

# GCSORT 1.03.04

## [15 GEN 2015 Version]

# User's Guide

**1<sup>st</sup> Edition**, 15 January 2016

Sauro Menna  
mennasauro@gmail.com

GCSORT Copyright © 2016-2017 Sauro Menna  
GCSORT Copyright © 2009 Cedric Issaly

Under the terms of the GNU General Public License

Document Copyright © 2016 Sauro Menna

Permission is granted to copy, distribute and/or modify this document  
under the terms of the GNU Free Documentation License [FDL], Version 1.3  
or any later version published by the Free Software Foundation.

*This work is dedicated to the memory of my niece Federica,  
a strong young woman, sweet and resourceful.  
You will always be in my heart and mind.*

## Summary of Changes

Edition	Date	Change Description
1 <sup>st</sup>	15 Jan 2016	INITIAL RELEASE OF DOCUMENT
	09 Nov 2016	UPGRADE version with integration of LIBCOB New Data Types SubString search Conditional
1.0.1	15 Oct 2020	New option in command line -fsign=EBCDIC/ASCII for NUMERIC field.
1.0.1	09 Jan 2021	INREC OVERLAY – OUTREC OVERLAY
1.03.02	18 Jan 2022	RECORD CONTROL STATEMENT / DATE - Current Date : DATE1, DATE2, DATE3, DATE4 / INREC CHANGE / OUTREC CHANGE / MODS E15 – E35
1.03.03	27 Mar 2022	JOIN Statement
1.03.04	4 Agu 2022	FINDREP in INREC/OUTREC Control statement

# Table of Contents

1.Introduction.....	6
1.1.What is GCSort?.....	6
2.Features.....	6
3.Environment and first use .....	9
3.1.Following the steps for the first use .....	9
3.2.Modify first environment variables .....	9
3.3.Use TAKE command .....	9
4.Process Schema .....	10
5.Sort .....	11
6.Merge .....	11
7.File Organization and Record Type .....	11
8.Field Type.....	11
Date Format.....	12
9.Commands.....	13
9.1.SORT .....	13
9.2.MERGE .....	13
9.3.COPY .....	13
9.4.FIELDS .....	13
9.5.USE.....	14
9.6.GIVE .....	15
9.7.INCLUDE/OMIT .....	15
9.8.INREC/OUTREC .....	17
9.9.SUM FIELDS.....	19
9.10.RECORD.....	19
9.11.OUTFIL .....	20
9.12.OPTION .....	21
Exit Routines.....	21
10.JOIN Statement.....	23
Join Process Schema.....	24
11.Environment Variables.....	28
11.1.Byte Order .....	28
11.2.Temporary Files .....	28
11.3.Memory Allocation.....	28
11.4.Statistics.....	29
12.Command Line.....	31
13.Padding and Truncating .....	31

14.Retun Code .....	31
15.File Conversion .....	31
16.Performance and Tuning .....	32
17.Limits .....	33
18.Errors and Warnings .....	33
19.GCSort by examples .....	34
19.1.SORT .....	34
19.2.MERGE .....	34
19.3.COPY .....	35
19.4.SUMFIELDS .....	35
19.5.OUTREC.....	36
19.6.OUTFIL .....	37
19.7.INREC/OUREC CHANGE .....	37
19.8.DATE.....	38
19.9.RECORD CONTROL STATEMENT .....	38
19.10.DATE – Option Y2PAST .....	38

# 1. Introduction

## 1.1.What is GCSort?

This document describes the features of the GCSORT utility.

GCSORT is an open-source tool for operations of sort/merge/copy files (Line Sequential, Sequential, Indexed and Relative) produced by GNUCobol compiler.

The principal developers of GCSORT are Cedric Issaly and Sauro Menna.

This document was intended to serve as a full-function reference and user's guide for GCSORT utility.

## 2. Features

Version 1.3.2 of GCSort contains a follow constructs:

---

gcsort help	
gcsort is a program to sort, merge and copy records in a file into a specified order	

---

Syntax case insensitive	
Return code : 0 (ok) - 4 (warning) - 16 (error)	

---

Usage with file parameters	: gcsort <options> take filename
Usage from command line	: gcsort <options> <control statements>

---

gcsort options	
-fsign=[ASCII EBCDIC] define display sign representation	

---

gcsort control statements	
Notations: '{name}' = parameters , ' ' = Alternative format of control statement	

---

SORT   MERGE   COPY	
FIELDS Control statement for Sort or Merge file(s)	
USE	Declare input file(s)
GIVE	Declare output file
[ SUM FIELDS ]	Sum fields for same record key, or eliminate duplicate keys)
[ RECORD ]	Record control statement
[ INCLUDE ]	Select input records that respect include condition(s)
[ OMIT ]	Omit input records that respect include condition(s)
[ INREC ]	Reformat input record Before sort, merge or copy operation
[ OUTREC ]	Reformat input record After sort, merge or copy operation
[ OUTFIL ]	Create one or more output files for sort,merge or copy operation
[ OPTION ]	Specifies option for control statements

---

gcsort	
SORT   MERGE   COPY	
FIELDS({Pos},{Len},{FormatType},{Order}, ...)	
FIELDS({Pos},{Len},{Order}, ...),FORMAT={FormatType}	
FIELDS=COPY	
USE {Filename}	
ORG {Org}	

---

```

RECORD [F,{RecordLen}] | [V,{MinLen},{MaxLen}]
      [KEY ({Pos},{Len},{KeyType})]

```

GIVE same parameters of USE

```

SUM FIELDS = [{Pos},{Len},{FormatType2}, ...] |
             [{Pos},{Len}, ...],FORMAT={FormatType2} |
             [NONE] | [(NONE)]

```

```

RECORD [TYPE=[{V} (Variable-length)/{F} (Fixed-length)], [LENGTH=[{len} (L1-Input record
length) ]

```

```

                                     ', ' [{len} (L2-Record
length) ]

```

```

                                     ', ' [{len} (L3-Output
record length) ]

```

```

INCLUDE | OMIT
      COND=({Condition})[,FORMAT={FormatType}]

```

```

INREC   FIELDS | INREC   BUILD =({FieldSpec})
INREC   OVERLAY =({FieldSpec})
OUTREC  FIELDS | OUTREC  BUILD =({FieldSpec})
OUTREC  OVERLAY =({FieldSpec})

```

OUTFIL

```

INCLUDE | OMIT ({Condition})[,FORMAT={FormatType}]
OUTREC = ({FieldSpec})
FILES/FNAMES= {Filename} | (file1, file2, file3,...)
STARTREC={nn}      Start from record nn
ENDREC={nn}        Skip record after nn
SAVE
SPLIT              Split 1 record output for file group (file1, file2, file3,...)
SPLITBY={nn}       Split n records output for file group (file1, file2, file3,...)

```

OPTION

```

SKIPREC={nn}      Skip nn records from input
STOPAFT={nn}      Stop read after nn records
VLSCMP            0 disabled , 1 = enabled -- temporarily replace any
                  missing compare field bytes with binary zeros
VLSHRT            0 disabled , 1 = enabled -- treat any comparison
                  involving a short field as false
Y2PAST            (YY) - Sliding, (YYYY) century
MODS E15=(<name>) [,]      <name>= Name E15 Cobol Program for input
E35=(<name>)             <name>= Name E35 Cobol Program for ouput

```

{Parameters}	{Relational}
{FileName} = Filename or Env. Variable	EQ = Equal
{Pos} = Field Position	GT = GreaterThan
{Len} = Field Length	GE = GreaterEqual
{RecordLen}= Record Length	LT = LesserThan
{MinLen} = Min size of record	LE = LesserEqual
{MaxLen} = Max size of record	NE = NotEqual
{Order} = A(ascending)   D(descending)	SS = SubString (only for Field Type 'CH')

{Condition}
Format 1 - (Pos,Len,{FormatType},{Relational},[AND OR],Pos,Len,{FormatType})
Format 2 - (Pos,Len,{FormatType},{Relational},[X C'[value]']   numeric value)]
Format 3 - ( {Condition} ,[AND OR],{Condition} )
Format 4 - ( Pos,Len,{FormatType},{Relational}, [DATE1][(+/-)num]   [DATE2][(+/-)num] [DATE3][(+/-)num]   [DATE4][(+/-)num]
DATE - Current Date : DATE1 (C'yyyymmdd'), DATE2 (C'yyyymm')

DATE3 (C'yyyyddd'), DATE4 (C'yyyy-mm-dd') (no Timestamp)  
 [(+/-)num] [+num] future date, [-num] past date)

{Org} File Organization			{KeyType} Mandatory for ORG = IX		
LS	=	Line Sequential	P	=	Primary Key
SQ	=	Sequential Fixed or Variable	A	=	Alternative Key
IX	=	Indexed Fixed or Variable	D	=	Alternative Key with Duplicates
RL	=	Relative Fixed or Variable	C	=	Continue definition
{FormatType} Field Format Type			{FormatType2} Format Type SumField		
CH	=	Char	BI	=	Binary unsigned
BI	=	Binary unsigned	FI	=	Binary signed
FI	=	Binary signed	FL	=	Floating Point
FL	=	Floating Point	PD	=	Packed
PD	=	Packed	ZD	=	Zoned
ZD	=	Zoned	CLO	=	Numeric sign leading
CLO	=	Numeric sign leading	CSL	=	Numeric sign leading separate
CSL	=	Numeric sign leading separate	CST	=	Numeric sign trailing separate
CST	=	Numeric sign trailing separate			
Format_Len_Type Date field			Format_Len_Type Date field		
Y2T	=	8 ZD CCYYMMDD	Y2D	=	1 PD YY
Y2T	=	4 ZD YYXX	Y2P	=	2 PD YY
Y2T	=	2 ZD YYX	Y2U	=	3 PD YYDDD
Y2T	=	3 ZD YY	Y2S	=	2 ZD YY
Y2T	=	5 ZD YYDDD	Y2V	=	4 PD YYMMDD
Y2T	=	6 ZD YYMMDD	Y2X	=	3 PD DDDYY
Y2B	=	1 BI YY	Y2Y	=	4 PD MMDDYY
Y2C	=	2 ZD YY	Y2Z	=	2 ZD YY
{FieldSpec} Field Specification					
pos, len		pos = position input record, len = length of field			
posOut:pos,len		posOut = position output, pos = position input , len = length			
n:X		Filling with Blank character from last position to n (absolute position of output record).			
n:Z		Filling with zero Binary character from last position to n (absolute position of output record).			
C'constant'		constant character value.			
nC'constant'		repeat n times constant character value.			
nX		repeat n times Blank character.			
nZ		repeat n times Binary (0x00) character.			
X'hh...hh'		hexadecimal characters.			
nX'hh...hh'		repeat n times hexadecimal characters.			
CHANGE=(vlen,[C   X]'<valueFind>',[C   X]'<valueSet>',...),NOMATCH=([C					
X]'<valueSet>)					
CHANGE=(vlen,[C   X]'<valueFind>', posIn, lenIn), NOMATCH = (posIn, posLen)					

#### Environment Variables

COB_VARSEQ_FORMAT	Used by GnuCOBOL
GCSORT_DEBUG	0 no print info, 1 info DEBUG, 2 for info Parser
GCSORT_MEMSIZE	Memory Allocation in byte (Default 512000000 byte)
GCSORT_PATHTMP	Pathname for temporary files (Default TMP / TEMP / TMPDIR)
GCSORT_STATISTICS	0 minimal informations, 1 for Summary, 2 for Details
GCSORT_TESTCMD	0 for normal operations , 1 for ONLY test command line (NO SORT)



### 3. Environment and first use

GCSort is a executable program written in 'C'.

Dependencies of executable GCSort are:

- **libcob** - GNUCobol
- **libm** - Math library

#### 3.1. Following the steps for the first use

- Make executable gcsort
- Set environment variable to find library at runtime
- Run *gcsort <option> <command line>*
  - o *<option> -fsign=[EBCDIC | ASCII]*

The *-fsign=EBCDIC* option can be used for files with ZONED fields and EBCDIC sign.

#### 3.2. Modify first environment variables

- Set Memory Allocation (GCSORT\_MEMSIZE)
- Set Statistics (GCSORT\_STATISTICS) to view details of execution

#### 3.3. Use TAKE command

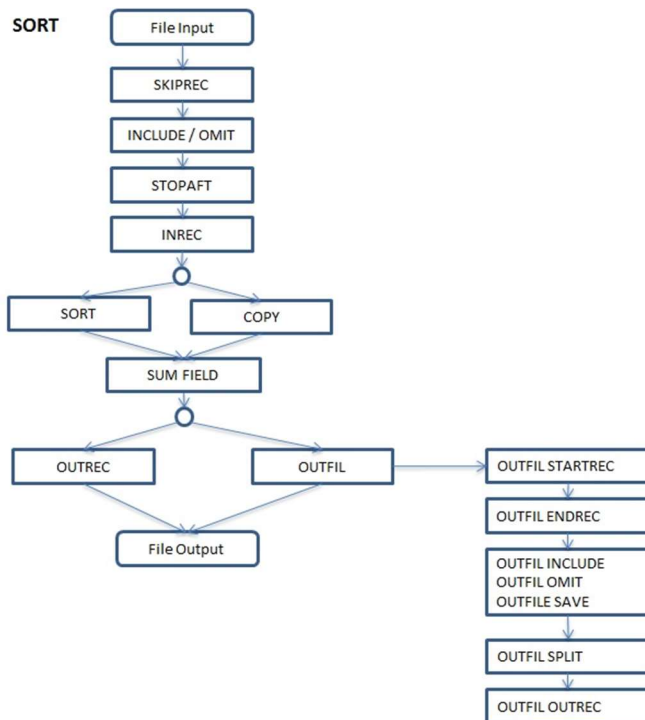
- Create file text
- Insert command. Single row o one row for command.
- In the file TAKE the '\*' character indicates that the rest of the line is treated as a comment
- Run : *gcsort TAKE filename*

Example to create TAKE file with script sh.

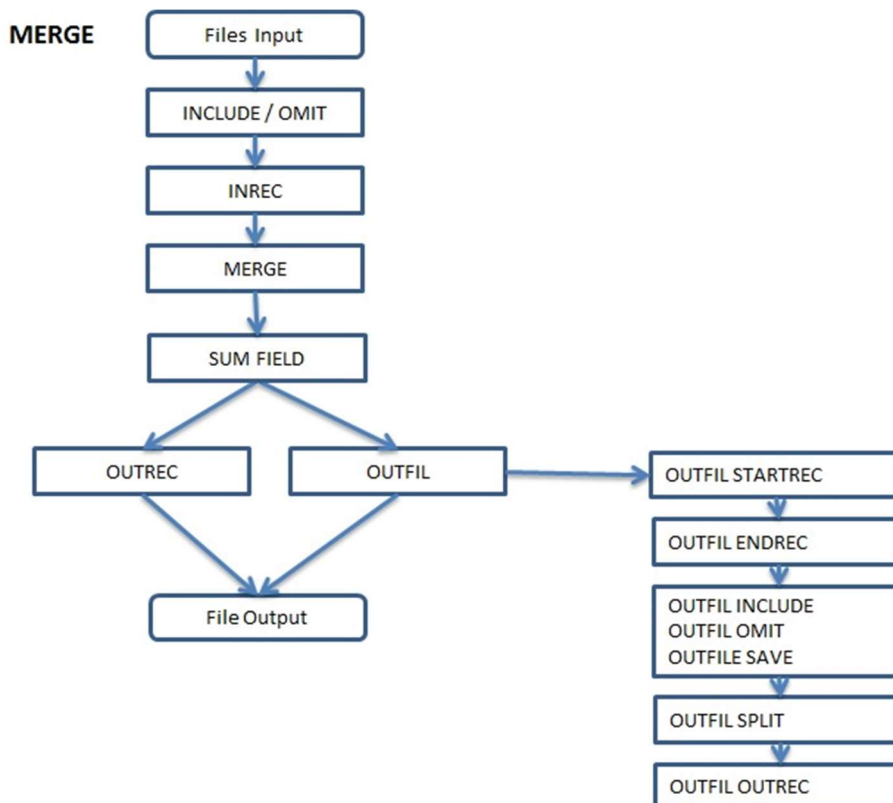
```
export LD_LIBRARY_PATH=/usr/local/lib
export GCSORT_MEMSIZE=1024000000
export GCSORT_BYTEORDER=0
export GCSORT_STATISTICS=2
echo "      * This is comment "                >TAKEFILE.PRM
echo "SORT FIELDS(4,1,CH,A) "                  >TAKEFILE.PRM
echo "SUM  FIELDS=(1,2,ZD,4,2,ZD,7,4,ZD,12,4,ZD) " >>TAKEFILE.PRM
echo "USE  ../files/SQZD03 RECORD F,396 ORG SQ   " >>TAKEFILE.PRM
echo "GIVE ../files/SQZD03.SRT RECORD F,396 ORG SQ " >>TAKEFILE.PRM
../bin/gcsort TAKE TAKEFILE.PRM
```

## 4. Process Schema

This picture show logical schema of utility GCSort for SORT operations.



This picture show logical schema of utility GCSort for MERGE operations.



## 5. Sort

The purpose of SORT is read one or more files and create a output file with data ordered as indicated by the sort key fields.

## 6. Merge

The purpose of MERGE is read one or more files and create a output file with data ordered as indicated by the merge key fields.

It is mandatory that the input data is already sorted.

## 7. File Organization and Record Type

File organization identifies the type of file.

The types of file organization utility managed GCSORT are:

**LS** = Line Sequential

**SQ** = Sequential

**IX** = Indexed

**RL** = Relative

Record type identifies the record structure

Record type are

**F** = Fixed

**V** = Variable (first n byte record len, verify COB\_VARSEQ\_FORMAT in GNUCobol )

## 8. Field Type

Field type detects typology of field, Field type used are:

Type	Description
<b>CH</b>	Char
<b>BI</b>	Binary unsigned
<b>FI</b>	Binary signed
<b>FL</b>	Floating Point
<b>PD</b>	Packed
<b>ZD</b>	Zoned
<b>CLO</b>	Numeric sign leading
<b>CSL</b>	Numeric sign leading separate
<b>CST</b>	Numeric sign trailing separate

## Date Format

Field Formats and Lengths for date.

Format	Len	Type	Date field		Format	Len	Type	Date field
Y2T	= 8	ZD	CCYYMMDD		Y2D	= 1	PD	YY
Y2T	= 4	ZD	YYXX		Y2P	= 2	PD	YY
Y2T	= 2	ZD	YYX		Y2U	= 3	PD	YYDDD
Y2T	= 3	ZD	YY		Y2S	= 2	ZD	YY
Y2T	= 5	ZD	YYDDD		Y2V	= 4	PD	YYMMDD
Y2T	= 6	ZD	YYMMDD		Y2X	= 3	PD	DDYY
Y2B	= 1	BI	YY		Y2Y	= 4	PD	MMDDYY
Y2C	= 2	ZD	YY		Y2Z	= 2	ZD	YY

```
[ DATE - Currente Date : DATE4 ]
INCLUDE COND=(1,13,CH,GT,DATE4)
USE ../files/inp5000.txt ORG LS RECORD F,5000
GIVE ../files/inp5000.txt.srt ORG LS RECORD F,5000
SORT FIELDS=(35,5,ZD,A)

[ DATE + / - day - month ]
COND=(1,13,CH,GT,DATE1+5)
COND=(1,13,CH,GT,DATE1-5)
COND=(1,13,CH,GT,DATE2+3)
COND=(1,13,CH,GT,DATE2-8)
COND=(1,13,CH,GT,DATE3+150)
COND=(1,13,CH,GT,DATE3-15)

[ DATE4 ]
OMIT COND=(1,13,CH,GT,DATE4)
```

## 9. Commands

### 9.1. SORT

SORT is command for ordering data.

**Format 1**      **SORT**

### 9.2. MERGE

MERGE is command for merging data.

**Format 1**      **MERGE**

### 9.3. COPY

In SORT or MERGE command **FIELDS=COPY** copy data from input to output file.

**Format 1**      **FIELDS=COPY**

### 9.4. FIELDS

This command specify fields for sort/merge operations. The fields are the key for order or merging data from files.

**Format 1**                      FIELDS (pos,len,type,order, ...)                      |  
**Format 2**                      FIELDS ((pos,len, order, ...),FORMAT=TYPE                      |  
**Format 3**                      FIELDS=COPY

[FIELDS \(pos, len, type, order,...\)](#)

---

- pos**      specifies the first byte of a control field relative to the beginning of the input record.  
The first data byte of a fixed-length record has relative position 1.  
The first data byte of a variable-length record has relative position 1.
- len**      specifies the length of the field. Values for all fields must be expressed in integer numbers of bytes.
- type**      specifies the format of the data of field.

Type	Description
CH	Char
BI	Binary unsigned
FI	Binary signed
FL	Floating Point
PD	Packed
ZD	Zoned
CLO	Numeric sign leading
CSL	Numeric sign leading separate
CST	Numeric sign trailing separate

**order** specifies how the field is to be ordered. The valid codes are:

**A** ascending order

**D** descending order

***FIELDS ((pos,len,order, ...),FORMAT=type***

---

**FORMAT=type** can be used to specify a particular format for one or more control fields. f from FORMAT=f is used for p,m,s fields.

***FIELDS=COPY or FIELDS=(COPY)***

---

Causes GCSORT to copy a file input to the output data sets. Records can be edited INCLUDE/OMIT, INREC, OUTREC, and OUTFIL statements; and SKIPREC and STOPAFT parameters.

## 9.5.USE

USE command declare input file for SORT and MERGE operation.

Format for USE:

USE <filename > ORG <organization> RECORD [<record format>,< length>]  
[KEY ({Pos},{Len},{KeyType})]

USE <filename > ORG <organization> RECORD [<record format>, <lenght min>,< length max>]  
[KEY ({Pos},{Len},{KeyType})]

**filename** Input file name, with or without pathname

**organization** **LS** = Line Sequential

**SQ** = Sequential

**RL** = Relative

**IX** = Indexed

**record format** **F** = Fixed

**V** = Variable

**length** Length of record

**length min** Minimum length of record

**length max** Maximum length of record

Structure of key (Mandatory for ORG = IX)

**Pos** Position of key

**Len** Length of key

**KeyType** **P** = Primary Key

**A** = Alternative Key

**D** = Alternative Key with Duplicates

C = Continue definition

## 9.6.GIVE

GIVE command declare output file for SORT and MERGE operation.

Same rules of USE control statement.

Format for GIVE:

GIVE <filename> ORG <organization> RECORD [<record format>,< length>]  
[KEY ({Pos},{Len},{KeyType})]

GIVE <filename> ORG <organization> RECORD [<record format>,<length min>,< length max>]  
[KEY ({Pos},{Len},{KeyType})]

## 9.7.INCLUDE/OMIT

INCLUDE condition statement is used for **select** records to insert in the file output.

OMIT condition statement is used for **exclude** certain records from the file input.

**INCLUDE/OMIT COND=(condition) [FORMAT=type]**

**condition**

Format 1 (pos , len , type , cond, pos , len , type)

Format 2 (pos , len , type , cond, [X|C|Z]"value"])

Format 3 (condition , relcond , condition)

*Format 1 (pos , len , type , cond, relcond , pos , len , type)*

---

**pos** specifies the first byte of a control field relative to the beginning of the input record.

The first data byte of a fixed-length record has relative position 1.

The first data byte of a variable-length record has relative position 1.

**len** specifies the length of the field. Values for all fields must be expressed in integer numbers of bytes.

**type** specifies the format of the data of field.

Type	Description
CH	Char
BI	Binary unsigned
FI	Binary signed
FL	Floating Point
PD	Packed
ZD	Zoned
CLO	Numeric sign leading
CSL	Numeric sign leading separate
CST	Numeric sign trailing separate

**cond** Comparison operators are as follows:

**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  
**SS** SubString

---

**Format 2**    *(pos , len , type , cond, [X|C]'[value]')|[/-nnnn]*

---

**pos** specifies the first byte of a control field relative to the beginning of the input record.  
 The first data byte of a fixed-length record has relative position 1.  
 The first data byte of a variable-length record has relative position 1.  
**len** specifies the length of the field. Values for all fields must be expressed in integer numbers of bytes.  
**type** specifies the format of the data of field.

Type	Description
CH	Char
BI	Binary unsigned
FI	Binary signed
FL	Floating Point
PD	Packed
ZD	Zoned
CLO	Numeric sign leading
CSL	Numeric sign leading separate
CST	Numeric sign trailing separate

**cond** Comparison operators are as follows:  
 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

**C'cc...c'**      **Character String Format** . The value c is a ASCII character/string.

**X'hh..hh'**      **Hexadecimal String Format**. The value hh represents any pair of hexadecimal digits.

**+/- nnnn..**      **Decimal Number Format**

---

**Format 3**    *(condition , relcond , condition)*

---

**condition**      Format 1 or Format 2

**relcond**      Relational conditions can be logically combined, with AND or OR.  
 The relational condition specifies that a comparison test be performed.  
 Relational conditions can be logically combined, with AND or OR.



*Format 4      (pos, len , CHANGE=(vlen, [X|C]'[value Find]', [X|C]'[value Set]' ....  
NOMATCH=([X|C]'[value]')*

---

CHANGE Specifies how the input field or parsed input field is to be changed to the output field, using a lookup table.

NOMATCH if an input field value does not match any of the find constants, NOMATCH values is used for output field.

*Format 5      (pos, len , CHANGE=(vlen, [X|C]'[value Find]', posFind,  
lenFind .... NOMATCH=(posNoMatch, lenNomatch)*

---

CHANGE Specifies how the input field or parsed input field is to be changed to the output field, using position(*posFind*) and length(*lenFind*) of input record.

NOMATCH if an input field value does not match any of the find constants, NOMATCH input record *position* and *length* are used for output field.

## 9.8.INREC/OUTREC

INREC redefines the structure of record input. This operation is executed after read file input e before all operations.

The INREC control statement reformat the input records **before** they are sorted, merged, or copied. All fields specifications presents in OUTREC, Sort Key, ... must be referred to a new structure defined by INREC.

**Format 1**      INREC FIELDS=(FIELD-SPEC...)  
**Format 2**      INREC BUILD=(FIELD-SPEC...)  
**Format 3**      INREC OVERLAY=(FIELD-SPEC...)  
**Format 4**      INREC FINDREP=(FIELD-FINDEREP-SPEC

OUTREC defines structure record output for output file.

**Format 1**      OUTREC FIELDS=(FIELD-SPEC...)  
**Format 2**      OUTREC BUILD=(FIELD-SPEC...)  
**Format 3**      OUTREC OVERLAY=(FIELD-SPEC...)  
**Format 4**      INREC FINDREP=(FIELD-FINDEREP-SPEC

Use **OVERALY** only to overwrite existing columns or to add fields at end of every record.

Field specification is the same for INREC and OUTREC.

**BUILD** or **FIELDS** are synonymous.

**FIELD-SPEC** ( **pos, len** | **posOut:pos,len** | **n:X** | **n:Z** | **nC'constant'** | **nX** | **nZ**, |**X'hh'** )

One or more occurrence of follow elements, separated by comma.

<b>pos, len</b>	<b>pos</b> = position input record, <b>len</b> = length of field
<b>posOut:pos,len</b>	<b>posOut</b> = position output, <b>pos</b> = position input record, <b>len</b> = length of field
<b>n:X</b>	Filling with Blank character (0x20) from last position to <b>n</b> (absolute position of output record).
<b>n:Z</b>	Filling with zero Binary (0x00) character from last position to <b>n</b> (absolute position of output record).
<b>C'constant'</b>	constant character value.
<b>nC'constant'</b>	repeat <b>n</b> times constant character value.
<b>nX</b>	repeat <b>n</b> times Blank character.
<b>nZ</b>	repeat <b>n</b> times Binary (0x00) character.
<b>X'hh...hh'</b>	hexdecimal string .
<b>nX'hh...hh'</b>	repeat <b>n</b> times hexdecimal string .

---

**FIELD-FINDREP-SPEC\_\_Field Find/Replace Specification**

<b>IN=C'constant'</b> , <b>OUT=C'constant'</b>	<b>constant character value.</b>
<b>IN=(C'constant'</b> , <b>C'constant'</b> ....) , <b>OUT=C'constant'</b>	<b>constant character value.</b>
<b>INOUT=(C'constantIn'</b> , <b>C'constantOut'</b> , <b>C'constantIn'</b> , <b>C'constantOut'</b> , ....)	
<b>STARTPOS=pos</b>	<b>pos</b> = Start Position to find/replace
<b>ENDPOS=pos</b>	<b>pos</b> = End Position to find/replace
<b>DO=n</b>	<b>n</b> =Maximum number of times find and replace
<b>MAXLEN=n</b>	<b>n</b> =Maximum len of record <b>n</b>
<b>OVERRUN=TRUNC ERROR</b>	Truncate or Error(Default) for overrun
<b>SHIFT=YES NO</b>	Shift data or no (default) when different length between find replace

---

## 9.9.SUM FIELDS

SUM FIELDS is command for aggregate record and summarize value for numeric fields.  
All fields present in SUM FIELDS are aggregate when more records has same key.

**Format 1** SUM FIELDS = (pos,len,type, ...)

**Format 2** SUM FIELDS = (NONE) or SUM FIELDS = NONE

There are two formats for SUM FIELD, the first summarize numeric fields, the second NOT summarize, but eliminate duplicate key.

### *Format 1 SUM FIELDS = (pos,len,type, ...)*

---

**pos** specifies the first byte of a control field relative to the beginning of the input record.  
The first data byte of a fixed-length record has relative position 1.

The first data byte of a variable-length record has relative position 1.

**len** specifies the length of the field. Values for all fields must be expressed in integer numbers of bytes.

**type** specifies the format of the data of field.

Type	Description
BI	Binary unsigned
FI	Binary signed
FL	Floating Point
PD	Packed
ZD	Zoned
CLO	Numeric sign leading
CSL	Numeric sign leading separate
CST	Numeric sign trailing separate

### *Format 2 SUM FIELDS = (NONE) or SUM FIELDS = NONE*

---

In this case Format2 insert into output file one occurrence of same key specified by SORT KEY.

The record output contains the first record in order of reading.

For identify a first occurrence of data, GCSORT verified the value of pointer of record into file input, selecting the lowest value.

## 9.10. RECORD

RECORD control statement is option to specify the type and lengths of the records.

RECORD [TYPE=[{V}/{F}{Fixed-length}]] , [LENGTH=[{len}{L1-Input record length}]

,[{len}{L2-Record length}]

,[{len}{L3-Output record length}]

TYPE = V (Variable-length) / F (Fixed-length)

LENGTH = (L1, L2, L3)

L1 = Input length

L2 = Record length after E15

L3 = Output record length

L1 is ignored if the input record length is available from USE command.  
 L2 is ignored if E15 is not used.  
 L3 is ignored if the input record length is available from GIVE command.

Example:

**[ RECORD CONTROL STATEMENT ]**

```

SORT FIELDS=(8,5,CH,A) USE ../files/sqbig01.dat  ORG SQ GIVE ../files/sqbig01_gcs.srt  ORG SQ RECORD TYPE=F, LENGTH=500

RECORD TYPE=F, LENGTH=(500)

RECORD TYPE=F, LENGTH=(500, ,500)

RECORD TYPE=F LENGTH=(, ,500)

RECORD TYPE=F,LENGTH=(, ,500)
  
```

## 9.11. OUTFIL

OUTFIL is command to create one or more output file for a sort, copy, or merge operation.  
 Each file output is defined from OUTFIL command

### FORMAT

#### OUTFIL

FILES/FNAMES= (environment variable)  
 STARTREC=nn  
 ENDREC=nn  
 [SAVE|[INCLUDE|OMIT] (CONDITION) [FORMAT=TYPE]]  
 SPLIT  
 OUTREC = (FIELD-SPEC...)

#### OUTFIL

FILES/FNAMES=filename	filename = Identify a environment variable the contain the file name
STARTREC=nn	Start write after <b>nn</b> records
ENDREC=nn	Stop write after <b>nn</b> records
SAVE	Save records that not used by command INCLUDE/OMIT.
INCLUDE/OMIT (CONDITION) [FORMAT=TYPE]]	Same definition for COND-FIELD (INCLUDE/OMIT)
SPLIT	Split 1 record for each File in Group definition (FILE=file1,file,file2)
SPLITBY=n	Split n records for each File in Group definition (FILE=file1,file,file2)
OUTREC = (FIELD-SPEC...)	Define structure output data. Same definition for (FIELD-SPEC...).

If the environment variable filename for FILES/FNAMES is not defined, GCSort writes output file in local folder assuming the name equal at value of identifier filename (FILES/FNAMES=*filename*).

## 9.12. OPTION

This command allows you to change the behavior of the utility.

Format1 **OPTION** [SKIPREC=nn][**STOPAFT**=nn][**VLSCMP**][**VLSHRT**] | [**Y2PAST**=YY] | [YYYY]

<b>SKIPREC</b> =nn	Skip nn records from input
<b>STOPAFT</b> =nn	Stop read after nn records
<b>VLSCMP</b>	0 disabled , 1 = enabled -- temporarily replace any missing compare field bytes with binary zeros
<b>VLSHRT</b>	0 disabled , 1 = enabled -- treat any comparison involving a short field as false
<b>Y2PAST</b> =YY	(YY) – Sliding = Numbers of years to subtract from the current year.

year.

(YYYY) – Century= Specifies the beginning of the fixed century window.

**MODS** [E15 =( <name> )] [E35 =( <name> )] Routine name E15 and/or E35 Cobol Program.

## Exit Routines

### E15 – Routine called after file read

E15 routine is a COBOL program.

Linkage :

```

LINKAGE      for fixed records
01  RECORD - FLAGS          PIC 9(8) BINARY.
    88  FIRST - REC          VALUE 00.
    88  MIDDLE - REC         VALUE 04.
    88  END - REC            VALUE 08.
01  NEW-REC                 PIC X(nn) .
01  RETURN-REC              PIC X(nn) .
01  UNUSED1                 PIC 9(8) BINARY.
01  UNUSED2                 PIC 9(8) BINARY.
01  NEW-REC-LEN             PIC 9(8) BINARY
(Only for Variable Length)
01  RETURN-REC-LEN          PIC 9(8) BINARY                      (Only
for Variable Length)
01  UNUSED5                 PIC 9(8) BINARY.
01  EXITAREA-LEN            PIC 9(4) BINARY.
01  EXITAREA.
    05  EAREA      OCCURS 1 TO 256 TIMES
                  DEPENDING ON EXITAREA-LEN  PIC X.
```

### E35 – Routine called before write output

E35 routine is a COBOL program.

```

LINKAGE      for fixed records
01  RECORD-FLAGS          PIC 9(8) BINARY.
    88  FIRST-REC          VALUE 00.
```

```

      88 MIDDLE-REC          VALUE 04.
      88 END-REC            VALUE 08.
01 LEAVING-REC.
      05 LREC OCCURS 1 TO 200 TIMES
          DEPENDING ON LEAVING-REC-LEN      PIC X.
01 RETURN-REC.
      05 RREC OCCURS 1 TO 200 TIMES
          DEPENDING ON RETURN-REC-LEN      PIC X.
01 OUTPUT-REC.
      05 OREC OCCURS 1 TO 200 TIMES
          DEPENDING ON OUTPUT-REC-LEN      PIC X.
01 UNUSED1                PIC 9(8) BINARY.
01 LEAVING-REC-LEN        PIC 9(8) BINARY.
01 RETURN-REC-LEN         PIC 9(8) BINARY.
01 OUTPUT-REC-LEN         PIC 9(8) BINARY.
01 EXITAREA-LEN           PIC 9(4) BINARY.
01 EXITAREA.
      05 EAREA OCCURS 1 TO 256 TIMES
          DEPENDING ON EXITAREA-LEN      PIC X.

```

#### **E15 - Return code**

00 - No Action  
 04 - Record deleted  
 08 - Do Not Return  
 12 - Record inserted  
 16 - Terminate DFSORT  
 20 – Record Altered or Replaced

#### **E35 – Return code**

00 - No Action  
 04 – Record deleted  
 08 - Do Not Return  
 12 - Insert record  
 16 - End of GCSort

## 10. JOIN Statement

The purpose of the JOIN statement is to perform JOIN between two files (F1 and F2). You can perform different types of join on two files (F1 and F2) by one or more keys with GCSort using the following statements:

### JOINKEYS

JOINKEYS specifies the definition of the JOIN key.

It is necessary to specify a JOINKEYS statement for each file, one for F1 and one for F2.

Each JOINKEYS statement must specify the starting position, the length and the sequence of the keys that file. You can also optionally specify if the file is already sorted by the keys and if sequence checking of the keys is not needed, or stop reading the file after n records.

### JOIN

JOIN tells gcsort how to match records in the JOIN command.

**Inner join** – Default, only paired records from F1 and F2 are processed.

**Left outer join** - Unpaired F1 records as well as paired records.

**Right outer join** - Unpaired F2 records as well as paired records.

**Full outer join** - unpaired F1 and F2 records as well as paired records.

**Unpaired F1,ONLY** - Only unpaired F1 records

**Unpaired F2,ONLY** - Only unpaired F2 records

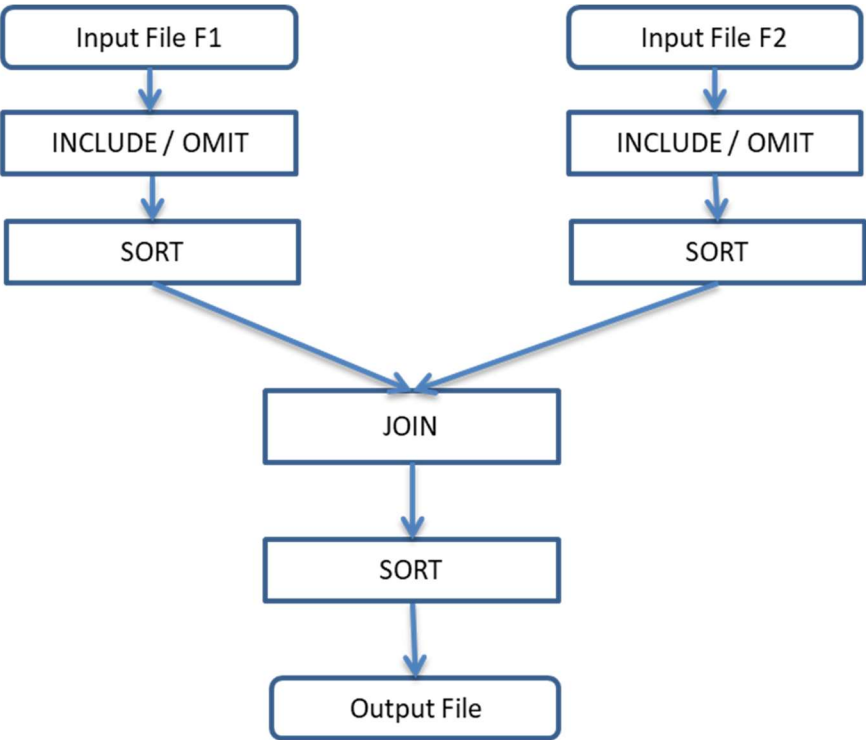
**Unpaired F1,F2,ONLY / Unpaired,ONLY**- Only unpaired F1 and F2 records

### REFORMAT

REFORMAT statement specified the fields of F1 and/or F2 in the joined records.

# Join Process Schema

**JOIN**





For details *gcsort --help JOIN*

`gcsort --help JOIN`

---

```
gcsort help
gcsort is a utility to sort, merge, copy and join records in a file into a
specified order in GnuCOBOL environment.
```

---

```
Syntax case insensitive
Return code : 0 (ok) - 4 (warning) - 16 (error)
```

---

```
Usage with file parameters : gcsort <options> take filename
Usage from command line   : gcsort <options> <control statements>
```

---

```
gcsort options
-fsign=[ASCII|EBCDIC] define display sign representation
```

---

```
=====
Section for JOIN control statement
=====
```

```
JOIN file(s)
  USE                Declare input file F1
  USE                Declare input file F2
  GIVE               Declare output file
  JOINKEYS FILES=F1.. Declare keys file F1
    [ INCLUDE]      Input file F1 - Select input records that respect include
condition(s)
    [ OMIT   ]      Input file F1 - Omit input records that respect omit condition(s)
  JOINKEYS FILES=F2.. Declare keys file F2
    [ INCLUDE]      Input file F2 - Select input records that respect include
condition(s)
    [ OMIT   ]      Input file F2 - Omit input records that respect omit condition(s)
  UNPAIRED           Declare join type
  REFORMAT FIELDS    Declare output format
  [ INCLUDE]         Output file - Select input records that respect include condition(s)
  [ OMIT   ]         Output file - Omit input records that respect omit condition(s)
  [ INREC  ]         Output file - Reformat input record before join operation
  [ OUTFIL ]         Output file - Create one or more output files from join operation
```

---

```
JOIN
USE {Filename}          [File F1]
  ORG {Org}
  RECORD [F,{RecordLen}] | [V,{MinLen},{MaxLen}]
    [KEY ({Pos},{Len},{KeyType})]

USE {Filename}          [File F2]
  ORG {Org}
  RECORD [F,{RecordLen}] | [V,{MinLen},{MaxLen}]
    [KEY ({Pos},{Len},{KeyType})]

GIVE same parameters of USE

JOINKEYS FILES=F1,FIELDS=[({Pos},{Len},{Order}, ...)]
    [,SORTED] [,STOFAFT={nn}]
    [, INCLUDE ] | [, OMIT]
    [ COND=({Condition})[,FORMAT={FormatType}] ]

JOINKEYS FILES=F2,FIELDS=[({Pos},{Len},{Order}, ...)]
    [,SORTED] [,STOFAFT={nn}]
    [, INCLUDE ] | [, OMIT]
    [ COND=({Condition})[,FORMAT={FormatType}] ]

JOIN UNPAIRED [,F1][,F2][,ONLY]
  UNPAIRED,F1,F2 or UNPAIRED
    Unpaired records from F1 and F2 as well as paired records (Full outer join).
```

---

```
UNPAIRED,F1
    Unpaired records from F1 as well as paired records (Left outer join).
UNPAIRED,F2
    Unpaired records from F2 as well as paired records (Right outer join).
UNPAIRED,F1,F2,ONLY or UNPAIRED,ONLY
    Unpaired records from F1 and F2.
UNPAIRED,F1,ONLY
    Unpaired records from F1.
UNPAIRED,F2,ONLY
    Unpaired records from F2.
```

```

      REFORMAT FIELDS=({File}:{Pos},{Len},{?},{File}:{Pos},{Len}...) [,FILL=[C'constant'
| [X'hh']]

```

### Commands for output file

INCLUDE | OMIT

COND=({Condition})[,FORMAT={FormatType}]

```
INREC  FIELDS | INREC  BUILD =({FieldSpec})
```

```
INREC      OVERLAY =({FieldSpec})
```

```
OUTREC  FIELDS | OUTREC  BUILD =({FieldSpec})
```

```
OUTREC  OVERLAY =({FieldSpec})
```

OUTFIL

**INCLUDE | OMIT** ({Condition}) [, **FORMAT**={FormatType}]

```
OUTREC BUILD | BUILD = ({FieldSpec})
```

FILES/FNAMES= {Filename}

{Parameters}	{Parameters}
{File} = F1 or F2	? = 1-byte indicator joined record
{Pos} = Field Position	'B' = 'Both' - Key found in F1 and F2
{Len} = Field Length	'1' = Key found in F1, but not in F2
{Order} = A(ascending)   D(descending)	'2' = Key found in F1, but not in F1
C'Constant' = Character fill byte	nn = Numbers of records from input file
X'hh' = Hexadecimal fill byte (00-FF).	

{Parameters}	{Relational}
{FileName} = Filename or Env. Variable	EQ = Equal
{Pos} = Field Position	GT = GreaterThan
{Len} = Field Length	GE = GreaterEqual
{RecordLen}= Record Length	LT = LesserThan
{MinLen} = Min size of record	LE = LesserEqual
{MaxLen} = Max size of record	NE = NotEqual
{Order} = A(ascending)   D(descending)	SS = SubString (only for Field Type 'CH')

{Condition}	
Format 1	- (Pos,Len,{FormatType},{Relational},[AND OR],Pos,Len,{FormatType})
Format 2	- (Pos,Len,{FormatType},{Relational},[X C'[value]']   numeric value)
Format 3	- ( {Condition} ,[AND OR],{Condition} )
Format 4	- ( Pos,Len,{FormatType},{Relational}, [DATE1][(+/-)num]   [DATE2][(+/-)num] [DATE3][(+/-)num]   [DATE4][(+/-)num]
DATE	- Current Date : DATE1 (C'yyyymmdd'), DATE2 (C'yyyymm'), DATE3 (C'yyyddd'), DATE4 (C'yyyy-mm-dd') (no Timestamp) [(+/-)num] [+num] future date, [-num] past date) only for DATE1,DATE2,DATE3

{Org} File Organization	{KeyType} Mandatory for ORG = IX
LS = Line Sequential	P = Primary Key
SQ = Sequential Fixed or Variable	A = Alternative Key
IX = Indexed Fixed or Variable	D = Alternative Key with Duplicates
RL = Relative Fixed or Variable	C = Continue definition

{FormatType}_____Field Format Type_____	{FormatType2}_____Format Type SumField_____
CH = Char	BI = Binary unsigned
BI = Binary unsigned	FI = Binary signed
FI = Binary signed	FL = Floating Point
FL = Floating Point	PD = Packed
PD = Packed	ZD = Zoned
ZD = Zoned	CLO = Numeric sign leading

CLO = Numeric sign leading		CSL = Numeric sign leading separate
CSL = Numeric sign leading separate		CST = Numeric sign trailing separate
CST = Numeric sign trailing separate		

Format	Len	Type	Date field	Format	Len	Type	Date field
Y2T = 8	ZD		CCYYMMDD	Y2D = 1	PD		YY
Y2T = 4	ZD		YYXX	Y2P = 2	PD		YY
Y2T = 2	ZD		YYX	Y2U = 3	PD		YYDDD
Y2T = 3	ZD		YY	Y2S = 2	ZD		YY
Y2T = 5	ZD		YYDDD	Y2V = 4	PD		YYMMDD
Y2T = 6	ZD		YYMMDD	Y2X = 3	PD		DDYY
Y2B = 1	BI		YY	Y2Y = 4	PD		MMDDYY
Y2C = 2	ZD		YY	Y2Z = 2	ZD		YY

{FieldSpec}	Field Specification
pos, len	pos = position input record, len = length of field
posOut:pos,len	posOut = position output, pos = position input , len = length
n:X	Filling with Blank character from last position to n (absolute position of output record).
n:Z	Filling with zero Binary character from last position to n (absolute position of output record).
C'constant'	constant character value.
nC'constant'	repeat n times constant character value.
nX	repeat n times Blank character.
nZ	repeat n times Binary (0x00) character.
X'hh...hh'	hexdecimal characters.
nX'hh...hh'	repeat n times hexdecimal characters.
CHANGE=(vlen,[C X]'<valueFind>',[C X]'<valueSet>',....),NOMATCH=([C X]'<valueSet>')	
CHANGE=(vlen,[C X]'<valueFind>', posIn, lenIn), NOMATCH = (posIn, posLen)	

#### Environment Variables

COB_VARSEQ_FORMAT	Used by GnuCOBOL
GCSORT_DEBUG	0 no print info, 1 info DEBUG, 2 for info Parser
GCSORT_MEMSIZE	Memory Allocation in byte (Default 512000000 byte)
GCSORT_PATHTMP	Pathname for temporary files (Default TMP / TEMP / TMPDIR)
GCSORT_STATISTICS	0 minimal informations, 1 for Summary, 2 for Details
GCSORT_TESTCMD	0 for normal operations , 1 for ONLY test command line (NO SORT)

## 11. Environment Variables

### 11.1. Byte Order

GCSort can treat numeric fields in both binary format BigEndian or Native. To indicate a byte order is used environment variable GCSORT\_BYTEORDER that assume 0 for Native or 1 for BigEndian. This value affects the treatment of SORT and SUM KEY FIELDS.

### 11.2. Temporary Files

When dimension of files input is greater of memory available, GCSort creates temporary files for sort operation. Temporary files is created in pathname specified from GCSORT\_TMPFILE environment variable, if this value is not available, GCSort use TMP/TEMP environment variable or use current directory. For Windows the filename is composed from:

- Prefix = Srt
- Name = name ( created from GetTempFileName())
- Extension = .tmp
- 

For Linux file name is composed from:

- Prefix = Srt
- Name = PID of process GCSort
- Num = Progressive of file
- Extension = .tmp

Temporary files are destroyed after sort operation.

### 11.3. Memory Allocation

The environment variable GCSORT\_MEMSIZE specify amount of memory that GCSORT will use for sort operation.

GCSort analyze the value and made two area for sort operation:

- (1) Key Area : this area is used for sort in memory
- (2) Data Area : this area contains data record

The optimization for use of memory GCSort check dimension of key and record.

Key Area =  $[GCSORT\_MEMSIZE] * ((Key\ Length + 8 + 4 + 8) / Record\ Length)$

Data Area =  $[GCSORT\_MEMSIZE] - Key\ Area$

$(8 + 4 + 8)$  8 is pointer of record into file, 4 record length, 8 pointer to record area in memory.

If value of  $((Key\ Length + 8 + 4 + 8) / Record\ Length)$  is minor of 15% or major of 50%, GCSORT force this value to 15%.

## 11.4. Statistics

GCSort produce in output a lot of information about execution.

You can setting GCSORT\_STATISTICS environment variable to three values:

### 0 = minimal information

Example:

```
=====
GCSort Version 01.00.00
=====
TAKE file name
D:\GNU_COBOL\GCSort_1_0_0\gcsort_testcase\take\par_SORT_debug.par
=====
File : D:\GCSORTTEST\OCFILES\TEST9\INP000.txt
Size : 1194
=====
Record Number Total      : 15
Record Write Sort Total  : 0
Record Write Output Total : 15
=====
Start   : Mon Jan 25 11:17:55 2016
End     : Mon Jan 25 11:17:55 2016
Elapsed Time 00hh 00mm 00ss 000ms

Sort OK
```

### 1 = medium information

Example

```
=====
GCSORT
File TAKE : D:\GNU_COBOL\GCSort_1_0_0\gcsort_testcase\take\par_SORT_debug.par
=====
SORT FIELDS(3,1,CH,A)
USE D:\GCSORTTEST\OCFILES\TEST9\INP000.txt ORG LS RECORD V,1,27990
GIVE D:\GCSORTTEST\OCFILES\TEST9\OUT000.SRT ORG LS RECORD V,1,27990

=====
GCSort Version 01.00.00
=====
TAKE file name
D:\GNU_COBOL\GCSort_1_0_0\gcsort_testcase\take\par_SORT_debug.par
=====
Operation : SORT

INPUT FILE :
      D:\GCSORTTEST\OCFILES\TEST9\INP000.txt VARIABLE (1,27990) LS
OUTPUT FILE :
      D:\GCSORTTEST\OCFILES\TEST9\OUT000.SRT VARIABLE (1,27990) LS
SORT FIELDS : (3,1,CH,A)
=====
File : D:\GCSORTTEST\OCFILES\TEST9\INP000.txt
Size : 1194
=====
Record Number Total      : 15
Record Write Sort Total  : 0
```

Record Write Output Total : 15

=====  
Start : Mon Jan 25 11:20:01 2016  
End : Mon Jan 25 11:20:01 2016  
Elapsed Time 00hh 00mm 00ss 000ms

Sort OK

## 2 = details information

=====  
GCSORT

File TAKE : D:\GNU\_COBOL\GCSort\_1\_0\_0\gcsort\_testcase\take\par\_SORT\_debug.par

=====  
SORT FIELDS(3,1,CH,A)

USE D:\GCSORTTEST\OCFILES\TEST9\INP000.txt ORG LS RECORD V,1,27990

GIVE D:\GCSORTTEST\OCFILES\TEST9\OUT000.SRT ORG LS RECORD V,1,27990

=====  
GCSort Version 01.00.00

=====  
TAKE file name

D:\GNU\_COBOL\GCSort\_1\_0\_0\gcsort\_testcase\take\par\_SORT\_debug.par

=====  
Operation : SORT

INPUT FILE :

D:\GCSORTTEST\OCFILES\TEST9\INP000.txt VARIABLE (1,27990) LS

OUTPUT FILE :

D:\GCSORTTEST\OCFILES\TEST9\OUT000.SRT VARIABLE (1,27990) LS

SORT FIELDS : (3,1,CH,A)

=====  
File : D:\GCSORTTEST\OCFILES\TEST9\INP000.txt

Size : 1194

After job\_loadFiles - Mon Jan 25 11:21:44 2016

After job\_sort - Mon Jan 25 11:21:44 2016

After job\_save - Mon Jan 25 11:21:44 2016

=====  
Record Number Total : 15

Record Write Sort Total : 0

Record Write Output Total : 15

=====  
Memory size for GCSort data : 133875000

Memory size for GCSort key : 23625000

BufferedReader MAX\_BUFFER : 4063232

MAX\_SIZE\_CACHE\_WRITE : 4063232

MAX\_SIZE\_CACHE\_WRITE\_FINAL : 4063232

MAX\_MLTP\_BYTE : 63

BYTEORDER : 0

=====  
Start : Mon Jan 25 11:21:44 2016

End : Mon Jan 25 11:21:44 2016

Elapsed Time 00hh 00mm 00ss 000ms

Sort OK

## 12. Command Line

GCSort command line accepts the following parameters:

<b>gcsort</b>	print version and options.
<b>gcsort --help</b>	print help.
<b>gcsort --help SORT   MERGE   COPY   JOIN</b>	print help for specific control statement.
<b>gcsort --version</b>	print version.
<b>gcsort --config</b>	print the value of environment variables.
<b>gcsort <i>command line</i></b>	execute command line.
<b>gcsort TAKE <i>filename</i></b>	read filename where are present commands for Sort/Merge.

The file used in the TAKE command is free format.

## 13. Padding and Truncating

GCSort uses LIBCOB that defines how made record in write output operation.

## 14. Return Code

GCSort has two values for return code:

0	for success
16	for failure

## 15. File Conversion

GCSort permit to specify 'ORGANIZATION' and 'RECORD TYPE' for output data different structure from input data, to permit the conversion of file format.

In this case GCSort convert data from a structure to another structure, for example, from Sequential to Line Sequential or vice versa.

If you want sort a text file (LS) and you don't know the record length, you can specify RECORD V with max len very large, example:

```
SORT KEY (1,20,CH,A)
USE F1.TXT ORG LS RECORD V,1,3000
GIVE F1.TXT.OUT ORG LS RECORD V,1,3000
```

## 16. Performance and Tuning

For tuning performance of GCSort is good practices modify the settings of value for memory allocation and modify dimension of area for Memory Mapped File.

**GCSORT\_MEMSIZE**      Indicate amount of memory for sort.

**GCSORT\_MLT**            Indicate the number of views for MMF in temporary files. This number is multiplied by Page Size of system (example 65536). Increasing this value the view for read file in memory is more greater and can reduce the elapsed time.(Temporary files).

By default GCSORT\_MLT assume 63 ( Example:  $63 * 65536 = 4\text{Mbyte}$  dimension of view for MMF).



## 17. Limits

The max numbers of input files for Merge is 16.

The max numbers of temporary files is 16. The temporary files is reused when the size of files input is more of size of (Memory GCSORT\_MEMSIZE \* 16 files).

## 18. Errors and Warnings

GCSORT produces two types of messages:

- Error           format '\*GCSort\*Snnn'
- Warning       format '\*GCSort\*Wnnn'

For Error message GCSort break execution and terminate operation with message and return code.

For Warning message GCSort continue execution and continue operation with message.

The message string identify a specific condition of error or warning, in the of warning print a specific action.

## 19. GCSort by examples

### 19.1. SORT

#### SORT single file

```
=====
SORT   FIELDS (3,1,CH,A)
USE     ../PJTestCaseSort/SQBI01          RECORD F,51 ORG SQ
GIVE    ../PJTestCaseSort/SQBI01.SRT.TST  RECORD F,51 ORG SQ
=====
```

#### SORT single file with INCLUDE condition

##### Order KEY

- 1) Position 37, Len 1, Character, Descending
- 2) Position 18, Len 17, Character, Ascending

Filter only records with character in position 37 Equal 'C'.

```
=====
SORT FIELDS=(37,1,CH,D,18,17,CH,A)
INCLUDE COND=(37,1,EQ,C'C') FORMAT=CH
USE   FIL_100.TXT          RECORD F,3000 ORG LS
GIVE  FIL_100.TXT.SRT      RECORD F,3000 ORG LS
=====
```

### 19.2. MERGE

#### MERGE

Merge files with KEY Position 1, Len 50, Char, Ascending

Input files sorted

Input Record Variable from 1 to 27990 ORGAnization Sequential

Output Record Variable from 1 to 27990 ORGAnization Sequential

```
=====
MERGE FIELDS(1,50,CH,A)
USE    D:\GCSORTTEST\OCFILES\RGX10.DAT    RECORD V,1,27990 ORG SQ
USE    D:\GCSORTTEST\OCFILES\RGX10.DAT    RECORD V,1,27990 ORG SQ
USE    D:\GCSORTTEST\OCFILES\RGX10.DAT    RECORD V,1,27990 ORG SQ
GIVE   D:\GCSORTTEST\OCFILES\RGX10.DAT.MRG RECORD V,1,27990 ORG SQ
=====
```

#### MERGE

FIELDS=COPY

Copy records from input to output.

Include condition check binary value (low-value)

Pos	Len	Condition	Value
from 305	04	Not Equal	Hex '00000000'

```
=====
USE D:\GCSORTTEST\FilesT\FIL_OUTFIL_500.TXT ORG LS RECORD F,3000
GIVE D:\GCSORTTEST\FilesT\FIL_OUTFIL_500_023.TXT.SRT ORG LS RECORD F,3000
OPTION VLSHRT,VLSCMP,EQUALS
=====
```

```
MERGE    FIELDS=COPY
INCLUDE  COND=(305,4,NE,X'00000000'),FORMAT=CH
```

### 19.3. COPY

#### COPY

Copy data from input to output with record filter.

Input FIXED Line Sequential, Output FIXED Line Sequential

Omitted (not insert in output file) records with condition:

- a) Position 1, Len 12, EQual , Character '000000006060'
- OR
- b) Position 1, Len 12, EQual , Character '000000000030'
- OR
- c) Position 1, Len 12, EQual , Character '000000000051'

```
=====
USE      F1IN.DAT                RECORD F,3000 ORG LS
GIVE     F1IN.DAT_002.SRT        RECORD F,3000 ORG LS
MERGE    FIELDS=COPY
OMIT     COND=(01,12,EQ,C'000000006060',OR,
              01,12,EQ,C'000000000030',OR,
              01,12,EQ,C'000000000051'),FORMAT=CH
=====
```

#### SORT without duplicates

Sort Key Pos 5, len 6, Ascending

SUM FIELDS = (NONE) delete duplicates

```
=====
USE     FIL_OUTFIL_100.TXT        ORG LS RECORD F,3000
GIVE    FIL_OUTFIL_100_020.TXT.SRT ORG LS RECORD F,3000
SORT    FIELDS=(5,6,A),FORMAT=CH,EQUALS
SUM     FIELDS=(NONE)
=====
```

### 19.4. SUMFIELDS

#### SUMFIELDS

Sort Key Pos 1, len 1, Ascending

SUM FIELDS Binary fields

```
=====
SORT    FIELDS(3,1,CH,A)
SUM     FIELDS=(1,2,BI,7,3,BI,15,4,BI,20,3,BI,29,4,BI,34,8,BI,43,8,BI)
USE     ../PJTestCaseSort/SQBI01 RECORD F,51 ORG SQ
GIVE    ../PJTestCaseSort/SQBI01.SRT.TST RECORD F,51 ORG SQ
=====
```

## 19.5. OUTREC

### OUTREC FIELDS/BUILD

SORT FIELDS = COPY (copy record NO Sort)

Format output : OUTREC

Output structure

Pos	Len	Value
01	16	Record input Pos:1,Len 16
17	2	Blank ('X' = blank)
19	2	Record input Pos:18,Len 2
21	1	Character '-'
23	2	Record input Pos:20,Len 2
25	1	Character '-'
26	2	Record input Pos:22,Len 2
28	2	2 blank

```
=====
USE  ../Files/FIL_OUTFIL_200.TXT          ORG LS RECORD F,3000
GIVE  ../Files/FIL_OUTFIL_200_007.TXT.SRT  ORG LS RECORD F,3000
SORT FIELDS=COPY
OUTREC=(01,16,2X,18,2,C'-',20,2,C'-',22,2,2X)
END
```

OUTREC FIELDS=(8,2, 20:5,10,3C'ABC',80:X)

Position Input	Len Input	Position output	Len output	Value
8	2	1	2	
5	10	20	10	Characters from pos 5, len10 from input
		30	9 (3 times x 3 char)	'ABCABCABC'
		80		Padding from 39 to 80

OUTREC FIELDS=(5C'LITERAL -',10X'414243',3X'525558',120,18)

Position Input	Len Input	Position output	Len output	Value
		1	45 (5 time x 9 char)	'LITERAL -LITERAL -LITERAL LITERAL-LITERAL-'
		46	30 (10 times 1 char hex)	'ABCABCABCABCABCABCABCABCABCABC'
		76	9 (3 times x 3 char hex)	'RUXRUXRUX'
80	18	85	18	Input record from 80 for 18 characters

OUTREC FIELDS=(1,40,60:Z,81:X)

Position Input	Len Input	Position output	Len output	Value
1	40	1	40	Input record from 1 for 40 characters
		41	20 (60 abs position - 40 current position)	20 characters with '00' binary
		61	20	21 characters with '20' space

## 19.6. OUTFIL

### OUTFIL INCLUDE

Example with more files for OUTFIL

Each file output with Include condition

The purpose is merge files and write four output.

```
FNAMES=FOUT201_1
FOUT201_1      Environment Variable
FOUT201_2      Environment Variable
FOUT201_3      Environment Variable
FOUT201_SAVE   Environment Variable
=====
USE  ../FIL_OUTFIL_001.TXT      ORG LS RECORD F,3000
GIVE ../FIL_OUTFIL_001.TXT.OUT ORG LS RECORD F,3000
MERGE  FIELDS=COPY
OUTFIL INCLUDE=(01,03,CH,EQ,C'201',AND,24,03,CH,LE,C'999'),FNAMES=FOUT201_1
OUTFIL INCLUDE=(01,03,CH,EQ,C'210',AND,24,04,CH,GT,C'0000',AND,24,04,CH,LE,C'9999'),FNAMES=FOUT201_2
OUTFIL INCLUDE=(01,03,CH,EQ,C'230',AND,36,04,CH,GT,C'0000',AND,36,04,CH,LE,C'9999'),FNAMES=FOUT201_3
OUTFIL SAVE,FNAMES=FOUT201_SAVE
=====
```

### OUTFIL OMIT

Format output record

OMIT Condition for input.

```
FOUTKEY_YES      Environment Variable
FOUTKEY_NO       Environment Variable
=====
USE  D:\GCSORTTEST\FilesT\FIL_OUTFIL_050.txt ORG LS RECORD F,3000
GIVE D:\GCSORTTEST\FilesT\FIL_OUTFIL_050.txt.OUT ORG LS RECORD F,3000
      SORT FIELDS=COPY
      OUTFIL OMIT=(156,15,CH,LT,141,15,CH,AND,005,10,CH,EQ,C'KEYMAX800E'),FNAMES=FOUTKEY_YES
      OUTFIL SAVE,FNAMES=FOUTKEY_NO
      END
=====
```

## 19.7. INREC/OUTREC CHANGE

### [ INREC CHANGE ]

```
INREC FIELDS=(15,6,25,3,CHANGE=(1,C'K12',X'41',C'M22',X'42',C'P32',X'43'),NOMATCH=(X'49'))
INREC
FIELDS=(1,15,16,2,CHANGE=(1,C'22',X'41',C'88',X'48',C'44',X'42',C'66',X'43'),NOMATCH=(X'49'),17,83)
```

### [ OUTREC CHANGE ]

```
OUTREC FIELDS=(15,6,25,3,CHANGE=(1,C'K12',X'41',C'M22',X'42',C'P32',X'43'),NOMATCH=(X'49'),26,4974)
```

### [ CHANGE - Position ]

```
OUTREC FIELDS=(1,1,CHANGE=(6,C'2',28,6),NOMATCH=(2,6),X,8,19,35,15,51,59)
```

## 19.8. DATE

```
[ DATE - Currente Date : DATE4 ]
INCLUDE COND=(1,13,CH,GT,DATE4)
USE ../files/inp5000.txt ORG LS RECORD F,5000
GIVE ../files/inp5000.txt.srt ORG LS RECORD F,5000
SORT FIELDS=(35,5,ZD,A)
```

```
[ DATE + / - day - month ]
COND=(1,13,CH,GT,DATE1+5)
COND=(1,13,CH,GT,DATE1-5)
```

```
COND=(1,13,CH,GT,DATE2+3)
COND=(1,13,CH,GT,DATE2-8)
```

```
COND=(1,13,CH,GT,DATE3+150)
COND=(1,13,CH,GT,DATE3-15)
```

```
[ DATE4 ]
OMIT COND=(1,13,CH,GT,DATE4)
```

## 19.9. RECORD CONTROL STATEMENT

```
[ RECORD CONTROL STATEMENT ]

SORT FIELDS=(8,5,CH,A) USE ../files/sqbig01.dat ORG SQ GIVE ../files/sqbig01_gcs.srt ORG SQ
RECORD TYPE=F, LENGTH=500

RECORD TYPE=F, LENGTH=(500)

RECORD TYPE=F, LENGTH=(500, ,500)

RECORD TYPE=F LENGTH=(, ,500)

RECORD TYPE=F,LENGTH=(, ,500)
```

## 19.10. DATE - Option Y2PAST

```
[ DATE - Y2PAST ]
SORT FIELDS=(10,8,Y2T,A)
USE FDate.dat RECORD F,85 ORG SQ
GIVE FDate.dat.Y2T8.srt RECORD F,85 ORG SQ
OPTION Y2PAST=80
```