTFIL control statement.

WER147A **INVALID ADDRESS IN INVOKING PARM LIST**
EXPLANATION: The parameter list created by the invoking program contains an address pointer to a sort control statement that is outside of the partition.

WER148A **OUTFIL STMT HAS SYNTAX ERROR ON FNAME**
EXPLANATION: The specified filename on the OUTFIL statement is not found in any output DLBL/TLBL JCL.

WER149A **OUTFIL STMT HAS SYNTAX ERROR ON DATE**
WER149A **XSUMFIL STMT HAS SYNTAX ERROR ON DATE**
WER149A **XDUPFIL STMT HAS SYNTAX ERROR ON DATE**
EXPLANATION: The specified DATE=(DMYc) parameter has a syntax error. D, M, and Y represent the output sequence of day (D), month (M), or year (4 or Y). Check that each parameter occurs only once. For example, DATE-(MDY.) is correct whereas DATE=(MMY.) causes a syntax error.

WER150A **XSUMFIL INCOMPATIBLE WITH DUPKEYS**
WER150A **XDUPFIL INCOMPATIBLE WITH SUM**
EXPLANATION: The XSUMFIL control statement cannot be specified with the DUPKEYS control statement. In the same way, the XDUPFIL control statement cannot be specified along with the SUM control statement.

WER151A **LOCALE CONFLICT WITH CHALT**
EXPLANATION: The LOCALE option cannot be specified with CHALT.

WER152A **LANGUAGE ENV. NOT LINKED IN SYNCSORT**
EXPLANATION: SyncSort for z/VSE must be properly installed with the IBM Language Environment product library in order to use any

COBOL exit, C exit, and/or the LOCALE function. Contact your local system programmer for information on SyncSort for z/VSE installation.

WER153A **LOCALE SERVICE NOT LINKED IN SYNCSORT**
EXPLANATION: SyncSort for z/VSE was installed with a version of the IBM Language Environment product that does not support LOCALE. Contact your local system programmer for information on SyncSort for z/VSE installation.

WER154A **NO LOCALE SUPPORT FOR JOINKEYS**
EXPLANATION: The LOCALE option cannot be specified with JOIN processing.

WER155I **LOCALE IN EFFECT: xxxx**
EXPLANATION: Indicates that LOCALE processing is in effect. xxxxx is the name of the effective LOCALE.

WER156I **IGNRL4/VLSHRT IGNORED - LOCALE USED**
EXPLANATION: The IGNRL4 or VLSHRT option is ignored when the LOCALE option is specified.

WER157A **INVALID IFOUTLEN SPECIFIED**
EXPLANATION: The IFOUTLEN parameter has specified a length greater than the maximum record length (l_1) on the RECORD control statement.

WER158A **INTERNAL RECORD LENGTH EXCEEDS 64K**
EXPLANATION: The length of the records internal to the sort is larger than the allowed 64K bytes. Record length is determined after adjustments due to INREC and E15 processing and expansion due to EQUALS, LOCALE, and certain data types (such as AC, AQ, and the Y2K formats).

WER159A **NO LABEL FOUND FOR xxxxxxx**
EXPLANATION: A DLBL statement was not found for the sortwork file with xxxxxxx for its file name.
ACTION: Add a DLBL statement with the file name xxxxxxx or correct the DLBL statement with the misspelled file name.

WER160A **OPTION/SYMNAMES MUST BE THE 1ST STMT**
EXPLANATION: When specifying SYMNAMES=option in the OPTION statement, the OPTION statement must precede all other SyncSort control statements.

WER161A **MATCHING QUOTE IS MISSING**
EXPLANATION: The data dictionary statement contains an open quote that has no matching closing quote.

WER162A **DATA NAME IS A RESERVED WORD**
EXPLANATION: The data name to be defined is a reserved word.

WER163A **DATA NAME ALREADY IN DICTIONARY**
EXPLANATION: The data name to be defined has been defined previously.

WER164A **REFERENCE TO UNDEFINED DATA NAME**
EXPLANATION: The POSITION data dictionary statement refers to an undefined data name.

WER165A **POSITION/LENGTH VALUE IS INVALID**
EXPLANATION: The data dictionary statement contains a position or length field that has an invalid value.

WER166A **DATA NAME OR CONSTANT LENGTH ERROR**
EXPLANATION: The data name is too long or the length of the constant is invalid for that type of constant.

WER167A **INVALID CHARACTER FOUND IN CONSTANT**
EXPLANATION: The constant contains a character that is invalid for that type of constant.

WER168A **FORMAT NAME IS INVALID**
EXPLANATION: The format specified is not one of the valid data formats.

WER169A **STATEMENT HAS SYNTAX ERROR**
EXPLANATION: The data dictionary statement has a syntax error.

WER170A **INFINITE READ LOOP DETECTED**
EXPLANATION: A data dictionary member, or a subsequent member invoked by it, contains a SYMNAMES statement that invokes itself again.

WER171A **NESTED SYMNAMES EXCEED LIMIT**
EXPLANATION: Only 15 levels of nested member invocations via SYMNAMES statements are allowed.

WER172A **SYMNLIB CONTAINS NONEXIST LIB/SUBLIB**
EXPLANATION: One or more library/sublibrary pairs specified in the SYMNLIB parameter was not found.

WER173A **MEMBER NOT FIXED RECORD FORMAT**
EXPLANATION: The data dictionary member must be cataloged as a fixed format type.

WER174A **MEMBER NOT FOUND**
EXPLANATION: The library member referenced by the SYMNAMES statement was not found.

WER175A **LIBRM xxxxx FAILED, RC=yy ,RSN=X'zz'**
EXPLANATION: Library access macro xxxxx issued by SyncSort returns a non-zero code yy and a reason code zz (in hex).
ACTION: Refer to macro documentation provided with the operating system for further explanation.

WER176A **NO LIB/SUBLIB OR SYMNLIB SPECIFIED**
EXPLANATION: The SYMNAMES parameter or statement does not include library name and sublibrary name, and there is no SYMNLIB parameter that specifies which libraries and sublibraries to search for the member.

WER177A **DATA DICTIONARY HAS SYNTAX ERROR**
EXPLANATION: One or more data dictionary statements has a syntax error.

WER178A **DATA DICTIONARY HAS CRITICAL ERROR**
EXPLANATION: Data dictionary processing encountered a critical error.

WER179A **SYMBOL/SYNTAX ERROR ON xxxxx STMT**
EXPLANATION: A symbol or syntax error was detected during symbol substitution for the control statement.

WER180I **PROCESSING DATA DICTIONARY STATEMENT**
EXPLANATION: SyncSort dictionary processing has started.

WER181I **STARTING SYMBOL SUBSTITUTION**
EXPLANATION: The SyncSort dictionary has been established. The SyncSort control statements will be scanned and the field names or constant names will be replaced with their values as defined in the dictionary.

WER182I **DATA DICTIONARY PROCESSING COMPLETE**
EXPLANATION: SyncSort dictionary generation and substitution of field_name and constant_name values in the sort control statements have completed successfully.

WER183A **OPEN FAILED FOR SORTWKn**
EXPLANATION: An error occurred while opening SORTWKn. n indicates the SORTWK number, such as SORTWK1.

WER184I **SORTWKn TRUNCATED, xxxxxxx TRACKS RELEASED FROM=yyyyyyy TO=zzzzzzz**

EXPLANATION: This message is issued for every sortwork file that has tracks or blocks that can be truncated at the end of phase 2 (or phase 1 if phase 2 is not needed). For FBA sortworks, BLOCKS are shown instead of TRACKS. This message is only issued if DMS=BIME-PIC is coded in SYNCMAC.

WER185A **INVALID MINIMUM RECORD LENGTH (L_4)**
EXPLANATION: The specified record length is less than the minimum record length (l_4).

WER186A **JOIN CAPACITY EXCEEDED BY nnnnK**
EXPLANATION: SyncSort is unable to complete the join application because of insufficient memory. The message indicates the amount of additional memory required in the partition to successfully complete the join process.
ACTION: Run the join application in a larger partition that includes the additional memory. Contact SyncSort for z/VSE Product Services for assistance, if necessary.

WER187I **JOIN SPACE ALLOCATED=nnnnK, USED=NNN**
WER187I **ADDITIONAL VIA GETVIS=nnnnK, USED=NNN**
EXPLANATION: Indicates the amount of memory used by the join process. In addition, if GETVIS memory was used by the join process, it will also be indicated.

WER188A **SPLIT/SPLITBY/REPEAT CONFLICT WITH REPORT WRITING**

EXPLANATION: The SPLIT, SPLITBY, or REPEAT parameter and one or more report writing parameters have been specified on an OUTFIL control statement. SPLIT, SPLITBY, or REPEAT and report writing parameters are incompatible on the same OUTFIL control statement with HEADERn, TRAILERn, LINES, NODETAIL, and SECTIONS.

WER189I **JOINKEYS REFORMAT RECORD LENGTH=nnnn, TYPE=(F/V)**
EXPLANATION: Indicates the record length and record type generated by the join process.

WER190I **SORTxxx HAS UNIT ASSIGNED IGNORE**
EXPLANATION: The indicated input or output file has its logical unit assigned to IGN. If it is an input file, data will not be read from this file. If it is an output file, data will not be written to this file.

WER191A **UNEQUAL NUMBER OF JOINKEYS FIELDS**
EXPLANATION: The number of JOINKEYS fields specified on the two JOINKEYS control statements are not equal.
ACTION: Make sure that the two JOINKEYS control statements define the same number of fields in the FIELDS parameter.

WER192A **INPFIL STMT CONFLICTS WITH JOINKEYS**
EXPLANATION: INPFIL can not be used with JOIN.

WER193A **REFORMAT IS REQUIRED TO JOIN RECORDS**
EXPLANATION: A join application requires two JOINKEYS control statements and a REFORMAT control statement.

WER194A **JOIN FEATURE REQUIRES TWO INPUT FILES**
EXPLANATION: A join application requires two JOINKEYS statements that define the same number of JOINKEYS FIELDS, with each corresponding field having the same length and order specified.

WER195A **SUM JOIN CONFLICTS WITH JOINKEYS**
EXPLANATION: The JOIN parameter can not be specified when JOINKEYS control statements are specified.

WER196A **VSAMSAM BLOCK NOT A MULTIPLE OF LRECL**
EXPLANATION: The sort read a VSAM-managed SAM block whose length was not a multiple of the logical record length.
ACTION: Correct the logical record length.

WER197I **REFORMAT STMT IGNORED, NO JOINKEYS**
EXPLANATION: The REFORMAT statement was ignored because neither a JOINKEYS control statement was present nor the JOIN parameter specified on the SUM control statement.

WER198I **JOIN STMT IGNORED, NO JOINKEYS**
EXPLANATION: The JOIN control statement requires the presence of a JOINKEYS control statement for SORTIN1 and SORTIN2.

WER199A **SOME PARMS INCOMPATIBLE WITH JOINKEYS**
EXPLANATION: The following parameters are incompatible with the JOINKEYS control statement: E11, E15, E17, E31, E32, and E37 exits, SKIPREC on the OPTION control statement, and CKPT on the SORT control statement.

WER200A **XSUM PROCESSING HAS CRITICAL ERROR**
WER200A **XDUP PROCESSING HAS CRITICAL ERROR**
EXPLANATION: A critical error has occurred during XDUP or XSUM processing. The program has terminated.

WER201A **NO MATCH FOR INREC/OUTREC CHANGE PARM**
EXPLANATION: A field was found on an input record that did not match any search constants specified by the CHANGE subparameter on an INREC, OUTREC, or OUTFIL OUTREC control statement, and the NOMATCH subparameter was not specified.

ACTION: Add a NOMATCH subparameter, or add the missing search constant to the CHANGE subparameter list.

WER202A **INVALID DATA FOR ACTIVE LOCALE**
EXPLANATION: The active LOCALE produces no translation for data contained in a sort or merge field.

WER203A **LOCALE SERVICE (CEExxxx) FAILED, RC=nnnn**
EXPLANATION: A call to Language Environment service used to support LOCALE processing indicated a critical error in its feedback code. CEExxxx is the name of the service, and nnnn is the error message number contained in the feedback code.

WER204A **UNEXPECTED TERMINATION OF LE/VSE**
EXPLANATION: SyncSort did not complete because the Language Environment (LE/VSE) established by SyncSort has terminated prematurely.
ACTION: Examine the error message produced by LE/VSE to determine the source of the problem. Also, ensure there is enough GETVIS storage available to satisfy the application.

WER205A **INCOMPATIBLE JOINKEYS FIELDS FORMATS**
EXPLANATION: One or more of the corresponding fields on the two JOINKEYS control statements are in incompatible formats.

WER206A **LENGTH OF SECTIONS FIELD > 256**
EXPLANATION: The length value specified in a control field for the SECTIONS parameter of an OUTFIL control statement exceeds the allowed maximum.

WER207I **SUM OVERFLOW OCCURRED**
EXPLANATION: Two or more records that should have been combined due to the specification of the SUM or DUPKEYS statement were not combined, because addition of the sum fields would have caused arithmetic overflow in the field of the specified size.

WER208A **INVALID LOGICAL BLOCK FOUND**
EXPLANATION: A block read from the input file had one of the following inconsistencies:

1. The blocksize exceeded the value supplied for INPFIL BLKSIZE.

2. For fixed-length records only, the block size was not a multiple of the LRECL.

3. For fixed-length records only, INPFIL BLKSIZE was omitted, signifying unblocked input, and the blocksize was not equal to the LRECL.

WER209A **xxxxxxxx yyyyyyyy FIELD OVERFLOW**
EXPLANATION: xxxxxxxx is the output filename, yyyyyyyy can be TOTAL, SUBTOTAL, AVG, or SUBAVG. This message indicates that the value of the yyyyyyyy field is greater than 15 decimal digits.

WER210A **SORTOUT OUT OF SEQUENCE (MAY BE E35)**
EXPLANATION: The sort/merge has detected an out-of-sequence condition on output. This may be caused by sort key modification using an E35 exit routine or may be related to a WER215A overlapping SORTWK condition.
ACTION: Whenever an E35 exit routine is to modify sort control fields, the sequence check word should be set to a non-zero value. Also, see "Common SyncSort Errors" on page 12.29.

WER211A **THE FOLLOWING H/T LINE IS GT LRECL:**
EXPLANATION: This message flags any HEADERs or TRAILERs that exceed the LRECL specification. The HEADER or TRAILER itself will be included in the message text.

WER212A **OUTREC ARITHMETIC OVERFLOW**
WER212A **OUTFIL OUTREC ARITHMETIC OVERFLOW**
EXPLANATION: An overflow has occurred while processing an OUTREC arithmetic function.
ACTION: Expand the size of the OUTREC fields being processed to handle larger numbers.

WER213A **INVALID ANSI CONTROL CHARACTER FOUND**
EXPLANATION: The OUTFIL control statement specifies LINES as ANSI, requiring control characters in the output records. A carriage control character was found that is invalid for the device type specified. Note that SyncSort does not support X, Y, Z for a card punch.

WER214A **UNIT RECORD MAXIMUM EXCEEDED**
EXPLANATION: The number of cards punched or logical pages printed has exceeded either (1) the number specified in the CARDS or PAGES parameter; or (2) the installation default for CARDS or PAGES (delivered defaults: 32,767 CARDS, 32,767 PAGES).

WER215A **SORTWK OUT OF SEQ. CHECK FILE OVERLAP**
EXPLANATION: First, see "Common SyncSort Errors" on page 12.29. If the problem persists, rerun with SPECIAL DIAG (UPSI of 11110111) to obtain a dump and call VSE Product Services.

WER216I **ZERO RECORDS INPUT TO SORT**
EXPLANATION: No records were received by the sort. Either there was an empty input file or the records were deleted by an exit or by the INCLUDE/OMIT statement.

WER217A **INPUT SEGMENTS IN WRONG ORDER**
EXPLANATION: The sequence of segments of a spanned record in an input file is incorrect. For example, an initial segment must be read before a middle or ending segment, and an ending segment cannot precede a middle segment.

WER218A **SVA INSTAL ERR: PHASE=xxxxxxxx CODE=x**
EXPLANATION: SyncSort for z/VSE has been incorrectly installed in the SVA. This may imply an "SVA full" condition, or an error applying maintenance to SyncSort for z/VSE. "xxxxxxxx" is the phase name for which the error was found. The CODE=x is information for SyncSort for z/VSE Product Services.

WER219B **TRK OVER-ALLOC FACTOR (OAF)=ALLOC/USED=**
EXPLANATION: The number of tracks allocated for sortwork is divided by the number of tracks actually used by SyncSort, giving the over-allocated factor. To find out how many tracks SyncSort used, divide the number of tracks allocated by the number given above.

WER220A **SORTINx OUT OF SEQUENCE: record**
EXPLANATION: SORTINx file was not previously sorted according to the requested MERGE FIELDS. "record" is the number of the out of sequence record in the SORTINx file.

WER221I **B=**
EXPLANATION: This is the blocksize used for sortwork files.

WER222I **G=**
EXPLANATION: This is the number of records (or segments of variable-length records) that can fit in storage during Phase 1.

WER223B **NMAX=**
EXPLANATION: This is the number of records of the length defined in the RECORD statement that can be sorted in the sortwork area assigned to them.

WER224I **L5=xxxxx, L6=yyyyy, L7=zzzzz**
EXPLANATION: When the VL5L6L7 PARM is used, SyncSort analyzes the distribution of variable-length records to calculate optimal values of l_5, l_6, and l_7. These values can be specified for the LENGTH parameter on the RECORD control statement for subsequent sorts using this file to improve sorting performance. (If INREC is used, adjust l_5 as described on page 6.8.)

WER225I *****END SORT PHASE*****
EXPLANATION: The end of a sort phase has been reached.

WER226A **END SYNCSORT (jobname), RECORD=xxxxxxxxxx, INCORE**
MERGE
OAF=xxxx
UAF=xxxx
VSWK

EXPLANATION: The end of the sort/merge program indicated by the jobname has been reached. The number of records is the amount in SORTOUT or (if the job is a merge or has no SORTOUT file) the amount deleted. Incore sorts and MERGE applications are so identified. OAF and UAF values report on the result of dividing the number of tracks allocated by the user for SORTWK by the number of tracks SyncSort used. If the primary allocation was sufficient, this number will be greater than one and is reported as an overallocation factor (QAF). If secondary allocation was necessary, this number will be less than one and is reported as an underallocation factor (UAF). To find out how many tracks SyncSort actually used, divide the number of tracks allocated by the user by the OAF/UAF. In the case of underallocation, subtract the primary allocation from this figure to determine the secondary allocation in tracks. For VSE/ESA 1.3 or above, VSWK indicates that the sort used Dynamic Storage Manager. For releases of VSE/ESA prior to release 1.3, VSWK means that the sort used Virtual Sortwork.

WER227I **RCD IN xxx, OUT yyy, XSUM zzz**
WER227I **RCD IN xxx, OUT yyy, XDUP zzz**

EXPLANATION: The number of records that went into the sort and the number of records that came out at this phase in the program are given. If the XSUM or XDUP parameter is specified, the number of records that were eliminated by SUM or DUPKEYS is also given.

EXPLANATION: The number of records that were inserted and the number of records that were deleted from the sort/merge at this phase in the program are given.

WER228I **INSERT xxx, DELETE yyy**

EXPLANATION: The number of records that were inserted and the number of records that were deleted from the sort/merge at this phase in the program are given.

WER229A **OPEN ERROR FOR SORTIN [VSAM CODE=xx]**

EXPLANATION: An error occurred while opening SORTIN. If SORTIN is a VSAM file, xx gives the VSAM error code.

WER230A **OPEN ERROR FOR SORTOUT [VSAM CODE=xx]**

EXPLANATION: An error occurred while opening SORTOUT. Whenever SORTOUT is a VSAM file, xx gives the VSAM error code.

WER231A **INVALID CISIZE AFTER OPEN**

EXPLANATION: The CISIZE value on an output file has been changed to a value insufficient to accommodate one block of data. This usually occurs when a disk space management package changes the CISIZE value via a catalogue entry.

ACTION: Increase the CISIZE value or decrease the blocksize.

WER232A **IMPROPER RETURN CODE FROM E15**
E18 E32 E38
E25 E35 E39

EXPLANATION: The user's exit program passed SyncSort an invalid return code.

WER234A **DIAG=nnnnnnnn nnnnnnnn nnnnnnnn nnnnnnnn nnnnnnnn nnnnnnnn**

EXPLANATION: This message contains information that can be used by SyncSort for z/VSE Product Services to diagnose errors or to enhance performance.

WER235B **TRK ALLOC=xxx, USED=yyy**

EXPLANATION: The number of tracks allocated for sortwork and the number of tracks actually used for sortwork are given.

WER236A **USER EXIT RETURN CODE TERMINATE**

EXPLANATION: The user's program passed SyncSort a code of 16 to terminate the sort and it has been obeyed.

WER237A **RECORD LENGTH EXCEEDS MAXIMUM (L_1/L_2)**

EXPLANATION: A variable-length input record has been found that is greater than the length specified in the l_1/l_2 parameter on the RECORD statement.

WER238A **INCORE SORT CAPACITY EXCEEDED**

EXPLANATION: An incore sort was requested, but input was too large to fit into storage.

ACTION: Either supply sortwork files or increase storage size.

WER239A **TURNAROUND SORT PERFORMED**

EXPLANATION: Although sortwork files were supplied, the input was able to be sorted completely in storage, so the intermediate and final merge stages were bypassed.

WER240A **SORTWK DEVICE NOT EQUAL DEFAULT/DEVWK**

EXPLANATION: The SORTWK device type does not correspond to the installation default or to the device types specified for the DEVWK parameter in the OPTION control statement.

WER241A **RECORD LEN EXCEEDS L3, USER REQ ABORT**

EXPLANATION: A variable-length record was found that is greater than the length specified in the l_3 parameter on the RECORD statement.

WER242I **SEGLEN**

EXPLANATION: The value given above was used by SyncSort to sort the user's variable-length records.

WER243A **NOT ENOUGH STORAGE IN INT. MERGE PHASE**
FINAL MERGE PH.

EXPLANATION: The sort ran out of storage during the phase shown.

WER244A **STRING CT OVERFLOW; INCREASE STORAGE**

EXPLANATION: There was too much data for the sort to process in the storage available.

ACTION: Specify an increased value for the SIZE parameter on the JCL EXEC statement and/or VSCORE/VSCORET parameter on the OPTION statement.

WER245A **RECORD LENGTH LESS THAN MINIMUM (L4)**

EXPLANATION: The sort found a variable-length record that is shorter than the value the sort is using for l_4.

WER246I **WRONG LENGTH RECORD. filename**
BYPASSED

EXPLANATION: The sort found a wrong-length physical or logical record. If BYPASSED is on the message, the record was ignored and the sort continued. If BYPASSED is not on the message, the filename for the wrong-length record will be given.

WER248A **RECORD LENGTH EXCEEDS TRACK CAPACITY**

EXPLANATION: The record length specified must be no greater than the track capacity of the SORTWK disk device.

WER250A **ZERO LENGTH RECORD FOUND**

EXPLANATION: A variable-length record with a length of zero in its Record Descriptor Word was found by the sort/merge.

WER251A **RDW INVALID, OVERFLOWS BUFFER**

EXPLANATION: The sort found a variable-length record with a length value in its Record Descriptor Word that, when added to the address of the end of the previous record in this input buffer, would overflow the buffer. This message will also be issued when the length of an assembled spanned record exceeds the l_1 value (input buffer size).

ACTION: Check for a bad record in the input files.

WER252A **HEADER/TRAILER EXCEEDS LINES ON PAGE**
EXPLANATION: The total number of lines in the HEADER(S) and/or TRAILER(s) exceeds the number of lines per page determined either by default or by the LINES parameter on the OUTFIL control statement.
ACTION: Increase the number of lines per page or reduce the total number of lines in the HEADER(S) and/or TRAILER(s).

WER253A **SORT CAPACITY EXCEEDED**
EXPLANATION: Sortwork file space was exceeded.
ACTION: Increase sortwork file allocations.

WER254A **UNSUPPORTED DEVICE SWITCH ON OUTPUT**
EXPLANATION: There are three "classes" of output devices; DASD, tape, and unit record. The output file switched from one class to another, and that class switch is unsupported.

WER255I *****END PHASE 2*****
EXPLANATION: The sort/merge has completed Phase 2.

WER256I **EXTENTS EXHAUSTED FOR SORTWKx**
WER256I **PROCEEDING WITH REMAINING SORTWK FILES**
EXPLANATION: The two WER256I messages are informational and do not require any response. The WER256I message may be preceded by the IBM message:

4n50A NO MORE AVAILABLE EXTENTS

SyncSort for z/VSE tries to obtain secondary disk space allocations for all VSAM-SAM SORTWK files. SyncSort will first obtain allocations for SORTWK1 until no further secondary allocations are available for SORTWK1, then will try to obtain secondary disk space allocations for SORTWK2. This process will be repeated for every VSAM-SAM SORTWK file until sufficient SORTWK space has been obtained to complete the sorting process.

WER300A **ABE xxx CALL SYNCSORT INC. AT 201-930-8210**
EXPLANATION: The sort diagnosed a critical internal error and can no longer run.
ACTION: Rerun the program using the DIAG option or UPSI 11111111 to get a dump. Then, call SyncSort for z/VSE Product Services at the above number.

WER301A

IOERROR,	**SORTWK**	**CCB@**	
IOERROR,	**filename**		
IOERROR,	**CCB=**		
IOERROR,	**LOGICAL,**	**VSAM ERROR CODE=xx,**	**R15=yy**
IOERROR,	**PHYSICAL,**	**VSAM ERROR CODE=xx,**	**R15=yy**
IOERROR,	**UNKNOWN,**	**VSAM ERROR CODE=xx,**	**R15=yy**

EXPLANATION: An I/O error has occurred. If byte 5 of the CCB shows a X'40', there is a wrong-length record. If the error occurred while processing a VSAM file, the message will indicate whether it is known to be logical or physical. xx is the error code; yy is the value of register 15 returned by VSAM.

WER400A **CRITICAL ERROR. SORT TERMINATED**

EXPLANATION: Due to a critical error, the sort could not run to completion.

ACTION: Check for other error messages to determine the nature of the error.

WER401I **STARTING INTERNAL SORT FOR JOIN**

EXPLANATION: SyncSort has started an additional internal sort as part of JOIN processing.

WER402I **END INTERNAL SORT, RECORD=nnnnn, OAF**
UAF
INCORE

EXPLANATION: The additional internal sort started by SyncSort as part of JOIN processing has successfully completed. nnnnn is the number of records processed by this internal sort.

WER403A **ERROR RETURNED FROM INTERNAL SORT**

EXPLANATION: The additional internal sort started by SyncSort as part of JOIN processing has encountered a problem. There will be additional messages displayed related to this problem.

WER405A **INPFIL LRECL REQUIRES AN INREC**

EXPLANATION: An INPFIL control statement specifying LRECL was different than the length specified on the LENGTH parameter on the RECORD control statement, but no INREC control statement was included.

WER406I **SORTINx: IN nnnnn, OUT nnnnn**
LRECL= nnnnn, BLKSIZE=xxxxx, INREC=(inr1,inr2)
CISIZE=xxxxx

EXPLANATION: The number of records that went into the INPFIL processor and the number of records that came out of this processor are given. If the input file is VSAM, xxxxx indicates the control interval size SyncSort has selected, otherwise the BLKSIZE value is displayed along with the LRECL value. If INREC is specified, inr1 indicates the user-specified input record length and inr2 indicates the record length selected by SyncSort.

WER407I **filename: DATA RECORDS OUT nnnnnnnnn**
EXPLANATION: nnnnnnnnn represents the number of data records (exclusive of HEADERs and/or TRAILERs) in this output data set.

WER408I **filename: TOTAL RECORDS OUT yyyyyyyyy**
EXPLANATION: yyyyyyyyy represents the total number of records in this output data set (data records and HEADERs and/or TRAILERs). Note that the total number of lines written to the line printer may be greater than the actual record count, since multiple lines of blank output are considered one data record.

WER409I **JOIN FILE n: PAIRED nnnnn, UNPAIRED nnnnn**
EXPLANATION: The n value is either 1 or 2 and indicates the number of paired and unpaired records processed by the join application.

WER500A **SYNCSORT IN ERROR RECOVERY, PLEASE DO NOT CANCEL**
EXPLANATION: A critical failure has occurred and SyncSort is in error recovery. See SYSLOG and/or SYSLST output for an explanation of the error.

WER900A **FIELDS=COMPARE. THE RECORDS ARE UNEQUAL. PDUMP SHOWS EACH SORTIN DTF. THE NEXT PDUMP SHOWS THE RESPECTIVE RECORDS.**
EXPLANATION: Two non-EOF records paired by the MERGE were not byte-for-byte identical. For a description of the two PDUMPS, see "Using the FIELDS=COMPARE Parameter" on page 5.26. This message will be followed by WER901A. Although this is an "A" message, it will *not* appear on the console.

WER901A **FIELDS=COMPARE; SORTIN1 RCDS=xxxxxxxxxx**
SORTIN2 RCDS=xxxxxxxxxx
EXPLANATION: Two non-EOF records paired by the MERGE were not byte-for-byte identical. At this point, an equal number of records has been processed from each of the two files (since the MERGE processes record pairs); the numbers given reflect the position of the current unequal records in their respective files. Although this is an "A" message, it will *not* appear on the console.

WER901I **FIELDS=COMPARE; SORTIN1 RCDS=xxxxxxxxxx**
SORTIN2 RCDS=xxxxxxxxxx
EXPLANATION: The merged files were identical. The numbers given reflect the number of records in each file. Since the files are identical, these numbers must be equal.

WER902A **FIELDS=COMPARE; SORTIN1 RCDS=xxxxxxxxxx**
SORTIN2 RCDS=xxxxxxxxxx; THE FILES ARE UNEQUAL.
EXPLANATION: The two files are of different sizes, with the first n

records (the smaller number in the message) of the larger file being identical to the smaller file in its entirety. The numbers given reflect the number of non-EOF records read from each file. Since the file size comparison is done in terms of record pairs processed, however much larger one file is than another, it will still be reported as having only one additional record. Although this is an "A" message, it will not appear on the console.

Common SyncSort Errors

The errors described in this section are those that occur most frequently.

1. WER253A SORT CAPACITY EXCEEDED. This occurs when the work file space allocated for the sort is insufficient in size. As a general rule, the amount of work file space should be 15 to 20 percent more than the amount of space required for the total sort input.

2. WER039A INSUFFICIENT STORAGE. There is not enough storage for the sort to run in. This may occur when the sort is initiated from a program that occupies a large amount of storage. This error is also caused by inadvertently specifying 30K rather than 300K on the SIZE parameter of the EXEC job control statement. In either situation, the sort/merge will terminate and a message will be issued indicating the error.

3. WER246I WRONG LENGTH RECORD

   - User input consists of variable-length records and the Block Descriptor Words on the user's tape contain errors. When this occurs, the user's blocked input has been incorrectly described to the sort. The sort may still run, but the results will be unpredictable.

   - The wrong record size or block size is specified. This type of error is created by typing or simple arithmetic errors. If any of the length values specified on either the RECORD statement or the BLKSIZE parameters on the OUTFIL statement is incorrect, the effects on the user's program are unpredictable.

4. DATA EXCEPTION. The user's control fields contain bad data. This error is often a result of: a programmer specifying ZD or PD for a field in the SORT statement that contains unsigned data; including a blank in a field specified as PD or ZD. To avoid abnormal termination of the sort when this message is received, the user can specify CMP=CLC on the OPTION statement. SyncSort will then accept the records containing the bad data and proceed with the sort/merge.

5. WER301A I/O ERROR. Input/output errors may occur. This common error can be caused by a hardware failure such as a bad spot in a tape or disk. When this occurs the sort is unable to read or write the user's data. The program will terminate, generating a message to indicate the error.

6. VARIOUS SYNTAX ERRORS. Conflicting parameters are specified in the control statements. An example of this situation would be a programmer specifying DELBLANK in the RECORD statement when either an INCLUDE statement or an OMIT statement has been specified in the program. In this situation the sort/merge terminates, generating a message to indicate the error.

7. INCORRECT OUTPUT. There are program exits, but the MODS statement is omitted from the control stream. In this situation the sort/merge will run, but the exits will not be utilized.

8. An E15 program that reads input or an E35 program that writes output is included without the inclusion of an INPFIL or OUTFIL statement specifying EXIT. In this situation the sort/merge terminates, generating a message to indicate the error.

9. The use of SYSIPT for card image data. SYSIPT must be assigned to an actual card reader or to SYSRDR simulated input (i.e., Diskette).

10. WER215A SORTWK OUT OF SEQ. CHECK FILE OVERLAP

    - Overlapping SORTWK extents. To solve the problem, change the SORTWK JCL.

    - Another simultaneously executing program is overlapping your SORTWK area(s). This should only arise in the absence of a Disk Space Management package. Both the JCL and the label cylinders must be checked to determine if this is the problem. Revise the extent information in the JCL to eliminate overlap.

11. 0S12I 'SUB XXXXXX CANCELED DUE TO MAIN TASK TERMINATION'. This message may not indicate a problem, since it may have been caused by an early termination code from an E15 or E35 exit (or COBOL output procedure). Typical examples include branching out after processing a fixed number of records or branching out as soon as a certain condition is met. Whenever an E35 exit (COBOL output procedure) neglects to process the entire sorted file, this message will be generated and should be ignored. In other circumstances, this message may reflect a serious problem in that the invoking program has terminated without proceeding to the sort EOJ.

SSRAM Messages

SSRAM Error Checking

SSRAM checks the format of the call and contents of the parameter table each time that it is called. If SSRAM discovers an error in the call or the parameter table, it terminates processing immediately and issues an error message. If SSRAM experiences an error during processing, its action depends on whether the invoking program requested return codes.

If the invoking program does not request return codes, SSRAM sends an error to the console if SYSLOG is assigned. A partition dump may be produced and the program is canceled.

If the invoking program does request return codes, SSRAM passes control back to the invoking program along with the return code table. The return code table will contain either return code 4 or 16 and the error message. If SSRAM then receives another call from the invoking program after experiencing an error on the preceding call, SSRAM cancels the program. Therefore, the invoking program should include a check of requested return codes so that it takes the appropriate action.

Any error SSRAM encounters during initialization is non-recoverable regardless of whether return codes were requested by the invoking program. The program is canceled.

SSRAM Message Format

All SSRAM messages have the following format:

```
WERSnnn xxxxx message
```

Figure 450. SSRAM Message Format

nnn This is the message number.

xxxxx This is either INIT or the sortwork filename passed in the parameter table. INIT means that the error occurred during initialization. The sortwork file name lets you identify which sort experienced the error.

message This is the message text.

Note: These messages are available in the VSE console EXPLAIN function if your systems programmer installed SyncSort with this option.

WERS001 **RE-OPEN FUNCTION NOT SUPPORTED.**
EXPLANATION: An attempt was made to use either the SRTOPN, CAROP, or SSAROP call to re-use a sortwork file and this function is not supported.

WERS002 **DUPLICATE RECORD DELETE OPTION NOT SUPPORTED.**
EXPLANATION: An attempt was made to delete a duplicate record during sort processing and this function is not supported.
ACTION: You can delete only records that have duplicate keys.

WERS005 **MULTIPLE SRTCORE CALLS ARE ILLEGAL**
EXPLANATION: More than one SRTCORE function call was used for the application.

WERS006 **SRTCORE CALL MUST PRECEDE SRTOPEN CALL**
EXPLANATION: The application issued a SRTCORE function call after issuing a SRTOPEN function call.

WERS010 **SRTCORE STORAGE ADDRESS IS INVALID**
EXPLANATION: The storage address passed in the SRTCORE parameter list is invalid. The address is invalid for one of the following reasons:

- The address is lower than the beginning address of the partition.
- The address is higher than the ending address of the partition.
- A storage length value was not specified and the address is in the partition GETVIS area.

WERS011 **SRTCORE STORAGE LENGTH IS INVALID**
EXPLANATION: The storage length value passed in the SRTCORE parameter list is invalid. The storage length is invalid for one of the following reasons:

- If the SRTCORE storage address is in the problem program area of the partition (not the GETVIS area), then the ending address implied by adding the storage length to the storage address must also be in the problem program area (not the GETVIS area).
- If the SRTCORE storage address is in the GETVIS area, then the ending address implied by adding the storage length value to the storage address must be less than the ending address of the GETVIS area.

WERS101 **SUBSID SVC FAILURE. RETURN CODE=xx**
EXPLANATION: An SVC 105 (X'69') was issued and failed. xx is the return code from SUBSID.
ACTION: Call SyncSort for z/VSE Product Services.

WERS102 **INIT OPERATING SYSTEM NOT SUPPORTED.**
EXPLANATION: SSRAM supports only standard IBM operating systems.

WERS103 **INIT INSUFFICIENT STORAGE.**
EXPLANATION: Either the main storage value passed in the SSRAM parameter table during a SRTOPEN call was less than 10K *or* there was not enough storage in the partition to run SSRAM.
ACTION: In the first case increase the main storage value in the SSRAM parameter table to at least 10K. In the second case, SSRAM

requires a minimum of 64K in the partition for each sort in addition to the storage required for the invoking program and the 8K required by SRAM itself.

WERS104 **INIT GETVIS NOT INITIALIZED.**
EXPLANATION: The partition GETVIS area has not been properly initialized by VSE.
ACTION: Call SyncSort for z/VSE Product Services.

WERS105 **INIT PARAMETER TABLE NOT FULLWORD ALIGNED.**
EXPLANATION: The SSRAM parameter table is not aligned to a fullword boundary.
ACTION: If you are using PL/I, use the ALIGNED keyword in the definition of the parameter table. If you are using COBOL, use the SYNCHRONIZED keyword in the definition of the parameter table.

WERS106 **xxxxx LABEL ERROR DETECTED.**
EXPLANATION: The DLBL statement in the JCL does not match bytes 1 to 5 of the SSRAM parameter.
ACTION: Correct the DLBL statement so that it matches bytes 1 to 5 of the SSRAM parameter.

WERS107 **xxxxx KEY DEFINITION ERROR DETECTED.**
EXPLANATION: There is a syntax error in the sort key definition.
ACTION: Ensure that opening and closing parentheses match, that all the keys are valid, or that there are no typographical errors. See "The Key Definition Table" on page 10.3.

WERS108 **xxxxx ALREADY OPENED.**
EXPLANATION: An attempt was made to open the specified file and that file is open already.
ACTION: Check to see if more than one sort is using the same parameter table.

WERS109 **xxxxx LOGICAL UNIT ASSIGNMENT ERROR.**
EXPLANATION: There are several possible reasons for this error message. The SSRAM workfile name in the parameter table is assigned to a tape (TLBL) instead of a disk (DLBL). Either the Logical Unit Number (LUN) is a non-numeric value, is greater than 255, is unassigned, or is set to IGNORE.
ACTION: Change the TLBL to a DLBL. The LUN cannot be a non-numeric value, cannot be greater than 255, cannot be unassigned and cannot be set to IGNORE. Correct the parameter table and re-run the job.

WERS110 **xxxxx FILL CALL ERROR.**
EXPLANATION: The FILL call was issued in an improper sequence,

either preceding an SRTOPEN call or following a CLOSE or GET call.
ACTION: Correct the program that issues the calls.

WERS111 **xxxxx FILL PARAMETER LIST INCOMPLETE.**
EXPLANATION: The record area is missing from a FILL call.
ACTION: Provide the record area.

WERS113 **xxxxx GET CALL ERROR.**
EXPLANATION: The GET call was issued in an improper sequence, either preceding an OPEN or FILL call or following a CLOSE.
ACTION: Correct the program that issues the calls.

WERS114 **xxxxx GET PARAMETER LIST INCOMPLETE.**
EXPLANATION: The record area is missing from a GET call.
ACTION: Provide the record area.

WERS116 **xxxxx CLOSE CALL ERROR.**
EXPLANATION: The CLOSE call was issued in an improper sequence, preceding an OPEN, FILL, or GET call.
ACTION: Correct the program that issues the calls.

WERS191 **xxxxx INVALID DUPLICATE RECORD PROCESSING CODE.**
EXPLANATION: A code other than 0, 4, or 8 was passed to SSRAM.
ACTION: 0, 4, or 8 are the only valid duplicate record processing codes. Correct the fullword that contains the duplicate record processing code.

WERS201 **xxxxx FREEVIS FAILURE.**
EXPLANATION: SSRAM received a non-zero return code in response to a FREEVIS request.
ACTION: Call SyncSort for z/VSE Product Services.

WERS202 **xxxxx WORK FILE DEVICE NOT SUPPORTED.**
EXPLANATION: The LUN is assigned to a device other than DASD.
ACTION: Correct the JCL and rerun the job.

WERS203 **INIT INCORRECT RELEASE OF SYNCSORT/VSE.**
EXPLANATION: This release of SSRAM is incompatible with the release of SyncSort for z/VSE that you are trying to use.
ACTION: Call SyncSort for z/VSE Product Services.

WERS204 **INIT PHASE RELEASE MISMATCH.**
EXPLANATION: SSRAM has detected that its two phases are from different releases.
ACTION: Call SyncSort for z/VSE Product Services.

WERS210 **xxxxx SORT INITIALIZATION ERROR.**
EXPLANATION: Prior to the completion of the first FILL call, SyncSort for z/VSE experienced an error.

ACTION: See accompanying sort messages. Call SyncSort for z/VSE Product Services.

WERS211 **xxxxx SORT ERROR DURING FILL PHASE.**
EXPLANATION: While SSRAM was passing records with a FILL call, SyncSort for z/VSE experienced an error.
ACTION: See accompanying sort messages. Call SyncSort for z/VSE Product Services.

WERS212 **xxxxx SORT ERROR DURING INTERMEDIATE MERGE PHASE.**
EXPLANATION: Prior to the completion of the first GET call, SyncSort for z/VSE experienced an error.
ACTION: See accompanying sort messages. Call SyncSort for z/VSE Product Services.

WERS213 **xxxxx SORT ERROR DURING FINAL MERGE PHASE.**
EXPLANATION: While SSRAM was receiving records with a GET call, SyncSort for z/VSE experienced an error.
ACTION: See accompanying sort messages. Call SyncSort for z/VSE Product Services.

SYNCBIX Messages

SYNCBIX is SyncSort's high performance replacement for IDCAMS. All SYNCBIX messages have prefix SBIX. Otherwise the message format is the same as WER messages. SYNCBIX messages printed are written to SYSLST with a record length of 121 bytes and SYSLST must be ASSGNed to a print device.

Note: These messages are available in the VSE console EXPLAIN function if your systems programmer installed SyncSort with this option.

SBIX001A **INVALID DEBUGGING OPTION**
EXPLANATION: The option specified for debugging is not permissible.

SBIX002A **FILE xxxxxxxx OPEN ERROR. VSAM RC=X'nn' , EC=X'nn'....**
EXPLANATION: An error was encountered while opening a VSAM file.
ACTION: Check the VSAM Return and Error codes for detailed information.

SBIX003A **FILE xxxxxxxx : INVALID RECORDSIZE SPECIFIED**
EXPLANATION: The value specified in the RECORDSIZE parameter of the VSAM cluster definition exceeds the maximum limit of 32760 bytes.

SBIX004A **FILE xxxxxxxx OPEN ERROR: NOT AIX FILE**
EXPLANATION: The file specified by the OUTFILE parameter must be defined as an AIX file.

SBIX005A **AIX DATASET MUST BE A KSDS**
EXPLANATION: The file specified by the OUTFILE parameter is not properly defined as an alternate index.

SBIX006A **FILE xxxxxxxx IS NOT A BASE CLUSTER**
EXPLANATION: The file specified by the OUTFILE parameter is not defined as a base cluster or a path over a base cluster.

SBIX007A **INVALID CONTINUATION STMT:**
EXPLANATION: A continuation character was specified on the current control statement with no additional control statements following. The invalid statement is displayed following this error message.

SBIX008A **INVALID PARAMETER SPECIFIED:**
EXPLANATION: A keyword or value is not recognized. It may be misspelled. The invalid parameter is displayed following this error message.

SBIX009A **DUPLICATE xxxx PARAMETER FOUND**
EXPLANATION: The specified parameter xxxx was specified more than once.

SBIX010A **MISSING INFILE/OUTFILE PARAMETER**
EXPLANATION: One or more of the required parameters were not specified.

SBIX011A **INVALID SORTWK NUMBER SPECIFIED**
EXPLANATION: The allowable WORK values are from 1 through 9.

SBIX012I **TRUNCATED nnnn EXCESS AIX KEY xxxx**
EXPLANATION: The alternate index record was too short to contain all of the prime key or RBA pointers. The record is created with only those pointers that would fit. nnnn gives the number of pointers that could not fit into the record. xxxx represents the AIX key value that had too many primary keys associated with it.
ACTION: Increase the maximum length of the RECORDSIZE value in the IDCAMS DEFINE AIX command.

SBIX013I **VSAM FILE xxxxxxxx BUILT WITH ERRORS**
EXPLANATION: This is an informational message, indicating that the file was built with alternate index records without all of the

record pointers. This message is displayed along with message SBIX012I.

SBIX014A **VSAM FILE xxxxxxxx PUT ERROR, RC=X'XX' ,EC=X'xx'**
EXPLANATION: An error occurred while writing a logical record into the VSAM AIX file. Check the VSAM Return and Error codes for detailed information.

SBIX015I **VSAM FILE xxxxxxxx SUCCESSFULLY BUILT**
EXPLANATION: The alternate index file was built successfully with no errors.

SBIX016A **VSAM FILE xxxxxxxx WAS NOT BUILT**
EXPLANATION: The alternate index file was not built due to some error.

SBIX020A **SYNCBIX TERMINATED. ERROR FROM SORT, RC=X'xxxx'**
EXPLANATION: SYNCBIX was terminated due to an error that occurred during sort operation.

SBIX021A **CRITICAL ERROR. INVALID AIX TYPES**
EXPLANATION: SYNCBIX was terminated due to internal errors.

SBIX022I **NONUNIQUE AIX KEY xxxx; PRIME KEY yyyy**
EXPLANATION: The AIX was defined as unique. More than one record with AIX key value xxxx was encountered. The primary key was yyyy. SYNCBIX continued to build the remainder of the AIX. The keys are displayed in hexadecimal. If the length of a key is greater than 20 bytes, only the first 20 bytes of the key are displayed in the message.

SBIX023A **INSUFFICIENT STORAGE**
EXPLANATION: SYNCBIX did not have enough storage to load the SORT phase. SYNCBIX requires at least 100K plus the maximum record length of the alternate index file.

SYNCHSTO Messages

There are two main types of messages for the SYNCHSTO program:

1. **SHSnnnI** messages, messages ending in 'I', are informational. They describe control information used by SYNCHSTO and the characteristics of the records that have been read.

2. **SHSnnnA,** messages ending in 'A', indicate critical error conditions. SYNCHSTO terminates to allow you to correct the error(s) so that a successful program may be run.

SHS001I **NUMBER OF RECORDS nnnn**
EXPLANATION: This displays the number of records that have been read by SYNCHSTO. This is either the total number of records in the file, or the limit imposed by the NRECS parameter.

SHS002I **TOTAL LENGTH OF ALL RECORDS nnnn**
EXPLANATION: This is the total number of bytes read.

SHS003I **AVERAGE RECORD LENGTH nnnn**
EXPLANATION: This is the total number of bytes read divided by the number of records read.

SHS004I **KEY END POSITION nnnn**
EXPLANATION: This is the end location of the last sort control field to be used, specified by the KEYEND control parameter (or using the assumed default value).

SHS005I **HISTOGRAM LINE WIDTH nnnn**
EXPLANATION: This is the numeric interval between the minimum and maximum record lengths for each line of the histogram display.

SHS006I **LONGEST RECORD nnnn**
EXPLANATION: This is the length of the longest record read.

SHS007I **SHORTEST RECORD nnnn**
EXPLANATION: This is the length of the shortest record read.

SHS008I **MOST COMMON RECORD LENGTH (L5) nnnn**
EXPLANATION: This is the most frequently occurring record length in all of the records that have been read. Supply SyncSort with this value as the l_5 value in the LENGTH parameter of the RECORD control statement to improve sort performance.

SHS009I **AVG. SPACE PER RECORD (L6) nnnn**
EXPLANATION: This is the estimated average work space per record that has been calculated by SYNCHSTO. Supply SyncSort with this value as the l_6 value in the LENGTH parameter of the RECORD control statement to improve sort performance.

SHS010I **RECOMMENDED SEGMENT SIZE (L7) nnnn**
EXPLANATION: This is the recommended segment size that has been calculated by SYNCHSTO. Supply SyncSort with this value as the l_7 value in the LENGTH parameter of the RECORD control statement to improve sort performance.

Since the l_7 value depends on the location of the sort key within the record, the actual optimal value for l_7 may be affected if INREC or

an E15 reformats the records. In this case, the value estimated by SYNCHSTO is less accurate for a particular sort application than the value of l_7 calculated using the VL5L6L7 PARM during that sort.

SHS101A **NO PARAMETER STATEMENT FOUND**
EXPLANATION: No parameter statement followed the
// EXEC SYNCHSTO statement.

SHS102A **INVALID LRECL VALUE**
EXPLANATION: LRECL must be a numeric value no larger than 65524.

SHS103A **INVALID BLKSIZE VALUE**
EXPLANATION: BLKSIZE must be a numeric value no larger than 65528.

SHS104A **INVALID SORTNAME**
EXPLANATION: SORTNAME must be 1-8 alphanumeric characters beginning with an alphabetic character.

SHS105A **INVALID NRECS VALUE**
EXPLANATION: NRECS must be a numeric value no larger than 2147483647.

SHS106A **INVALID KEYEND VALUE**
EXPLANATION: KEYEND must be a numeric value less than or equal to LRECL.

SHS107A **INVALID PARAMETER SPECIFIED**
EXPLANATION: An invalid or misspelled parameter has been specified.

SHS108A **BAD RETURN CODE FROM SYNCSORT**
EXPLANATION: The sort that is used by SYNCHSTO has encountered an error. See the beginning of this chapter for further information about error messages produced by the sort.

SHS109A **INVALID RECORD TYPE FOUND**
EXPLANATION: The input file must have variable-length records. If the file has blocked records, the BLKSIZE parameter must be specified.

SHS110A **NO PARAMETERS FOUND**
EXPLANATION: A blank parameter statement has been found.

SHS111A **DUPLICATE PARAMETERS SPECIFIED**
EXPLANATION: A valid parameter has been specified more than

once and may have conflicting values (such as KEYEND=nnnn,KEYEND=mmmm).

SHS112A **CONFLICTING PARAMETERS SPECIFIED**
EXPLANATION: Parameters have been specified with explicitly conflicting values (such as OPEN=RWD,OPEN=NORWD).

SHS113A **REQUIRED LRECL PARM NOT FOUND**
EXPLANATION: The LRECL parameter must be specified.

SHS114A **INSUFFICIENT STORAGE**
EXPLANATION: There was insufficient storage available for SYNCHSTO to run. Increase the SIZE=parameter on the // EXEC statement, or run the program in a larger partition.

SHS115A **INVALID VOLUME VALUE**
EXPLANATION: VOLUME must be a numeric value.

SHS116A **INVALID WIDTH VALUE**
EXPLANATION: WIDTH must be a numeric value, an even multiple of 4, and no larger than 65524.

SHS117A **MISSING CONTINUATION**
EXPLANATION: A continuation has been specified by ending a control statement with a comma, but no continuation statement followed it.

SHS118A **SORTIN1 FILE CONTAINS NO RECORDS**
EXPLANATION: The input file to SYNCHSTO is empty.

Appendix A. Devices and Software Supported by SyncSort

Devices Supported by SyncSort

All current disk and tape devices may be used for SyncSort input and output files as well as any card reader for input and card punch or printer for output; all current disk devices may be used for workfiles.

Whenever new devices become available, support will be developed for them. Should any problems arise with new devices, please let us know.

Proprietary Software Packages Compatible with SyncSort

All proprietary software packages that interface with a sort/merge program are compatible with SyncSort. Should any problems arise, please let us know.

Via a default setting (see the *SyncSort for z/VSE Installation Guide*), SyncSort can take advantage of the presence of a variety of disk space management packages to dynamically appropriate and release disk space at various stages of the sort/merge. For example, under any package that allocates and frees disk space at OPEN and CLOSE, unused SORTOUT space and all SORTWK extents will be returned to the system at termination of the sort. Whenever all current SORTWK space is exhausted and more is needed, SyncSort will OPEN additional extents from the preassigned pool that is part of these disk space management packages.

For information regarding specific compatibility with a disk space management package, please contact SyncSort for z/VSE Product Services. (See *SyncSort for z/VSE Installation Guide* for technical support.)

Appendix B. Helpful Formulas for SyncSort Programs

Calculating How Much Disk Workfile Space Is Needed for a Job

The rule-of-thumb is 15 to 20 percent more workfile space is needed than the amount of space occupied by the total input for the job.

Minimum Storage Needed to Run a Sort or Merge

The minimum storage requirement for SyncSort is 64K. This minimum applies for small applications; for more complex applications (such as the use of SortWriter features, multiple output files and user exits) it is recommended that at least 128K of storage be used.

Appendix C. VSE/VSAM Space Management for SAM Files

Introduction

This appendix outlines the use of VSE/VSAM space management for SAM files (VSAM/DMS/SAM) with SyncSort. It assumes a basic understanding of VSAM file organization and access techniques, VSAM space allocation, and the IDCAMS utility. In particular, the reader is assumed to have a working knowledge of the material covered in these IBM publications:

- *VSE/VSAM Commands*
- *VSE/VSAM User's Guide and Application Programming*

The discussion that follows is for the user who has VSAM/DMS/SAM installed. This user has already allocated space for Master and User catalogues, and for VSAM to build and manage data structures and files. This appendix discusses the use of VSAM/DMS/SAM for SORTOUT, SORTWK, and SORTIN files.

SAM ESDS Files

By definition, SAM ESDS files are SAM files that have been loaded into VSAM/DMS/ SAM; these files are typically changed to control interval format as part of the load. "Work files" are those files whose space is managed by VSAM/DMS/SAM but whose format is important only to the program using the file, e.g., SORTWK files. Regardless of the RECORDFOR-

MAT value specified by the user, SORTWK files will not be loaded in control interval format, since this would lead to inefficiency.

SAM ESDS files can be read and accessed by VSAM as well as SAM access methods, providing RECORDFORMAT is set to F, FB, V, or FB, and not to U (undefined) or NCIF (non-control interval, used only for work files). The VSAM parameter must be specified in the DLBL statement. In the absence of other deciding factors, SyncSort will choose the access method for a particular SAM ESDS file based on the presence or absence of a VSAM parameter in the INPFIL/OUTFIL statement:

1. VSAM in the INPFIL or KSDS/ESDS/RRDS in OUTFIL statement: VSAM access.

2. VSAM parameter omitted from INPFIL/OUTFIL statement: SAM access.

With a RECORDFORMAT value of U, however, only SAM access methods can be used.

VSAM-Managed SORTIN Files

A SAM ESDS input file is a file that already exists and has been defined as part of VSAM/DMS/SAM. Therefore, a cluster already exists and it is necessary only to alert SyncSort to its presence. The JCL for SAM ESDS SORTIN files must include a DLBL statement specifying VSAM; EXTENT and ASSGN statements are not necessary.

For VSAM access of a SAM ESDS input file, the format is given below in Figure 451.

```
// DLBL   SORTIN1,'cluster NAME',,VSAM
          .
          .
          .
   INPFIL VSAM
```

Figure 451. Sample for VSAM Access of a SAM ESDS Input File

For SAM access of a SAM ESDS input file, the format is given below in Figure 452.

```
// DLBL   SORTIN1,'cluster NAME',,VSAM
          .
          .
          .
   INPFIL BLKSIZE=xxx
```

Figure 452. Sample for VSAM Access of a SAM ESDS Input File

When mixing SAM ESDS files with other files as SORTIN, note the following:

- VSAM access is used if the VSAM parameter is specified in the INPFIL control statement.

- It is possible to mix a VSAM file and/or a SAM ESDS file.

- SAM ESDS files of different control intervals can be mixed.

- RECORDSIZE (block size) values can be mixed provided each value evenly divides into the BLKSIZE value specified in the INPFIL statement. In order for the BLKSIZE parameter to be processed, VSAM can not be specified in the INPFIL statement; SAM access methods will be used. As always, VSAM must appear in the DLBL statement to indicate a VSAM-managed file.

VSAM-Managed SORTWK Files

The goal is to have SORTWK space managed by VSAM/DMS/SAM, i.e., to have VSAM dynamically allocate the amount of SORTWK space requested for a particular job at OPEN and then delete that space at CLOSE. SyncSort opens its SORTWK files as OUTPUT using DTFPH.

VSAM-managed SORTWK files can be defined explicitly or implicitly (with or without a model). If specified in the DEFINE CLUSTER command for an explicit define or a default model, the optional RECORDS-RECORDSIZE pair of parameters will be used to determine allocation size. RECORDS and RECORDSIZE will not be used to determine the characteristics of the SORTWK file; SyncSort will control the use of the space allocated for SORTWK in its own way. All SORTWK files will be treated as NCIF, regardless of the RECORD-FORMAT value specified in the DEFINE CLUSTER command.

Successive sections describe the three methods of defining a VSAM-managed SORTWK file, detailing each technique's advantages and drawbacks.

Explicit Define Using DEFINE CLUSTER

In order to explicitly define SORTWK1,..., SORTWK9 as dynamic files, it is necessary to specify NOALLOCATION and REUSE in their definitions. These parameters instruct VSAM not to allocate space for these files until OPEN time. It is more efficient to access single-extent SORTWK files. In order to ensure that VSAM will allocate DASD space for SORTWK as a single extent, the special prefix DOS.WORKFILE.SYS must precede the data component name. The IDCAMS utility program is executed with the DEFINE CLUSTER command specifying NAME(%DOS.WORKFILE.SYSnnn...) for each of the clusters.

Note: Under VSAM/DMS/SAM, SyncSort automatically obtains the secondary allocation for SORTWK1 (WORK=DA is treated as WORK=1); therefore, the prefix is not necessary when defining SORTWK1. However, the prefix is always required for SORTWK2,...,SORTWK9.

The IDCAMS job must be run once for each work file (SORTWK1,..., SORTWKn; n≤9) in each partition in which SyncSort will execute. The '%' will add a partition identifier (BG,F1,...,Fn) at the end of the cluster name (File-ID). Thus, DOS.WORKFILE.SYSWK4.BG could be used by a sort running in BG as its File-ID for SORTWK4.

The major disadvantage of this approach is the initial overhead (as many as 9 executions of IDCAMS per partition). On the other hand, VSAM processing is most efficient when a cluster is already explicitly defined at OPEN time, and there is never a need for an EXTENT statement for SORTWK files that are explicitly defined as SAM ESDS files.

Implicit Define Using a Model

First, define a model that VSAM can use whenever it encounters a managed file (indicated by the VSAM parameter of the DLBL statement) whose File-ID does not have a match in the catalog. VSAM will use the model, which includes VOLUME information, to define a cluster for all such files. Since VOLUME information is provided by the model, SORTWK files will not require EXTENT statements when defined in this way.

Implicit Define Without a Model

In order to define SORTWK files implicitly without a model, both a DLBL and an EXTENT statement will be necessary.

JCL Requirements for VSAM-Managed SORTWK Files

The required DLBL statement in the job control stream or standard label cylinder is used to trigger VSAM/DMS/SAM involvement. In the case of an implicit define without a model, the DLBL statement must include allocation information (TRACKS/CYLINDERS/BLOCKS or both RECORDS and RECSIZE). The DLBL statement's allocation parameters are also used to override allocation values provided by IDCAMS for a default model. Since SyncSort opens its SORTWK files as OUTPUT, it is important to override the KEEP default value with DISP=(,DELETE), as in Figure 453, below.

```
//  DLBL  SORTWK1,'%DOS.WORKFILE.SYSWK1',0,
          VSAM,RECORDS=400,RECSIZE=80,
          DISP=(,DELETE)
```

Figure 453. Sample DLBL Statement for a VSAM-Managed SORTWK File

The DLBL statement above in Figure 453 instructs VSAM to allocate enough space to contain 400 80-byte records (i.e., 32,000 bytes of data) and to DELETE the file at CLOSE. Provided the cluster was not defined with a specific secondary allocation value, the primary allocation is automatically supplemented by a secondary allocation of 20% (i.e., enough space to contain 6,400 bytes of data) when the primary allocation is exhausted. Since SyncSort's structuring of the SORTWK space is not constrained by the RECORDS or the

RECSIZE values, RECORDS=400,RECSlZE=80 is equivalent to RECORDS=80, RECSIZE=400; or RECORDS=1, RECSIZE=32000.

The DLBL statement below in Figure 454 defines the primary allocation as enough space to contain 40,000 bytes of data (40 1,000-byte records) and the secondary allocation as 50% of that (20 1,000-byte records or 20,000 bytes of data).

```
//  DLBL  SORTWK1,"%DOS.WORKFILE.SYSWK1',0,
          VSAM,RECORDS=(40,20),RECSIZE=(1000),
          DISP=(,DELETE)
```

Figure 454. Sample DLBL Statement for a VSAM-Managed SORTWK File

Again, DISP's KEEP default must be changed to DELETE. Assuming SORTWK1 was explicitly defined or defined using a default model, allocation could be determined by IDCAMS and allocation values omitted from the DLBL statement.

An EXTENT statement is required only when SORTWK files are implicitly defined without a model. Figure 455 illustrates how to direct VSAM to the proper volume. It is assumed that the user has already defined VSAM space on this volume.

```
//  EXTENT  ,VOL1
```

Figure 455. Sample EXTENT Statement for a VSAM-Managed SORTWK File

An ASSGN statement is never required, although some unassigned logical units must be available for VSAM's use.

Sample JCL/Control Streams

```
//  JOB       EXPLICIT DEFINE FOR SORTWK3
//  EXEC      IDCAMS,SIZE=AUTO
  DEFINE CLUSTER
              (NAME(%DOS.WORKFILE.SYS.SORTWK3)-
              VOLUMES(VSER77)-
              CYLINDERS(8)-
              RECORDFORMAT(NCIF)-
              REUSE-
              NOALLOCATION-
              NONINDEXED)
/*
/&
```

Figure 456. Explicitly Defined SAM ESDS SORTWK3 File

In Figure 456, DEFINE CLUSTER specifies REUSE and NOALLOCATION to make SORTWK3 a dynamic file (primary allocation at OPEN). A typical cluster NAME for SORTWK3 might be %DOS.WORKFILE.SYS.SORTWK3, where the prefix is required by SyncSort and the entryname, 'SORTWK3', may be selected by the user. IDCAMS must be executed once for each partition in which sorting will be done in order to succeed in defining SORTWK3 as a SAM ESDS file. Similar jobstreams would be constructed for all additional SORTWK files to be used. The DLBL statement for SORTWK3 in the SyncSort jobstream must specify VSAM in order to identify the file as a SAM ESDS file, as below in Figure 457.

```
//  DLBL    SORTWK3,'%DOS.WORKFILE.SYS.SORTWK3',0,
            VSAM,DISP=(,DELETE)
```

Figure 457. DLBL for Explicitly Defined SAM ESDS SORTWK File

Note the use of DISP=(,DELETE) to delete the file at CLOSE, as required. Since allocation values are not coded on the sample DLBL statement, allocation is determined by the IDCAMS executions, which provide eight cylinders of primary space.

An EXTENT statement is not required.

```
//      JOB     MODEL FOR IMPLICIT SAM ESDS FILES
//      EXEC    IDCAMS,SIZE=AUTO
 DEFINE CLUSTER
                (NAME(DEFAULT.MODEL.ESDS.SAM)-
                VOLUMES(VSER12)-
                TRACKS(20)-
                RECORDFORMAT(NCIF)-
                REUSE-
                NOALLOCATION-
                NONINDEXED)
/*
/&
```

Figure 458. Implicitly Defined SAM ESDS SORTWK File Using a Model

The model is defined using IDCAMS with the required cluster NAME of DEFAULT. MODEL.ESDS.SAM and the NOALLOCATION parameter (also required). The volume specified can be overridden by an EXTENT statement for an implicitly defined SAM ESDS.

Although one of the DEFINE CLUSTER's allocation parameters (TRACKS/BLOCKS/ CYLINDERS or both RECORDS and RECORDSIZE) is specified as required, the DLBL statement can be used to change the allocation size.

Assuming the allocation value is acceptable for SORTWK, SORTWK2 is implicitly defined as SAM ESDS using the DLBL statement in Figure 459.

```
//    DLBL      SORTWK2,'%DOS.WORKFILE.SYSWK2',,
                VSAM,DISP=(,DELETE)
```

Figure 459. DLBL for Implicitly Defined SAM ESDS SORTWK File

Neither an EXTENT nor an ASSGN statement is required, although the EXTENT statement could be used to override the model's VOLUMES specification.

VSAM-Managed SORTOUT Files

The techniques used to define VSAM-managed SORTWK files are basically applicable to the definition of VSAM-managed SORTOUT files with the exceptions noted.

As before, VSAM must be specified in the DLBL statement. To access a SAM ESDS output file, as a VSAM file, you must additionally specify the KSDS/ESDS/RRDS parameter in the OUTFIL Statement, as well as explicitly defining SORTOUT with RECORDFORMAT (F, FB, V, or VB). If the parameter ESDS is specified on the OUTFIL card, the file can be read subsequently as either VSAM or SAM ESDS.

Explicit Define Using DEFINE CLUSTER

In order to explicitly define SORTOUT, it is necessary to specify NOALLOCATION and REUSE in its definition. These parameters instruct VSAM not to allocate SORTOUT space until OPEN time. Because SyncSort supports multiple extents for SORTOUT, the special DOS.WORKFILE.SYS prefix need not be used in the cluster NAME.

Implicit Define Using a Model

Whenever a model is used to define SORTOUT, the model's RECORDSIZE value becomes important. Specifying RECORDSIZE (4000 4000), for example, will force a 4096-byte control interval even if the SORTOUT attributes (e.g., BLKSIZE=700) reveal that the data would be more efficiently written to a smaller control interval. Since a larger BLKSIZE will force a large control interval, it is advisable to use a relatively small value for RECORDSIZE, such as RECORDSIZE(512 512), in the model.

Since VOLUME information is provided by the model, an EXTENT statement is not required.

Implicit Define without a Model

In order to define the SORTOUT file implicitly without a model, both a DLBL and an EXTENT statement will be necessary. The default for RECORDFORMAT in this case is U.

If the file is defined implicitly with or without a model, it cannot be read subsequently using VSAM access methods.

JCL Requirements for VSAM-Managed SORTOUT Files

In Figure 460, the DLBL statement in the job control stream or standard label cylinder is used to trigger VSAM/DMS/SAM involvement.

```
//  DLBL  SORTOUT,'SAM.ESDS.SORTOUT',0,
          VSAM,RECORDS=1000,RECSIZE=80,
          DISP=(NEW,KEEP)
```

Figure 460. Sample DLBL Statement for a VSAM-Managed SORTOUT File

The sample DLBL statement instructs VSAM to allocate enough space to contain 1,000 80-byte records (i.e., enough space to contain 80,000 bytes of data). Provided the cluster was not defined with a specific secondary allocation value, this primary allocation is automatically supplemented by a secondary allocation of 20% (i.e., enough space to contain 16,000 bytes of data) when the primary allocation is exhausted. Note the use of DISP=(NEW,KEEP) to retain the sorted output after the sort.

An EXTENT statement is required only when SORTOUT is implicitly defined without a model. Figure 461 illustrates how to direct VSAM to the proper volume. It is assumed that the user has already defined VSAM space on this volume.

An ASSGN statement is never required.

```
//  EXTENT   ,VOL1
```

Figure 461. Sample EXTENT Statement for a VSAM-Managed SORTOUT File

Setting Up a JCL/Control Stream for Sorts with VSAM-Managed Files

Figure 462 shows a sort with normal SAM input, output, and work files. The object of this sort is to create a paycheck file from the input payroll data. Later, paychecks will be printed from this output disk. The OUTREC feature has been used to select the pay, department, and name fields from the input record for the new, shortened output records.

```
//     JOB          PAYROLL
//     ASSGN        SYS001,X'142'
//     ASSGN        SYS002,X'142'
//     ASSGN        SYS003,X'144'
//     ASSGN        SYS004,X'145'
//     DLBL         SORTOUT,PAYCHECKS'
//     EXTENT       SYS001,111111,1,0,2500,400
//     DLBL         SORTIN1,'PAYFILE',30
//     EXTENT       SYS002,111111,1,0,500,2000
//     DLBL         SORTWK1,,0,DA
//     EXTENT       SYS003,444444,1,0,200,500
//     EXTENT       SYS003,444444,1,1,1000,500
//     EXTENT       SYS003,444444,1,2,2300,500
//     EXTENT       SYS004,555555,1,3,450,500
//     EXTENT       SYS004,555555,1,4,1060,500
//     EXTENT       SYS004,555555,1,5,1800,500
//     EXEC         SORT
       SORT         FIELDS=(64,6,PD,A,2,3,CH,A)
       RECORD       TYPE=F,LENGTH=190
       INPFIL       BLKSIZE=3800
       OUTREC       FIELDS=(64,6,2,3,10,30)
       OUTFIL       BLKSIZE=780
       OPTION       PRINT=CRITICAL,ROUTE=LOG
       END          ***OCTOBER 15 PAYCHECKS***
/*
/&
```

Figure 462. Sample JCL/Control Statement Stream, Normal SAM Files

For a detailed discussion of this example, see "A Disk Input and Output Sort" on page 5.12.

Figure 463 on page C.10 illustrates how to code an equivalent jobstream for VSAM-managed input, output, and work files.

Note: This jobstream assumes that the input file was previously loaded into VSAM-managed space as a SAM ESDS file.

```
//  JOB      PAYROLL VSAM-MANAGED PAYROLL                          1
//  DLBL     SORTOUT,'PAYCHECKS',30,VSAM,                          2
           RECORDS=4300,RECSIZE=780
//  EXTENT   ,DISK03                                               3
//  DLBL     SORTIN1,'VSAM.PAYFILE',,VSAM                          4
//  DLBL     SORTWK1,,,VSAM,DISP=(,DELETE)                         5
//  EXEC     SORT                                                  6
    SORT     FIELDS=(64,6,PD,A,2,3,CH,A)
    RECORD   TYPE=F,LENGTH=190
    INPFIL   VSAM                                                  7
    OUTREC   FIELDS=(64,6,2,3,10,30)                               8
    OUTFIL   BLKSIZE=780
    OPTION   PRINT=CRITICAL,ROUTE=LOG
    END      ***OCTOBER 15 PAYCHECKS***
/*
/&
```

Figure 463. Sample JCL/Control Statement Stream, VSAM-Managed Files

1 Creates a jobname.

2 The DLBL statement gives the filename SORTOUT to the VSAM-managed output file. The file-id is 'PAYCHECKS' and the retention period is 30 days. This file is implicitly defined to contain 4,300 780-byte records, or 3,354,000 bytes of output data.

3 The EXTENT statement directs VSAM to define SORTOUT on disks with serial number DISK03. The EXTENT statement is not required if a default model is available.

4 This DLBL statement indicates a VSAM-managed input file with a filename of SORTIN1, and a file-id of 'VSAM.PAYFILE' This file was defined by IDCAMS and previously loaded by another program.

5 This DLBL statement gives the filename for the work file as SORTWK1, omits both the file-id and retention date, and specifies VSAM space management. This file is assumed to have been previously defined by IDCAMS with the NOALLOCATE and REUSE attributes. VSAM is instructed to DELETE the file at CLOSE.

6 These statements are identical to those in the previous (normal SAM) example.

The EXEC statement gives SORT as the SyncSort program name.

The SORT statement shows two control fields. The first, or major field, starts on byte 64 of the input record, is 6 bytes long, has packed decimal format, and is to be sorted according to ascending sequence. The second, or lesser field, begins

on byte 2 of the record, is 3 bytes long, has character data, and is to be sorted according to ascending sequence.

The RECORD statement codes an F for fixed-length records, and indicates that the length of the input record is 190 bytes.

Note: This is the l_1 value. The l_2 and l_3 values will be calculated by the sort and the correct values will be used when it is time to change the length of the record for the output file.

7 The INPFIL statement specifies VSAM as required for VSAM access of a SAM ESDS input file.

8 These statements are identical to those in the previous (normal SAM) example.

The OUTREC statement shows how the input record is to be reconstructed to form the output record. The FIELDS parameter shows that three fields will be picked up. The first starts on byte 64 of the input record and is 6 bytes long. The second starts on byte 2 of the input record and is 3 bytes long. The third starts on byte 10 of the input record and is 30 bytes long. These are the pay, department number, and employee name fields, respectively, that will later be reproduced on paychecks. They will now run side-by-side from byte 1 through byte 39 of the output record, and will be edited, with spacing, later when they are printed.

The OUTFIL statement gives the blocksize of the output file as 780 bytes, i.e., twenty records per block. (VSAM access is not possible when SORTOUT is implicitly defined as in this example.)

The OPTION statement requests that only critical messages be issued and that they appear on the console.

The END statement requests the end of the control statements. A comment is given.

The standard job control end statements are given.

Index

R

Syncsort Incorporated
50 Tice Boulevard
Woodcliff Lake, New Jersey 07677
(201) 930-8200
E-mail: vse_tech@syncsort.com