Assort

A high performance Sort/Merge/Select Utility

Bill Ahlbrandt Software
Little Rock, AR
(501) 280-0462

CompuServe 73121,652

Table Of Contents

Assort is a high performance sort/merge/copy utility designed for the manipulation of a wide variety of file formats.

Assort is optimized to minimize the consumption of system resources such as disk space, CPU, memory and I/O.

Assort provides significant performance improvements over competitive products at a value based price.

Assort was modeled after sort utilities that are available on IBM 370/390 Architecture machines. These utilities have been in use for years with various levels of integration with other mainframe based applications.

Assort allows the user to perform Sort/Merge/Copy operations in non-mainframe environments using a familiar control card format.

Assort can be initiated through the command line interface, or called as a subprogram. When called as a subprogram, the calling program can perform input/output, or Assort can do all input/output processing. When Assort is allowed to perform all input/output, processing time is generally lower due to the optimized I/O and memory management facilities of this product.

There are no run-time royalties associated with Assort. This allows the licensee of this product to integrate Assort into his product and distribute it without paying any run time royalties. The only restriction is that the product that is distributed is not a generic Sort/Merge/Select Utility.

1Introduction

1Benefits

[illegible] sequencing.

[illegible] environments by using familiar control syntax.

Merging Combine any number of sorted input files into one sorted output file

Copying duplicate the input set of records without changing their sequence

- Provide mainframe class reliability and a robust feature set for users upsizing applications.
- Provide a higher level of performance than competitive products.
- Dramatically decrease database load and index time by sorting tables prior loading.
- Efficiently process extremely large files.
- Select records
- Reformat records through parameter based data manipulation

2Basic Functions of Assort

Basic Functions of Assort

1Sorting

2Merging

3Copying

where:Assort can be invoked from the command line with the following syntax:

controlFileName	is the name of the text file that contains the control information
inputFileSpec	the specification for the input file(s) $\text{collection}_1[,\text{coll}$ where: $\text{collection}_i = \text{file}$
outputFileSpec	the specification for the output file(s) $\text{file}_1[,\text{file}_2\ldots,\text{fil}$

Multiple input files can be specified to concatenate multiple physical files.

All files must have the same record format. When specifying the record length for non-fixed length records, use the largest record length of all concatenated files.

The plus sign (+) is used to indicated that files are to be concatenated.

The comma (,) is used to indicated that the next collection of files corresponds to another input file set.

The first file in a file specification corresponds to SORTIN or SORTIN01, the second file corresponds to SORTIN02.

When merging multiple input files use a comma between the file name(s) to specify another input stream of data.

The SORTIN dataset is all that is used in this version of Assort.

The SORTOUT is the only output that is used in this version of Assort.

2Command Line Syntax

ASSORT *controlFileName inputFileSpec outputFileSpec*

If you had four files of sales information, one for each quarter of the year, and you wanted to merge them by date into one single output file, the following example shows how you would specify the customer account number, the following syntax would be used:

2Sorting Multiple Input Files

```
 ASSORT BYACCT.CTL Q1SLS.DAT+Q2SLS.DAT+Q3SLS.DAT+Q4SLS.DAT BYACCT.DAT
where:

BYACCT.CTL          contains the sort control specification

Q1SLS.DAT           contains the sales data for the first quarter
Q2SLS.DAT           contains the sales data for the second quarter
Q3SLS.DAT           contains the sales data for the third quarter
Q4SLS.DAT           contains the sales data for the fourth quarter

BYACCT.DAT          will contain the sorted data
```

Concatenating Input files

3Merging Multiple Input Files

```
 ASSORT BYDATE.CTL Q1SLS.DAT,Q2SLS.DAT,Q3SLS.DAT,Q4SLS.DAT BYDATE.DAT
where:
BYDATE.CTL          contains the merge control specification

Q1SLS.DAT           contains the sales data for the first quarter
Q2SLS.DAT           contains the sales data for the second quarter
Q3SLS.DAT           contains the sales data for the third quarter
Q4SLS.DAT           contains the sales data for the fourth quarter

BYDATE.DAT          will contain the merged data
```

Example of merging input files

Assort control statements are used to specify the desired output from input data set.

The supported control statements are as follows:

Statement	Function
ALTSEQ	Define an alternate collating sequence
END	Declare the end of the control statements
INCLUDE	Specify records to be included in processing
INREC	Reformat input records
MERGE	Specify merge fields and options
OMIT	Specify records to be omitted from processing
OPTION	Specify runtime options
OUTREC	Reformat output records
RECORD	Define input record characteristics
SORT	Specify sort fields
SUM	Specify fields to accumulate or eliminated duplicate records

3Control Statements

Assort Control Statements

Control Statement Name	Parameters	Default
ALTSEQ	CODE=(ccpp$_1$, . . . ,ccpp$_{256}$)	Unsigned binary collating Sequence
END	none	
INCLUDE	COND=(condition[,AND/,OR,condition])	Include all records
INREC	FIELDS=(field$_1$[,field$_2$] . . .)	Entire input record
MERGE	FIELDS=(p_1,l_1,f_1,o_1, . . ., p_n,l_n,f_n,o_n) FIELDS=(p_1,l_1,o_1, . . ., p_n,l_n,o_n),FORMAT=f FIELDS=COPY EQUALS/NOEQUALS FILES=n SKIPREC=n STOPAFT=n	 EQUALS Merge or Copy all Records Merge or Copy all Records
OMIT	COND=(condition[,AND/,OR,condition])	Omit no records
OPTION	[CORE=n]	CORE=4000000
OUTREC	FIELDS=(field$_1$[,field$_2$] . . .)	Entire input record
RECORD	LENGTH=(l_1, . . .,l_7), TYPE = [F V]	
SORT	FIELDS=(p_1,l_1,f_1,o_1, . . ., p_n,l_n,f_n,o_n) FIELDS=(p_1,l_1,o_1, . . ., p_n,l_n,o_n),FORMAT=f FIELDS=COPY DYNALLOC=(d) EQUALS/NOEQUALS FILSZ=n SIZE=n SKIPREC=n STOPAFT=n	 TMP environment variable EQUALS Sort or Copy all Records Sort or Copy all Records
SUM	FIELDS=(p_1,l_1,f_1,. . .,p_n,l_n,f_n) FIELDS=(p_1,l_1,. . .,p_n,l_n),FORMAT=f FIELDS=NONE FIELDS=(NONE)	No summarizing of records

If the process of summing two records causes a sum field to overflow, multiple records will be written out.

If SUM FIELDS=NONE is coded, all but the first record with equal sort fields will be deleted.

1Control Statement Summary Chart

Control Statement Summary Chart

Control statements my be in any order with the exception of the END statement. The END statement must be the last statement in the control file.

Each control statement may be only coded once.

Control statements must begin after column one.

Control statements must end before to column 72.

A control statement and it's parameters must be separated by at least one blank.

Parameters must be separated by a comma.

Columns 73-80 are ignored. These columns can be used for sequence numbers.

4Control Statement Formats

1Example of Control Statements

```
RECORD TYPE=F,LENGTH=80
SORT FIELDS=(1,80,BI,A)
END
```

Comments can be used to document or improve readability of the sort control information.

A comment card contains a '*' in column one. Comment cards are ignored.

Any number of comment cards can be coded.

Comments can also be coded after the end of a control statement by inserting one or more blanks between the control information and the comment.

If column 72 contains a non-blank character, the comment is continued to the next card.

2Comments

1Example of Comments

```
*
* This is a comment card.  It is ignored
* This sort will sort a fixed length file with 80 byte records
*
 RECORD TYPE=F,LENGTH=80
 SORT FIELDS=(1,80,BI,A),EQUALS      This is also a comment
 END
```

Labels can be used on control statements to improve readability. Labels are for your reference only. Assort ignores any labels that are coded.

Labels must begin in column one.

Labels can be any length provided other control statement rules are not violated.

At least one blank must separate a label from the control statement.

3Labels

1Example of Labels

```
           RECORD TYPE=F,LENGTH=80
DoTheSort  SORT FIELDS=(1,80,BI,A)
           END
```

column 72. The statement should be continued on the next line after column one.

A control statement may also be continued by placing a non-blank character in column 72, and continuing the statement on the next line after column one. The continuation will begin at the first non-blank character.

If the next line contains a label, place at least one blank after the label and continue the control statement.

If the continuation is to occur in a quoted string, place a non-blank character in column 72, and begin the continuation on the next line in column 16.

4Continuation of Control Statements

1Example of Continuations

```
 RECORD TYPE=F,
 LENGTH=80
 INREC FIELDS=(1,20,5:
     71,10)
 SORT FIELDS=(1,80,BI,A)
 END
 RECORD TYPE=F,LENGTH=8000
 INREC FIELDS=(1,8,400:9,2,200:79,2,60:7632,200,315:30,5,80,10,C'abcdefX
               ghijklmnop')
* This is a very long comment                                          X
                               and this is the rest of the comment.
 END
 RECORD TYPE=F,LENGTH=80
 INREC FIELDS=(1,8,
THERESTOFTHESTATEMENT C'ABC')
 END
```

The ALTSEQ statement is used to specify a user defined collating sequence. This can be used to sort files that contain information encoded in alternate character sets.

The syntax of the ALTSEQ statement is as follows:

cc is the hexadecimal representation of the character in the input file.
pp is the hexadecimal position the character should be placed in.

The cc and pp values should be in the range x'00' - x'FF'.

To sort ASCII data such that character zeros (x’30’) to be equal to blanks (x’20’) code the following:

5ALTSEQ Statement

ALTSEQ CODE=(ccpp$_1$, . . . ccpp$_{256}$)

```
ALTSEQ CODE=(3020)
ALTSEQ CODE=(3020,3120,3220,3320,3420,3520,3620,3720,3820,3920)
```

The END statement is used to signify the end of the control statements. The END statement is required, and must be the last control statement in the set of control cards.

There are no parameters for the END statement, but it can contain comments.

6END Statement

```
END
```

The INCLUDE/OMIT Statement specifies the conditions for records that should be included in or omitted from the sort/merge/copy process.

The INCLUDE Statement is used to specify the conditions for records that should be included in further processing.

Any level of nested parenthesis may be used.

Example of the FORMAT= parameter on the INCLUDE Statement

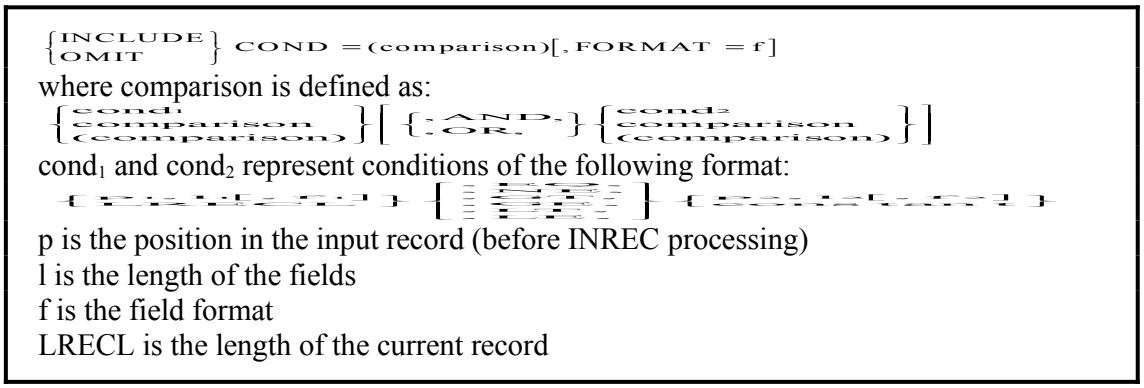
The OMIT Statement is used to specify the conditions for records that should be omitted from further processing.

One INCLUDE statement and one OMIT statement may be used. The combination of the statements is used to determine if the record is included in further processing.

These statements are useful for eliminating unnecessary records based on their length. In text processing applications, it may be useful to delete records of zero length.

There is no limit to the number of Include/Omit conditions that can be coded.

7INCLUDE/OMIT Statement



INCLUDE/OMIT syntax

1Examples of Include/Omit Statements:

```
 INCLUDE COND=(1,5,CH,EQ,C'72202',OR,1,3,CH,EQ,C'652')
 INCLUDE COND=(1,5,CH,EQ,C'72202',AND,(10,2,PD,GT,30,OR,10,2,PD,LT,4))
INCLUDE COND=(1,5,EQ,C'72202',AND,1,3,EQ,C'652'),FORMAT=CH
```

When specifying field to field comparisons on INCLUDE/OMIT cards, use the following table to determine if fields of different types can be compared.

FORMAT CODE	I	FI	BI	CH	ZD	PD
I	X	X	X			
FI	X	X				
BI	X		X	X		
CH			X	X		
ZD					X	X
PD					X	X

8Field to Field Comparisons

Table of valid field to field comparisons

When using the INCLUDE/OMIT statement, you can code constants in the following formats. The valid constants are listed in the table below:

Constant Type	Syntax	Examples	Notes
Character	C'...'	C'ALPHA' C'I'll be'	Use two single quotes to insert a quote in the constant.
Hexadecimal	X'....'	X'0D0F' X'7C'	The number of digits coded must be even.
Binary	B'....'	B'00011100' B'1.....'	Must code eight digits Use '.' to indicate don't care conditions
Numeric	numeric	89 +421323 -370	Plus sign is optional

For character fields, the padding takes place on the right. The pad character is a blank.

For Binary fields, the padding takes place on the right. The field is padded with binary zeros.

For numeric fields, zeros of the appropriate data type are used to pad the number on the left.

If the fields are signed data types such as FI or I, the sign bits are propagated when necessary if padding takes place.

9Specifying Constant Data

Specifying Constants

When using the INCLUDE/OMIT Statement, you can perform selections based on the length of the record. This is useful when you want to eliminate records that have zero length.

For Example, to omit records that have a length of zero code:

10Selection Based on Record Length

```
OMIT COND=(LRECL,EQ,0)
INCLUDE COND=(LRECL,GT,0,AND,LRECL,LE,80)
```

The INREC statement is used to format the record before it is sorted, merged or copied. This is useful to eliminate unnecessary fields to improve the performance of the sort.

The OUTREC statement is used to reformat the record after it has been sorted. This is useful when expanding the record or reformatting fields.

In order to optimize the performance of the sort, the variable length portion of the record ... fields should be eliminated from the record with the INREC statement ... with the OUTREC card.

The following table shows the syntax of INREC/OUTREC Constants.

- **c: Optional.** The column number where the field should be placed in the output record. If c is not specified, the column number will automatically be calculated to be one more than the last position of the previous field.
- To include part of the variable length portion of a record, specify the starting position without a length. The record will be constructed using the specified position to the end of the record. This field must be the last field specified on the INREC/OUTREC card.

p	The position in the record where the field begins For INREC Cards, this is the position in the input record as it is read from the input file. For OUTREC Cards, this is the position in the record after INREC and SUM fields processing.
l	The length of the field in bytes

[n]	Replicator. Specify a number to replicate the following constant. The default replicator is 1. n can be in the range from 1 to 2147483647.
B'bbbbbbbb'	Insert a binary constant into the record. Don't care conditions are not permitted. Valid binary digits are 0 and 1.
X	Use the X constant to create an output field that contains blanks.
X'hhhh...hh'	Use the X'hhhh...hh' constant to create an output field that contains a hexadecimal constant. n can be in the range from 1 to 2147483647.
C'literal'	Use the C'literal' constant to insert a literal string into the output record. n can be in the range from 1 to 2147483647.
Z	Use the Z literal to insert binary zeros into the output record. n can be in the range from 1 to 2147483647.
CRLF	Insert a Carriage Return/Line Feed (x'0D0A') in the output record. This can be used to convert a file containing Fixed or Variable length records to a Text file.

11INREC/OUTREC Statement

```
{INREC  }
{OUTREC } FIELDS =(field1[,field2]...)
where fields can be coded as follows:
[c:]p1,l1[,subparameters]
or
[n]B'bbbbbbbb'
[n]X
[n]X'hhhh...hh'
[n]C'literal'
[n]Z
```

INREC/OUTREC Position and Length parameters

INREC/OUTREC Constants

1Rules for specifying INREC/OUTREC fields

2

Use the Fields Subparameters for the following tasks:

- Force Halfword (two bytes), Fullword (four bytes), or Doubleword (eight bytes) alignment
- Edit a numeric field
- Convert a numeric field
- Convert a field to a printable hexadecimal representation

The following table shows the subparameters of the Fields parameter

a	Alignment specification. This can be either H, F, or D for Halfword, Fullword or Doubleword alignment. Halfword alignment rounds the output position up to the nearest two byte boundary Fullword alignment rounds the output position up to the nearest four byte boundary Doubleword alignment rounds the output position up to the nearest eight byte boundary An appropriate number of binary zeros will be inserted in the record to obtain the specified alignment. Note: Column one in the output record is double word aligned.
f	The format subparameter is used to define the format of the input field. The format should be BI, FI, PD, ZD or I. The length must be specified that is consistent with the eligible lengths for each data type. BI 1 to 4 bytes (kludge: are these correct?) FI 1, 2 or 4 bytes PD 1 to 8 bytes ZD 1 to 15 bytes I 1 to 4 bytes
Mm	One of the default edit masks M0-M9. See the section on default edit masks for more information
EDIT=(pattern)	A user defined edit pattern. See the section on defining edit patterns for more information.
SIGNS=(s1,s2,s3,s4)	This subparameter defines how the signs will be placed in the output record. See the section on specifying signs for more information.
LENGTH=n	The LENGTH subparameter is used to override the default length for an edited field.
HEX	The HEX subparameter is used to convert a single byte in the input record to a two byte hexadecimal representation of the field in the output record.

3Fields Subparameters

INREC/OUTREC Fields Subparameters

INREC/OUTREC Fields Subparameter

Editing is the process of converting a machine readable data element to a format that is pleasing to the eye. As such, it has the following default edit masks available for editing numeric fields:

Mask	*Edit Pattern*
M0	IIII...ITS
M1	TTTTT...TTTS
M2	I,III,...,IIT.TTS
M3	I,III,...IIT.TTCR
M4	SI,III,...,IIT.TT
M5	SI,III,...,IIT.TTS
M6	III-TTT-TTTT
M7	TTT-TTT-TTTT
M8	IT:TT:TT
M9	IT/TT/TT

4Editing

1Default Edit Masks

Default Edit Masks

[illegible]

[illegible]

[illegible] the following:

- Significant digit selectors
- Insignificant digit selectors
- Leading sign characters
- Trailing sign characters
- Any other characters to appear in the output record

[illegible] padded with blanks on the left so that the length of the pattern equals the 'n' value.

The maximum value which can be specified for 'n' is 32.

If a comma, blank, open or closing parenthesis are used as sign characters, the must be enclosed in single quotes.

If a single quote is to be used as a sign character, it must be entered as four single quotes as follows:

SIGNS=(s_1,s_2,s_3,s_4)

s_1	Leading positive sign indicator
s_2	Leading negative sign indicator
s_3	Trailing positive sign indicator
s_4	Trailing negative sign indicator

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 characters 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.

There is no predetermined limit to the length of an edit mask. Edit masks that exceed 40 characters are generally unreasonable due to the limited number of digits that can be created from an input field.

The significant and insignificant characters used in the edit pattern are determined by the EDxy keyword. The default is EDIT. This makes 'I' the insignificant character and 'T' the significant character. If you wish to use the characters 'I' or 'T' in an output field, modify the EDxy keyword and the edit mask accordingly. This allows any character to be placed in the output field.

2Defining Edit Patterns

EDxy=(pattern)
where: x = insignificant digit selector
y = significant digit selector

3The LENGTH=n Subparameter

4The Signs Subparameter

SIGNS Subparameter description

```
SIGNS=(,'''',,'''')
```

The MERGE statement is used to combine the fields and records of two files into a third file. The syntax of the MERGE statement is as follows:

12MERGE Statement

1FILES

The FILES parameter is used to specify number of files to be merged. This parameter is accepted, but not currently used. The number of files to be merged is determined by the number of files specified on the command line.

The OPTION statement defines the general processing options. The syntax is as follows:

13OPTION Statement

```
OPTION [CORE=n]
```

The CORE Option allows you to specify the number of bytes to allocate for internal sorting tables. The default value for *n* is 4000. This will cause approximately four megabytes of virtual storage to be allocated to the internal sorting process.

The CORE Option can be adjusted to optimize the sorting process depending on the physical memory size of your machine.

The CORE parameter only applies to Sorts, and is ignored for Merges and Copies.

TYPE	Description
F	Fixed length records. The length is obtained from the L value.
V	IBM style variable length records. The beginning of each record contains a four byte header. This header includes a two byte IBM 370 format Unsigned Integer and two bytes of reserved information. Following the four bytes of header information is the actual record data. The record length in the header includes the length of the four byte header.
T	Text type records. Records are delimited by: • a Carriage-Return/Line-Feed character combination • a newline character. An MS-DOS style End of file character is permitted, but not required.
I	Variable length records. The record length is obtained from a two byte Intel format unsigned integer that is appended to the front of each record. The two byte field contains the length of the record. This length does not include the two byte appendage.

14RECORD Statement

RECORD [illegible]

Valid Record Formats

The FIELDS parameter of the SORT statement defines the sort keys and order of the output file.

Variable	Meaning
p	Position in the input record in bytes after INREC processing.
l	Length of the Field in bytes. If this is the last sort field, you can omit the length and the remainder of the record will be used in field comparisons. This is useful when the length of the records is not known at run time such as in Variable and Text records.
o	A for Ascending D for Descending E as modified by an E61 user exit
f	the format of the field as documented in the valid formats chart

15SORT Statement

SORT FIELDS=

$$\left\{\begin{array}{l}(p_1,l_1,f_1,o_1,p_2,l_2,f_2,o_2,\ldots,p_n,l_n,f_n,o_n)\\ p_1,l_1,o_1,p_2,l_2,o_2,\ldots,p_n,l_n,o_n),FORMAT=f\\ COPY\end{array}\right\}$$

$$\left[,DYNALLOC=\left\{\begin{array}{l}pathName\\(pathName,n)\end{array}\right\}\right]$$

$$\left[\begin{array}{l},EQUALS\\,\underline{NOEQUALS}\end{array}\right]$$

$$[,SKIPREC=n][,STOPAFT=n]$$

1FIELDS

Sort Fields Description

The following table contains the valid format codes, the data representation, and the valid lengths for each code..

Format Code	Data Format	Valid Lengths
AC	The EBCDIC data is converted to ASCII prior to comparisons.	1 to record length
AQ	Alternate Collating sequence as defined in the ALTSEQ statement	1 to record length
ASL	ASCII Signed Leading An ASCII '+' or '-' precedes the numeric field. One digit per byte.	2 to record length
AST	ASCII Signed Trailing An ASCII '+' or '-' trails the numeric field. One digit per byte	2 to record length
AU	ASCII characters are converted to upper case prior to sorting	1 to record length
BI	IBM 370 format Binary Unsigned Fields of unequal length are zero padded and right justified.	1 to record length
CH	Same as BI Fields of unequal length are blank padded and left justified.	1 to record length
CLO or OL	Leading Overpunch sign. Hexadecimal D,B, or 2 in the first four bits of the field indicates a negative number. Any other value indicates a positive number.	1 to record length
CSL or LS	an EBCDIC '+' or '-' precedes the numeric field. One digit per byte.	2 to record length
CST or TS	an EBCDIC '+' or '-' trails the numeric field. One digit per byte.	2 to record length
FI	IBM 370 format Signed integer	1 to record length
I	Intel format Signed Integer. Byte order reversed	1 to 4 bytes
PD	Signed Packed decimal. The sign is determined by the last four bits of the field. If the last four bits are hexadecimal B, D, or 2, the field is negative, otherwise the field is positive.	1 to record length
ZD or CTO	Zoned Decimal. Trail overpunch sign in the first four bits of the last byte is used for the sign. Hexadecimal B, D, or 2 indicate a negative number. All other values result in a positive number.	1 to record length

2FORMAT Codes

SORT/MERGE Format Codes

3DYNALLOC

[illegible]

The default path is the directory that is pointed to by the environment variable TMP. This directory will be [illegible]

If the TMP environment variable is not set, or points to an invalid path, the DYNALLOC value is used. If this path is invalid, the current working directory is used..

The DYNALLOC parameter can be used to indicate where work files should be created. For large sorts, the work files could be directed to a different disk drive.

4EQUALS/NOEQUALS

[illegible] NOEQUALS, which means that the order of equal keyed records is undefined.

[illegible] This was not the case in previous versions, and may be different in future releases. . Therefore for compatibility purposes, it is recommended that the EQUALS parameter be coded if the ordering of equal keyed records is significant.

5SKIPREC

[illegible] in the input file.

[illegible]

6STOPAFT

[illegible]

For example:

To process records 100 through 110, code the following:

SKIPREC=99,STOPAFT=10

This will cause the first 99 records to be skipped, and 10 records to be processed.

The SUM statement is combination with the SKIPREC parameter. The valid lengths for each format are listed in the following table.

Format Code	Allowable Length
BI	1 to Record Length
FI	2 to Record Length (Must be an even number of bytes)
I	1, 2, or 4 bytes
PD	1 to Record Length
ZD	1 to Record Length

16SUM Statement

SUM FIELDS= [... — { ... }]

SUM FIELDS Subparameter descriptions

SUM FIELDS Format Codes

Assort can be invoked from the command line or from an application program.

1Programmer's Reference

There are two interface ways to Assort. One is straight initiation of the program, the is the record level interface. The record level interface allows you to pass records to the sort, sort them, and read them back.

The ASSORT.EXE uses the straight initiation of the program.

The Assort command replacement SORT.EXE uses the record level interface.

The main disadvantage to using the record level interface it that the optimized input/output facilities are not used. The driver program performs all input/output.

The advantage to using the record level interface is that file formats that are not supported by Assort can be processed.

Another advantage to the record level interface is that records do not have to reside on disk at all. They can be C++ objects that reside in memory or they can be records in an external database.

When sorting objects that reside in memory, the records could be four byte addresses, and the E61 user exit could be called to compare pairs of pointers to objects.

The following functions are used when calling Assort:

Function Name	Straight Initiation	Record Level Interface	Description/Usage
makeDeck	Always	Always	Creates a deck object
registerCallbackFunction	Sometimes	Sometimes	Registers a callback function
assortpar	Always	Always	Parses out control information
assortsor	Always		Executes based on the parsed information
sProc		Always	Sort Records
sPush		Always	Pull Sorted Records back
sPush		Always	Push Records to the Sort

17Function Definitions

The following functions are used when calling Assort.

deckT *makeDeck();

This function is used to construct a deckT object. If successful, returns a pointer to a deckT object. If unsuccessful, returns NULL.

int registerCallbackFunction(

```
  deckT *deck,                          // pointer to deckT object
  unsigned int routineNumber,           // callback Routine Number
  pCallbackFunction pcallback,          // address of Callback Routine
  void *userHandle                      // user specified handle
);
```

Pass the address of the deckT object created by the makeDeck() function. The value for routineNumber should be EMESSAGES, or E61Callback. These values are defined in the file *assort.h*.

The value for userHandle can be any four byte value including pointers. Whatever is passed in this parameter will be passed to your callback function.

int assortpar(

```
  deckT *deck,                                  // pointer to deckT object
  const char *controlInformation                // Assort control information
);
```

This function parses out control information. After calling this function you can use the parsed information to execute the Sort/Merge or Copy.

Pass the address of the deckT object created by the makeDeck() function. The controlInformation field contains the Assort control information. Control cards should be separated by at least one space. No comments or labels are permitted in this parameter.

```
int assortsor(
```

```
  deckT *deck,                          // pointer to deckT object
  const char *inputFileSpec,            // input file specification
  const char *outputFileSpec            // output file specification
);
```

Initiates the sorting process.

The inputFileSpec contains the specification for the input file name(s). This is the same format that the Assort command accepts. Concatenate inputs with a '+' sign and separate multiple inputs with commas. Multiple inputs are used for merges.

The outputFileSpec contains a single file name.

int killDeck(

```
  deckT *deck,                                  // pointer to deckT object
);
```

Initiates the sorting process.

The inputFileSpec contains the specification for the input file name(s). This is the same format that the Assort command accepts. Concatenate inputs with a '+' sign and separate multiple inputs with commas. Multiple inputs are used for merges.

The outputFileSpec contains a single file name.

```
int makePcb(
```

```
  pcbT *pcb,                    // pointer to a pcbT object
);const deckT *deck,            // pointer to a deckT object
  filesT *filesParm             // pointer to a filesT object
);
```

Destroys a pcbT object. This function is used when using the record level interface.

If successful, returns zero. Returns non-zero value if an error occurs.

```
int killPcb(
  pcbT *pcb,                    // pointer to a pcbT object
);const deckT *deck,            // pointer to a deckT object
  filesT *filesParm             // pointer to a filesT object
);
```

Creates a pcbT object. This function is used when using the record level interface.

If successful, returns zero. Returns non-zero value if an error occurs.

deck should be a pointer to a deckT object that was created with the assortpar function.

filesParm should be NULL.

```
int sProc(
```

int sPush(

Pushes a record to the sort. Call this function after the makePcb function. After all objects (records) have been pushed ...

If successful, returns zero. Returns non-zero value if an error occurs.

If successful, returns zero. Returns non-zero value if an error occurs. EOF indicates there are no more objects to process.

recordBuffer contains a pointer to a recordBuffer structure for record 2

Record 1 is greater than Record 2 — 1

Record 1 is less than Record 2 — -1

int someFunction() { ... registerCallbackFunction(deck, EMESSAGES, msgRoutine, NULL);

"RECORD TYPE=F LENGTH=1000 SORT FIELDS=(1,CH,A) END";

assort(deck, "c:\\test.in", "c:\\test.out");

registerCallbackFunction(deck, E61, e61Routine, NULL);

1Invoking Assort Directly

2Using The Record Level Interface of Assort

```
#include <stdio.h>
#include <string.h>
#include "assort.h"

void someFunction(void) {
	deck_t deck=NULL;
	pcb_t pcb=NULL;
	char *aRecord;
	unsigned long aLength;
	char record1[] = "cat";
	char record2[] = "goat";
	char record3[] = "zebra";
	char record4[] = "antelope";
	if (deck=makeDeck()) {
		if (0==assortpar(deck, controlInfo)) {
			if (!makePcb(&pcb, deck, NULL)) {
				sPush(pcb, record1, strlen(record1)+1);
				sPush(pcb, record2, strlen(record2)+1);
				sPush(pcb, record3, strlen(record3)+1);
				sPush(pcb, record4, strlen(record4)+1);
				sProc(pcb);
				printf("The sorted records:\n");
				while(EOF != sPull(pcb, &aRecord, &aLength)) {
					printf("%s\n", aRecord);
				} /* endwhile */
				killPcb(pcb);
			} else {
				printf("makePcb Failed\n");
			} /* endif */
		} else {
			printf("Parse Failed\n");
		} /* endif */
		killDeck(deck);
	} else {
		printf("makeDeck() failed!\n");
	} /* endif */
	return;
}

int main() {
	someFunction();
	return 0;
}
```

int sPull(

3Callback Functions

1Message callback function

1Example of using the message callback function

2E61 callback function

1Example of using the E61 callback function

5Performance and Tuning

By eliminating unnecessary fields from the input file, the total file size is reduced. This results in an overall improvement in performance.

By eliminating unnecessary records from the input file, the total file size is reduced. This results in an overall improvement in performance.

There are many ways to "tune" the sorting process. Depending on your objectives, you may wish to minimize elapsed time or work disk space requirements.

This chapter focuses on operational issues that arise in the Sort/Merge process and techniques that can be used to exploit this product efficiently.

An example of this would be if you had an input file that contained 4000 byte records. In each record there was a nine byte account number in positions one through nine. If you wanted to know the number of distinct account numbers in the file, you could sort the file on positions one through nine with the SUM FIELDS=NONE option. This would give you an output file with a single 4000 byte record for each distinct state code in the input file.

If the file had 20 million records, the maximum sortwork requirement would be 80 Gigabytes.

If you used the INREC card and only included the nine byte account number field, the maximum sortwork requirement would be 180 Megabytes

This would be about a 95% reduction in disk processing time. Reductions in disk processing time are directly proportional to total processing time, so the overall throughput would be significantly better.

1Minimizing Elapsed Time

1Eliminating Unnecessary Fields

2Eliminating Unnecessary Records

There are two major phases to the sort process:

Note that the previous examples can be placed on the same physical device without any performance impact because the input and output records are never needed at the same time.

The environment variable TMP can be set to a work drive. This drive should be a different device than where permanent data files are generally stored.

The DYNALLOC parameter of the SORT card can also be used to direct sortwork files to a directory on another device.

3Phase II: Writing Output Records

2Optimizing Disk Performance

1Optimizing Sorts

The phases of a Sort

2Optimizing Merges and Copies

The formula for copy processing performs I/O on the input and output files at the same time. Therefore it is best to have all files on different physical devices. In some cases this may not be feasible, due to a large number of files and a small number of physical devices.

Record Type	Description	Maximum Sortwork Required
F	fixed length records	$n*l*(1.01)$
V or T	Variable length or Text records	$n*(l+2)*(1.01)$
I	Variable length records (IBM Format)	$n*(l+4)*(1.01)$

where:

n is the number of records that survive INCLUDE/OMIT processing
l is the average record length of the records after INREC processing

3Sortwork calculations

Assort messages are all of the format ICY*nnnnt* or ICC*nnnnt.* ICC messages come from the command driver (ASSORT.EXE). ICY messages originate in the sorting engine (ASSORT.DLL).

2Messages

Where:

nnn is the message number
t is the message type
I for Informational Messages
E for Errors
W for warnings
B for tuning or performance statistics

ICC0004B CPU Time: 1 seconds, Elapsed Time: 1 seconds

ICC0012E Assort Failed

Message	Description	Action required or recovery procedure.
ICC0004B	CPU Time: nnnnn seconds, Elapsed Time: nnnnn seconds	none
ICC0011I	End Assort	Assort completed successfully
ICC0012E	Assort Failed	Correct error
ICY0001I	Copyright (c) 1993, 1994, 1995 Bill Ahlbrandt Software	none
ICY0002I	xxxxxxxx Callback routine registerd	A callback routine (User Exit) named xxxxxxxx was registered
ICY0002E	xxxxxxxx Callback routine unregisterd	A callback routine (User Exit) named xxxxxxxx was unregistered
ICY0004I	Unsupported Callback Routine xxxx	An attempt was made to register an unsupported callback routine
ICY0207W	Unregisterd Copy	Contact Bill Ahlbrandt for a registration information
ICY0502E	Unable to open input file xxxxxxxx	Correct the input file name specification
ICY0530I	File xxxxxxxx, Records In xxxxxxxx	Number of records read from the input file
ICY0530I	File xxxxxxxx, Records Out xxxxxxxx	Number of records written to the output file

Assort has been written entirely in the C++ programming language. Operating system dependent information has been minimized without sacrificing performance. Currently OS/2 2.x, OS/2 Warp, Windows 95 and Windows/NT are supported platforms. Other 32-bit platforms are scheduled to be supported.

3Operating System Specifics

There are currently no platform specific differences or limitations in this product.

4Installation

There are four directories on your installation disk. These are WIN32, OS2, INCLUDE and IVP. the WIN32 and OS2 directories contains the following files:

Filename	Description	Installation Directory	Notes
ASSORT.EXE	The command line interface for Assort.	Any directory that is in your path	Required
ASSORT.DLL	The Assort sorting engine.	Any directory that is in your path	Required
		On OS/2, this must be placed in a directory that is included in your LIBPATH variable that is specified in your CONFIG.SYS	
ASSORT.LIB	Import library for linking with your programs.	a directory your linker searches	Optional
SORT.EXE	A Sort command replacement for Win32 or OS/2 2.x	a directory in your path that is searched before the DOS or OS/2 system directory that contains the file SORT.EXE	Optional

The INCLUDE directory contains the file ASSORT.H. This file should be included into your C and C++ programs when calling Assort.

The IVP directory contains sample data files and control cards to ensure that the product has been installed properly.