Assort User Documentation

Fundamentals

Overview

In the data processing world, sorting has always been an necessary evil. Most tasks will sooner or later involve the sorting of records.

External sorting can make procedeural programming tasks easier and provide higer quality results.

This manual is designed to educate the user of this utility to attain higher productivity and produce high quality results from data processing issues.

This product performs many more tasks than simply sorting records. Performance and flexibility were fundamental requirements in the original design process. The syntax of the control information is compatible with third party sorting utilities that have been available for a number of years in the IBM MVS and VSE environments.

There has been a woeful lacking of robust utilities available for microcomputer platforms that have been commonplace in the mainframe world for years. Without these fundamental utilities, the microcomputer platforms will be limited to processing relatively small volumes of data at an agonizingly slow pace.

What has been lacking in the PC marketplace is a robust, high performance sort/merge/select utility that runs in the OS/2, Windows 95 and Windows/NT environments.

This product *efficiently* processes fixed and variable length records.

This product has been designed to efficiently process files in excess of one gigabyte. There are no arbitrary limits on the number of sort fields, file size, or number of files that can be processed.

There is currently limit on the record length you can process. The maximum record length is 4,294,967,295 bytes (32 bits). If you wish to process records that approach this size, ensure that your machine is configured with adequate virtual storage

File Formats

In order to effectively use this utility, a discussion about file formats is neccessary. A file is considered a set of records. Records contain fields, and fields contain the individual bytes of information.

Operating systems such as IBM's MVS®, VM®, VSE® and Hewlett Packard's MPE/ix® maintain information about the records in each file. This information includes the length (number of bytes) of each record, the format of the records, and the number of records in the file. In these environment, the operating system provides routines to process files a record at a time.

Typical file formats in these environments include Fixed, Variable and Undefined.

> Fixed format files contain records that are all the same lengths.
>
> Variable format files contain record of varying length. Each record contains header information that defines the length of the record.
>
> Undefined format files have no implied record length. They are simply a sequence of bytes.

Unix, OS/2 and DOS/Windows do not support the concept of a file format. All files are viewed as a sequence of bytes. These files are basically the same as Undefined format files in the "mainframe" environments.

For files that contain only text information, carriage return and line feed characters are commonly used to delimit records.

```
0123456789
0123456789
0123456789
```

For files containing binary information, there must be a way to decipher the information. In many PC products, the internal format of the file is known by the application processing the file. This is the case for programs such as Microsoft Word® and Excel®. The contents of a spreadsheet saved in Excel® is generally understood only by the Excel® application. In most cases, it would not make sense to "sort" an Excel spreadsheet directly, though it may be desirable to sort a range of rows and columns within the spreadsheet.

In this case you must convert the file to a format that the sorting utility can understand. This would most likely be a "text" file.

In the Unix, OS/2® and DOS/Windows® environments, the application is generally responsible for the formatting of the records and the file.

Since the Unix, OS/2® and DOS/Windows® operating systems do not maintain information about the format of the records in a file, it is common to use delimiters between records. These delimiters are usually the ASCII Carriage Return (Hex 0D) and Line Feed (Hex 0A)characters. Files of this type are referred to as "Text" files. Text files generally contain only ASCII character data. This is because binary characters that may occur in the file could be interpred as record delimiters.

In fixed format files, each record contains a fixed number of bytes. If a file contains 10,000 records that are 80 bytes in length, then the total amount of disk space consumed by that file will be 800,000 bytes. This is true regardless of the operating environment.

In "text" format files, each record contans a variable number of bytes. The end of the record is indicated by the presence of a Carriage Return (CR) and possibly a Line Feed (LF). These CR/LF characters are used to determine where the record ends. These characters really exist in the file on disk, but they are generally not seen because most products such as editors do not display them.

If you create a new file with Windows® Notepad, and key the following records, pressing *ENTER* after each of the three records:

If you do a directory command on this file the file will contain 36 characters, even though you only typed 30 characters. The six additional charcters are the *Carriage Return* and *Line Feed* charcters at the end of each line.

Some editors add an additional *End of File* character after the last record in the file. Assort ignores this character when you specify the Text record format.

When Assort reads a file of the Text record format, records are created by accumulating bytes until a Carriage Return/Line Feed combination is encountered. These characters are not actually part of the record. They only serve as delimiters between records in the file.

ASCII and EBCDIC

ASCII and EBCDIC are character sets that are commonly used to encode character information.

EBCDIC stands for Extended BCD Interchange Code. BCD refers to Binary Coded Decimal. EBCDIC is used predomitely on IBM computers such as the System/390®, AS/400® and RS/6000®.

ASCII stands for American National Standard Code for Information Interchange. ASCII is used on most all computers include IBM and APPLE personal computers.

When converting between ASCII and EBCDIC, be aware that there is not a one to one mapping of characters. The only time files should be converted is when they contain only character data (A-Z, 0-9 and some punctuation characters). Data in binary and packed decimal fields will probably be translated improperly.

Assort can operate directly on Packed Decimal (COBOL COMP-3), Zoned Decimal and signed or unsigned binary fields. Sorting or summing on these fields should be performed prior to translation to ASCII. Numeric fields such as these can be translated to their character representations with the EDIT option of the INREC/OUTREC Statements.

If it is desired to sort the data based on its ASCII collating sequence, you can use the AC data format option on the SORT/MERGE Statement. This option converts the EBCDIC characters to ASCII prior to performing comparisons on the records.

The ASCII and EBCDIC character sets are documented in Appendix A of this manual.

The character set is generally of little importance to the end user, but becomes a significant issue when transporting files from one platform to another. If both platforms use the same character set, there is no issue. When the two platforms use different character sets, you must be aware of this and take appropriate action to convert the data so that it is properly interpreted when read by application programs.

Assort has the ability to perform conversion from EBCDIC to ASCII. This allows the user to read the converted file into an application that only recognizes the ASCII character set.

An example of the need to convert data would be if you downloaded a file encoded with the EBCDIC character set from an IBM mainframe and wanted to sort reformat it so that it could be processed by another PC based application. The procedure would be as follows:

Identify the record length and record format of the mainframe base file.

Download the using a *Binary* file transfer utility.

Sort and/or reformat the records using Assort.

Specify the EBCDIC2ASCII and CRLF OUTREC options.

Process the file with the target application.

Assort Operations

Sorting

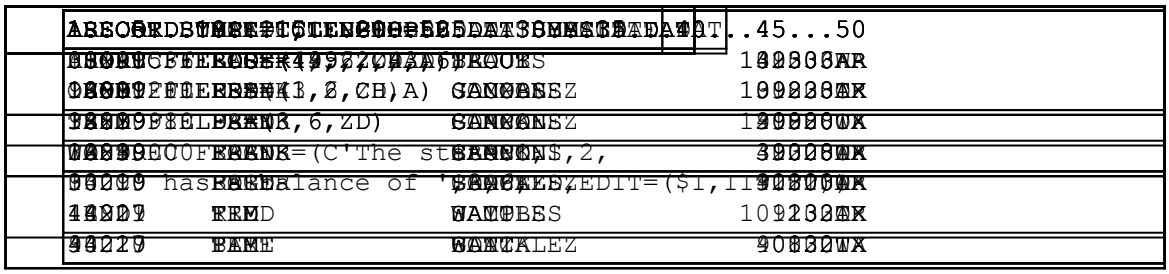
Sorting is the process of arranging a set of records into a defined sequence. The resulting sequence of records can be ascending (non-descending) or descending (non-ascending).

If you had a text file that had check book transactions in it, you may want to sort the file by check number or by date, or by date and check number.

The following examples will use the following file layout:

Field Name	Start Position	End Position	Field Length	Field Format
Account Number	1	5	5	Zoned Decimal
First Name	10	24	15	Character
Last Name	25	42	18	Character
Account Balance	43	48	6	Zoned Decimal (two implied decimal places)
State Code	49	50	2	Character

The input file contains the following records (PEOPLE.DAT):



Sort Staments (BYACCT.CTL):

Sort Command Line:

Output File (BYACCT.DAT):

In this example the input file is sorted by position one for five characters in ascending sequence. This is the position of the account number in the input record. The resulting file contains the same records, but ordered by account number.

If you wanted the file sorted by State Code, you would specify:

Sort Staments (BYSTATE.CTL):

Sort Command Line:

Output File (BYSTATE.DAT):

If you wanted the file sorted by descending account balance, you would specify:

Sort Staments (BYBALD.CTL):

Sort Command Line:

Output File (BYBALD.DAT):

Multiple records can be combined into a single record with the SUM function of the sort. This can be used to add up all of the values in a field for records with equal keys.

To obtain the total of all accounts by state, specify:

Sort Staments (SUMST.CTL):

Sort Command Line:

Output File (SUMST.DAT):

Note that the output record contains account number, first name and last name. These fields are not relevant. They contain the contents of one of the equal keyed records.

If you wanted the sort to be more efficient, you could eliminate unnecessary fields on input. The fields in the record can also be moved.

Sort Staments (SUMST1.CTL):

Sort Command Line:

Output File (SUMST1.DAT):

The output record now contains the state code and the total of all accounts for that state. The output record contains eight bytes.

To make the output even more appealing, editing can be used.

Sort Staments (SUMST2.CTL):

Sort Command Line:

Output File (SUMST2.CTL):

```
ABSORDMERGE=C5LENQ294=DA5,Q290.DA35Q3940DAT45Q49450DAT MERGED.DAT
AM8R6E98E:AR88N8Q8,Q8A,A2ce of $1,415.36
A28D9e2e:0K6H888a1b94a2ce of    $12.00
B26VA0Be:TX92B97a1b94a2ce of $1,299.80
F0699981e:WA88B19a1b94a2ce of   $400.00
X0000091    56717 1994 2
A2389000   798278 1994 3
A9238201  1998288 1994 4
B009A00D      390 1994 4
F0399992  1702900 1994 1
F0399992    99209 1994 2
F0399992   663526 1994 3
F0399992    27177 1994 4
H3392981   888819 1994 2
Y0000091    23110 1994 1
Y0000091    54497 1994 2
Y0000091      207 1994 3
Y0000091    56712 1994 4
ZZ389000   798273 1994 4
```

The output record contains 39 bytes.

Merging

Merging is the process of combining a number of sorted sequences of records into a single sorted sequence.

Suppose you had one file for each quarter of the year that contained the purchase each of your customers made sorted by account number. The files could all be read into a sort and sorted by account number, but it would be much more efficient to simply merge the files together.

The following examples will use the following file layout:

Field Name	Start Position	End Position	Field Length	Field Format
Customer Account ID	1	8	8	Character
Purchase Total	9	17	9	Zoned Decimal (two implied decimal places)
Year	19	22	4	Zoned Decimal
Quarter	24	24	1	Zoned Decimal

The input files contains the following records:

File for the first quarter of 1994 (Q194.DAT)

File for the second quarter of 1994 (Q294.DAT)

File for the third quarter of 1994 (Q394.DAT)

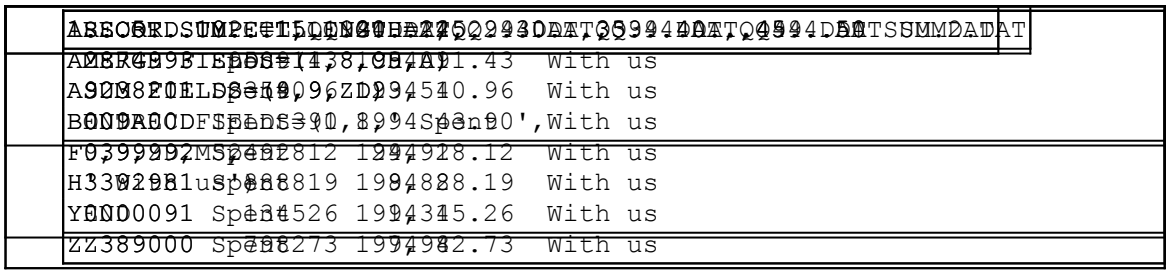
File for the fourth quarter of 1994 (Q494.DAT)

Sort Staments (MERGE.CTL):

Sort Command Line:

Output File (MERGED.DAT):

To perform summing by account number in the merge process, code the following control information:



Sort Staments (SUM.CTL):

Sort Command Line:

Output File (SUM.DAT):

To perform summing by account number in the merge process, and use the built in edit mask (M5) to reformat records, code the following control information:

Sort Staments (SUM2.CTL):

Sort Command Line:

Output File (SUM2.DAT):

Copying

The copy process is generally performed when the records need to be duplicated, but the order is not to be changed. The records can be reformatted during the copy process to create a file that contains a subset of the fields in the input file.

Copying can be performed for a single or multiple input files.

Reformatting

Reformatting is used to change the presentation of the data in a file. For example, the widths of fields can be truncated or expanded, numeric fields can be edited to make them more readable.

Appendix A

Code Tables

Decimal	Hex	EBCDIC	ASCII	Binary
0	00	NUL	NUL	0000 0000
1	01	SOH	SOH	0000 0001
2	02	STX	STX	0000 0010
3	03	EXT	ETX	0000 0011
4	04	SEL	EOT	0000 0100
5	05	HT	ENQ	0000 0101
6	06	RNL	ACK	0000 0110
7	07	DEL	BEL	0000 0111
8	08	GE	BS	0000 1000
9	09	SPS	HT	0000 1001
10	0A	RPT	LF	0000 1010
11	0B	VT	VT	0000 1011
12	0C	FF	FF	0000 1100
13	0D	CR	CR	0000 1101
14	0E	SO	SO	0000 1110
15	0F	SI	SI	0000 1111
16	10	DLE	DLE	0001 0000
17	11	DC1	DC1	0001 0001
18	12	DC2	DC2	0001 0010
19	13	DC3	DC3	0001 0011
20	14	RES/ENP	DC4	0001 0100
21	15	NL	NAK	0001 0101
22	16	BS	SYN	0001 0110
23	17	POC	ETB	0001 0111
24	18	CAN	CAN	0001 1000
25	19	EM	EM	0001 1001
26	1A	UBS	SUB	0001 1010
27	1B	CU1	ESC	0001 1011
28	1C	IFS	FS	0001 1100
29	1D	IGS	GS	0001 1101
30	1E	IRS	RS	0001 1110
31	1F	ITB/IUS	US	0001 1111
32	20	DS	SP	0010 0000
33	21	SOS	!	0010 0001
34	22	FS	"	0010 0010
35	23	WUS	#	0010 0011
36	24	BYP/INP	$	0010 0100
37	25	LF	%	0010 0101
38	26	ETB	&	0010 0110
39	27	ESC	'	0010 0111
40	28	SA	(	0010 1000
41	29	SFE	)	0010 1001
42	2A	SM/SW	*	0010 1010
43	2B	CSP	+	0010 1011
44	2C	MFA	,	0010 1100
45	2D	ENQ	_	0010 1101
46	2E	ACK	.	0010 1110
47	2F	BEL	/	0010 1111
48	30		0	0011 0000
49	31		1	0011 0001
50	32	SYN	2	0011 0010
51	33	IR	3	0011 0011
52	34	PP	4	0011 0100
53	35	TRN	5	0011 0101
54	36	NBS	6	0011 0110
55	37	EOT	7	0011 0111
56	38	SBS	8	0011 1000
57	39	IT	9	0011 1001
58	3A	RFF	:	0011 1010
59	3B	CU3	;	0011 1011
60	3C	DC4	<	0011 1100
61	3D	NAK	=	0011 1101
62	3E		>	0011 1110
63	3F	SUB	?	0011 1111

Code Tables (Continued)

Decimal	Hex	EBCDIC	ASCII	Binary
64	40	SP	SP	0100 0000
65	41	RSP	A	0100 0001
66	42		B	0100 0010
67	43		C	0100 0011
68	44		D	0100 0100
69	45		E	0100 0101
70	46		F	0100 0110
71	47		G	0100 0111
72	48		H	0100 1000
73	49		I	0100 1001
74	4A	¢	J	0100 1010
75	4B	.	K	0100 1011
76	4C	<	L	0100 1100
77	4D	(	M	0100 1101
78	4E	+	N	0100 1110
79	4F	|	O	0100 1111
80	50	&	P	0101 0000
81	51		Q	0101 0001
82	52		R	0101 0010
83	53		S	0101 0011
84	54		T	0101 0100
85	55		U	0101 0101
86	56		V	0101 0110
87	57		W	0101 0111
88	58		X	0101 1000
89	59		Y	0101 1001
90	5A	!	Z	0101 1010
91	5B	$	[	0101 1011
92	5C	*	\	0101 1100
93	5D	)	]	0101 1101
94	5E	;	^	0101 1110
95	5F	¬	_	0101 1111
96	60	-	`	0110 0000
97	61	/	a	0110 0001
98	62		b	0110 0010
99	63		c	0110 0011
100	64		d	0110 0100
101	65		e	0110 0101
102	66		f	0110 0110
103	67		g	0110 0111
104	68		h	0110 1000
105	69		i	0110 1001
106	6A	¦	j	0110 1010
107	6B	,	k	0110 1011
108	6C	%	l	0110 1100
109	6D	_	m	0110 1101
110	6E	>	n	0110 1110
111	6F	?	o	0110 1111
112	70		p	0111 0000
113	71		q	0111 0001
114	72		r	0111 0010
115	73		s	0111 0011
116	74		t	0111 0100
117	75		u	0111 0101
118	76		v	0111 0110
119	77		w	0111 0111
120	78		x	0111 1000
121	79	`	y	0111 1001
122	7A	:	z	0111 1010
123	7B	#	}	0111 1011
124	7C	@	¦	0111 1100
125	7D	'	}	0111 1101
126	7E	=	~	0111 1110
127	7F	"	DEL	0111 1111

Code Tables (Continued)

Decimal	Hex	EBCDIC	ASCII	Binary
128	80			1000 0000
129	81	a		1000 0001
130	82	b		1000 0010
131	83	c		1000 0011
132	84	d		1000 0100
133	85	e		1000 0101
134	86	f		1000 0110
135	87	g		1000 0111
136	88	h		1000 1000
137	89	i		1000 1001
138	8A			1000 1010
139	8B			1000 1011
140	8C			1000 1100
141	8D			1000 1101
142	8E			1000 1110
143	8F			1000 1111
144	90			1001 0000
145	91	j		1001 0001
146	92	k		1001 0010
147	93	l		1001 0011
148	94	m		1001 0100
149	95	n		1001 0101
150	96	o		1001 0110
151	97	p		1001 0111
152	98	q		1001 1000
153	99	r		1001 1001
154	9A			1001 1010
155	9B			1001 1011
156	9C			1001 1100
157	9D			1001 1101
158	9E			1001 1110
159	9F			1001 1111
160	A0			1010 0000
161	A1	~		1010 0001
162	A2	s		1010 0010
163	A3	t		1010 0011
164	A4	u		1010 0100
165	A5	v		1010 0101
166	A6	w		1010 0110
167	A7	x		1010 0111
168	A8	y		1010 1000
169	A9	z		1010 1001
170	AA			1010 1010
171	AB			1010 1011
172	AC			1010 1100
173	AD			1010 1101
174	AE			1010 1110
175	AF			1010 1111
176	B0			1011 0000
177	B1			1011 0001
178	B2			1011 0010
179	B3			1011 0011
180	B4			1011 0100
181	B5			1011 0101
182	B6			1011 0110
183	B7			1011 0111
184	B8			1011 1000
185	B9			1011 1001
186	BA			1011 1010
187	BB			1011 1011
188	BC			1011 1100
189	BD			1011 1101
190	BE			1011 1110
191	BF			1011 1111

Code Tables (Continued)

Decimal	Hex	EBCDIC	ASCII	Binary
192	C0	{		1100 0000
193	C1	A		1100 0001
194	C2	B		1100 0010
195	C3	C		1100 0011
196	C4	D		1100 0100
197	C5	E		1100 0101
198	C6	F		1100 0110
199	C7	G		1100 0111
200	C8	H		1100 1000
201	C9	I		1100 1001
202	CA	SHY		1100 1010
203	CB			1100 1011
204	CC			1100 1100
205	CD			1100 1101
206	CE			1100 1110
207	CF			1100 1111
208	D0	}		1101 0000
209	D1	J		1101 0001
210	D2	K		1101 0010
211	D3	L		1101 0011
212	D4	M		1101 0100
213	D5	N		1101 0101
214	D6	O		1101 0110
215	D7	P		1101 0111
216	D8	Q		1101 1000
217	D9	R		1101 1001
218	DA			1101 1010
219	DB			1101 1011
220	DC			1101 1100
221	DD			1101 1101
222	DE			1101 1110
223	DF			1101 1111
224	E0	\		1110 0000
225	E1	NSP		1110 0001
226	E2	S		1110 0010
227	E3	T		1110 0011
228	E4	U		1110 0100
229	E5	V		1110 0101
230	E6	W		1110 0110
231	E7	X		1110 0111
232	E8	Y		1110 1000
233	E9	Z		1110 1001
234	EA			1110 1010
235	EB			1110 1011
236	EC			1110 1100
237	ED			1110 1101
238	EE			1110 1110
239	EF			1110 1111
240	F0	0		1111 0000
241	F1	1		1111 0001
242	F2	2		1111 0010
243	F3	3		1111 0011
244	F4	4		1111 0100
245	F5	5		1111 0101
246	F6	6		1111 0110
247	F7	7		1111 0111
248	F8	8		1111 1000
249	F9	9		1111 1001
250	FA			1111 1010
251	FB			1111 1011
252	FC			1111 1100
253	FD			1111 1101
254	FE			1111 1110
255	FF	EO		1111 1111