DataGeneral

Sort/Merge Utility User's Manual (AOS)

093-000155-00

Sort/Merge Utility User's Manual (AOS)

093-000155-00

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 093-000155

Printed in the United States of America
Revision 00, June 1978

NOTICE

Sort/Merge Utility User's Manual
(AOS)
093-000155

Revision History:

Original Release - June 1978

Preface

This manual tells you how to use the various required and optional functions of the Sort/Merge Utility. It assumes you are familiar with sorting and merging in general, and with the AOS System and INFOS® System files. Consult the *AOS Software Documentation Guide* (093-000202) to determine other AOS manuals with which you should be familiar before using this utility. We have arranged the manual as follows:

- Chapter 1 gives an overview of the utility's capabilities.
- Chapter 2 tells how to invoke the utility.
- Chapter 3 describes the command file format and syntax.
- Chapter 4 tells how to specify command file declarations.
- Chapter 5 tells how to edit records and specify imperatives.
- Chapter 6 gives summary examples of sorting, merging, and copying AOS files.
- Chapter 7 tells how to define INFOS® input and output files.
- Appendix A describes the statistical information the utility returns after each successful invocation.
- Appendix B shows standard character sets.
- Appendix C summarizes the commands described in this manual.

Reader, Please Note:

We use these conventions for command formats in this manual:

COMMAND required *[optional]* ...

Where	Means
COMMAND	You must enter the command (or its accepted abbreviation) as shown.
required	You must enter some argument (such as a filename). Sometimes, we use: $\left\{\begin{matrix}\text{required}_1\\ \text{required}_2\end{matrix}\right\}$ which means you must enter *one* of the arguments. Don't enter the braces; they only set off the choice.
[optional]	You have the option of entering some argument. Don't enter the brackets; they only set off what's optional.
...	You may repeat the preceding entry or entries. The explanation will tell you exactly what you may repeat.

Additionally, we use certain symbols in special ways:

Symbol	Means
)	Press the NEW-LINE or RETURN key on your terminal's keyboard.
□	Be sure to put a space here. (We use this only when we must; normally, you can see where to put spaces.)

All numbers are decimal unless we indicate otherwise; e.g., 35_8.

Finally, we usually show all examples of entries and system responses in THIS TYPEFACE. But, where we *must* clearly differentiate your entries from system responses in a dialog, we will use

THIS TYPEFACE TO SHOW YOUR ENTRY)
THIS TYPEFACE FOR THE SYSTEM RESPONSE

End of Preface

Contents

Chapter 1 - Introduction

Easy to Use and Versatile 1-1
Sorting 1-1
Merging and Copying 1-1
Filtering and Massaging Records 1-1
Skipping Records 1-2
Reformatting Records 1-2
Replacing the Contents of a Record's Fields 1-2
Inserting New Fields 1-2
Padding Records 1-2
Translating Records 1-2
Compressing 1-2
Altering the Collating Sequence 1-2
Key Types 1-3
Device Types 1-3
Record Types 1-3

Chapter 2 - Operating Considerations

Global Switches 2-1
/S Controlling the Statistical Output 2-1
/L and /L=filename Controlling the Output Listing 2-1
/O Controlling the Output File 2-2
/C, /T=filename, and /C=filename Controlling the Command File 2-2
/N Controlling the Execution of the Utility 2-2
The Basic Sort 2-2
The Basic Merge 2-3
Tailoring the Utility's Operation 2-4

Chapter 3 - Command File Format and Syntax

General Information 3-1

Chapter 4 - Command File Declarations

AOS Input File Declaration 4-1
AOS Output File Declaration 4-3
Table Declaration 4-4
Key Declaration 4-8
Work File Declaration 4-9

Chapter 5 - Command File Directive-Group

Massage Statements 5-1
REFORMAT 5-1
REPLACE 5-2
INSERT 5-3
PAD 5-5
TRANSLATE 5-5
COMPRESS 5-6
IF 5-7
Imperatives 5-10
Sort and Stable Sort 5-10
Tag Sort and Stable Tag Sort 5-10
Merge 5-10
Copy 5-10

Chapter 6 - Summary Examples

Chapter 7 - INFOS Files

INFOS Input File Declaration 7-1
INFOS Output File Declaration 7-6
Examples 7-10

Appendix A - Statistical Information Returned by the Utility

Appendix B - Character Set

Appendix C - Command Summary

Illustrations

Figure	Caption	
7-1	Simple DBAM Index, EXAMPLE	7-4
7-2	Multi-indexed Database	7-7
7-3	Simple DBAM Index, SAMPLE	7-8
7-4	DBAM Database with Two Subindexes	7-9
7-5	Simplified Multikey ISAM/DBAM File, MASTER	7-11
7-6	DBAM File EMPLOYEE	7-13
A-1	Merge Passes	A-4

Tables

Table	Caption	
4-1	Simplified ASCII Collating Table	4-5
A-1	Statistics Produced per Operational Stage	A-1

Chapter 1
Introduction

You can execute three common data processing tasks with this utility program: sorting, merging, and copying AOS and INFOS® files. You work at a console using simple, easy to learn commands. The utility's options let you choose from four different sorting operations, or the merge or copy operation. Moreover, you can edit either a few or all your records. We call the process of editing records *"massaging"* and the process of selecting them *"filtering."*

Easy to Use and Versatile

You can execute some of your sorting and merging tasks just by invoking the utility: you simply specify SORT or MERGE and the names of the input and output files right on the CLI command line. A utility command line option lets you use one of your input files as the output file, so you can sort, merge, or copy a file into itself. For example, suppose you have two previously sorted files on magnetic tape and a master file on disk. If you want to merge the records in all three files into your master file, just specify the appropriate command line option and name the master file as one of the input files and as the output file.

To take full advantage of the utility's options you'll use a *command file.* You can create the command file right at the console when you invoke the utility. Or you can create it with a text editor and specify its name in the CLI command line when you invoke the utility. Alternatively, you can create a command file at the console during one invocation, save it, and specify its name for a subsequent invocation. Thus, for repetitive tasks you need to define a command file only once. Each time you want to execute that task, simply invoke the utility and specify the command file's name on the CLI command line.

Sorting

The four sort operations you can choose from are: a standard sort, a stable sort, a tag sort, and a stable tag sort.

In *standard* and *stable sorts* the utility retrieves the entire record and carries it throughout all phases of the utility's operation. In general, a standard sort is somewhat faster than a stable sort. But, in a stable sort the utility guarantees that input records whose sort keys are exact duplicates are written to the output file in exactly the order that the utility encounters them in the input files. In a *tag sort,* the utility creates a tag identifying each input record and carries the tags through to the output phase. During the output phase the utility retrieves the records and writes them to the output file. Because the system is not carrying each entire record through every tag sort phase, a tag sort requires less disk space for its execution. You can, therefore, sort more records in the same amount of disk space with a tag sort than you can with a standard sort. The difference between a *stable tag sort* and a tag sort is the same as between a standard sort and a stable sort.

Merging and Copying

You can merge the contents of two or more sorted files into a single output file. You can also copy the unordered contents of one or more input files to a single output file.

Filtering and Massaging Records

You can massage records during the utility's input and/or output phase. However, filtering generally makes sense only in the input phase. To filter input records, you simply define one or more conditions which, if met, cause the utility to perform the massaging operation you specify. For example, you may specify that if a particular field in a record contains some special character, like $, the utility should move that field to some other location in the record.

Skipping Records

You can tell the utility to skip if a condition is met. Skipping simply means you don't want the utility to write the input record that meets your specified condition to the output file. You can also save skipped records in a secondary output file, called a *skip file*. Thus, with the filtering option, you can create subsets of the input records.

Besides the filtering and skipping operations, there are six unique record massaging operations, as described below.

Reformatting Records

At times you'll want to rearrange the order of a record's fields. To do so, simply use a REFORMAT statement to specify the order you want. You can also delete fields from a record simply by not specifying them in a REFORMAT statement. When you reformat a record, its fields remain the same length; that is, you can't expand or contract reformatted fields.

Replacing the Contents of a Record's Fields

You can expand or contract field lengths with a REPLACE statement. For example, you may wish to replace a field whose content is ACCTS with a new field containing ACCOUNT NUMBERS; or, you may wish to replace PART NUMBER with PN.

Inserting New Fields

You can create new fields or insert control characters in a record with the INSERT statement. For example, you may wish to insert a field containing NAME: or one containing AMOUNT_DUE: in an appropriate place in a record. And, you may wish to insert a line feed after certain fields for formatting printed output.

Padding Records

You can convert variable-length records to fixed-length with the PAD statement. You simply specify your desired record length and the padding character.

Translating Records

With a TRANSLATE statement, you can specify that a record or selected fields of a record be translated from ASCII to EBCDIC or from EBCDIC to ASCII. Or, you can translate to or from a recording code which you supply. You may also translate lowercase ASCII letters to uppercase.

Compressing

You can use the COMPRESS statement to exclude certain characters from consideration in determining a record's position in the output. For example, your key field may contain a special character, like /. If you don't want the utility to consider the / when determining a record's output position, you can compress the key field to remove the / during input and reinsert it during output.

Altering the Collating Sequence

Normally, the utility collates your output in ascending sequence according to ASCII character value. For each sort or merge key you define, you can specify either the ascending or the descending sequence. Optionally, you can define a table in which you assign characters alternative collating values, and tell the utility to order your output using that table.

Key Types

When you use a command file you'll define one or more keys which the utility will use to order your output records. For example, if you want to sort input records by employee names, you can specify that the record's surname field contains the primary key, its first name field contains a secondary key, and its middle initial field contains a tertiary key. The sequence in which you define keys is the order in which the utility sorts or merges them. Also note that you can specify an ascending ASCII sequence for one key and a descending sequence for another.

The utility accepts a variety of key types. The default key type is essentially the same as a COBOL alphanumeric data type; we call it a *character* type key. You have the option of specifying any of the COBOL data types as your key type. And, if you specify a *decimal* type key, you can also specify leading or trailing overpunch and/or leading or trailing signs.

Device Types

If you have high speed devices, such as fixed-head disks, you can tell the utility to build its work files on those devices. Your input files can be on any device known to your AOS system and, in general, you can direct your output to any device. However, you must direct your INFOS output to a disk file.

Record Types

Input files can have fixed-length, AOS variable-length or AOS data-sensitive records. You can specify input files which have different record formats and the utility will create an output file with an appropriate record format. For example, if one of your input files has fixed-length records and another has variable-length records, the utility automatically creates an output file with the variable-length record format. Or, if all your input files have data-sensitive records, for example, the utility automatically creates an output file with the data-sensitive record format characteristic.

End of Chapter

Chapter 2
Operating Considerations

The CLI command you use to invoke the utility has three basic formats:

1. SORT [{ /L / /L=*filename* }] *[/N] [/O] [/S]* INTO outfile FROM infile ...

2. MERGE [{ /L / /L=*filename* }] *[/N] [/O] [/S]* INTO outfile FROM infile$_1$ infile$_2$ ...

3. { SORT / MERGE / FILTER } { /C *[/T=filename]* / /C=filename } [{ /L / /L=*filename* }] *[/N] [/O] [/S]* [*INTO outfile [FROM infile ...]*]

When you use either of the first two formats you must specify the operation (SORT or MERGE), where to put the output (INTO outfile), and where to find the input (FROM infile). When you use the third format you must specify one of the utility names (SORT, MERGE, or FILTER) and the global switch /C. Since there are several global switches common to each format, we'll describe those first; then we'll go into more detail about each command format.

Global Switches

The global switches control such utility operations as whether to generate a statistics file for the current run and where to store your interaction with the utility.

/S Controlling the Statistical Output

The utility automatically outputs statistics each time it runs. We describe these statistics in Appendix A. Normally, the utility displays the statistics at your console. You can, however, suppress all statistical output by including the global switch /S on the CLI command line. This is especially useful when you invoke the utility from another program, such as a COBOL application program.

/L and /L=filename Controlling the Output Listing

Normally, the utility displays your interaction with it on your console. However, you can tell the utility to write this information and the statistical output to a list file by using the global switch /L on the CLI command line. This tells the utility to write the output either to the AOS-defined @LIST file (by using a plain /L), or to the file you specify as filename (by including /L=filename). The file you specify need not exist; if it doesn't, the utility creates an AOS file with the specified name. If the file already exists, the utility appends the output for the current run to the end of it.

/O Controlling the Output File

Normally you specify the name of a *nonexistent* output file for a sort or merge operation on AOS input files. The utility then automatically creates an AOS file with the proper characteristics, which it derives from the input file's characteristics. However, you may specify an existing output file by including the global switch /O on the CLI command line. Then the utility deletes the file and recreates it with the output of the sort or merge operation. This, in effect, allows you to sort, merge, or copy a file into itself. The /O switch also allows you to direct your output to a predefined AOS file such as the line printer or a magnetic tape.

/C, /T=filename, and /C=filename Controlling the Command File

You'll use format 3 of the CLI command to tailor the utility's operation. When you use this format, you'll specify directives in a command file. (We describe command files in Chapter 3.) The /C switch specifies that you'll enter the command file at your console. If you also include /T=filename on the same command line, the utility writes the command file to the trail file, filename. This gives you a permanent copy of the directives which you can use as the command file to call the utility later. To use a previously created command file, specify /C=filename on the CLI command line.

/N Controlling the Execution of the Utility

When you specify the global switch /N the utility processes all input commands (i.e., the CLI command line and, possibly, the command file,) but does not execute the requested function. This allows you to detect errors in the input commands prior to attempting execution, or to create a command file for later execution.

The Basic Sort

Use format 1 of the CLI command to call the basic sort, which sorts the records in one or more AOS input files into a single AOS output file. The basic sort sorts *all* records but performs no massaging functions. The sort key for each record is the entire input record treated as a byte string. The utility writes the output records according to the ascending ASCII sequence of these sort keys.

If you want to use the basic sort, all your input files must have either fixed-length or data-sensitive records. If an input file has data-sensitive records, they cannot exceed 136 characters and must be delimited by one of the AOS default delimiters: null, line feed, carriage return, or form feed. To sort an AOS input file whose data-sensitive records exceed 136 characters or are not delimited by a default delimiter, you must supply a command file in which you define the input file in an AOS Input File Declaration (described in Chapter 4). Also, you cannot sort AOS files containing variable-length records by invoking the basic sort; again, you must provide a command file in which you describe the input file in an AOS Input File Declaration.

If you want to use one of the input files as the output file, specify the /O switch on the command line. (You should specify the /O switch if the output file exists, even though it's not an input file.) When you specify /O, the utility deletes the existing file and recreates it with your sorted output. If you don't specify /O and the output file exists, you'll get an error. In most cases, however, you'll specify an output file name and the utility will create a new file with that name. The utility creates the output file according to the following criteria:

- If all the input files have fixed-length records of the same length, the utility creates an output file with that fixed-length record's characteristics.
- If all the input files have fixed-length records but the record lengths vary, the utility creates an output file with the AOS variable-length record characteristic.
- If all the input files have data-sensitive records delimited by null, carriage return, line feed, or form feed, the utility creates an output file with data-senstive records. If the delimiters are not standard, you must define the input file in an AOS Input File Declaration.
- If the record format of one input file differs from that of any other input file, the utility creates an output file with variable-length records.

Suppose, for example, you have two transaction files, TR1 and TR2, which you want to sort using the input file TR1 as the output file. You could specify:

```
SORT/O INTO TR1 FROM TR1 TR2
```

If you have a scratch file that you normally use for temporary storage, you can sort into it by specifying:

```
SORT/O INTO SCRATCH FROM TR1 TR2
```

If you want to create a new output file to contain the sorted records, you could specify:

```
SORT INTO NEW_FILE FROM TR1 TR2
```

Now let's say that you have a command file, CFILE, which contains the names of the input files. Let's also assume that you want to send the statistics about this operation, along with a copy of the command file, to a file called MYFILE. Finally, let's call your output file OUTFILE. To do all this, you'd type:

```
SORT/C=CFILE/L=MYFILE/O INTO OUTFILE
```

You could also specify both the input *and* output files in the command file, which would change the command line to:

```
SORT/C=CFILE/L=MYFILE/O
```

Alternatively, you might keep the names of one or more input files in a command file, CMFILE, and want to sort those files with the contents of a separate input file (or files). To do this, you'd type:

```
SORT/C=CMFILE/O INTO OUTFILE FROM INFILE1
```

or

```
SORT/C=CMFILE/O INTO OUTFILE FROM INFILE1 INFILE2
```

and so forth.

The Basic Merge

Use format 2 of the CLI command to call the basic merge, which merges the records of two or more previously sorted AOS input files into a single AOS output file. If you want to use the basic merge, you must have sorted all your input files into ascending sequence, using the entire record as the sort key. Normally, the input files will be the output from two or more basic sort operations. Basic merge merges *all* input records but performs *no* massaging functions. The merge key for each record is the entire record treated as a byte string. The utility writes the output records in ascending ASCII sequence of these merge keys.

If you want to use the basic merge, all input files must conform to the standards previously described for the basic sort input files. The basic merge output file also conforms to these file standards.

Suppose, for example, you have two sorted transaction files, TR1 and TR2, which you want to merge into a master file. Assuming the master file exists, you could specify:

```
MERGE/O INTO MASTER FROM TR1 TR2
```

If you want to merge the two sorted transaction files and direct your output to the line printer, you could specify:

```
MERGE/O INTO @LPT FROM TR1 TR2
```

If you want to merge one or more files whose names you keep in a command file, COFILE, with TR1 and TR2, and then direct your output to a master file, you'd type:

```
MERGE/C=COFILE/O INTO MASTER FROM TR1 TR2
```

For further examples, simply substitute MERGE for SORT in the examples described in *The Basic Sort* section of this chapter. In other words, you can do everything in a merge that you can do in a sort.

Tailoring the Utility's Operation

To execute something other than the basic sort or merge, you must supply a *command file.* The next several chapters explain how to specify the contents of the command file. Use format 3 of the CLI command to tell the utility to accept a command file.

You specify the *operation* you want performed (Sort, Stable Sort, Tag Sort, Stable Tag Sort, Merge, or Copy) in the command file. Therefore, you can use any of the *utility names* (Sort, Merge, or Filter) when calling the utility to accept a command file. You must, however, specify either /C or /C=filename on the command line. The /C switch indicates that you'll enter the command file line-by-line at your console. If you follow /C with /T=filename the utility writes the command file to filename. When you specify /C=filename the utility accepts input directives from a previously created command file. You may have created a command file through a text editor, or during a previous evocation of the utility, that is, when you've specified /C and /T=filename on the same command line.

Note that if you make minor errors while entering the command file at your console and if you didn't specify /T=filename on the command line, the utility asks if you want to save your command file in a trail file (i.e., an unedited command file). This option allows you to correct errors and/or respecify lines in the command file with a text editor. Then you can reinvoke this utility, specifying /C=trailfilename to execute your desired operation. If you make a major error while entering your command file, the utility may continue to prompt for input after you've entered END. Should this occur enter CTRL-D to terminate your interaction with the utility; you will still have the option of saving your command file for editing and resubmission.

You may name an AOS output file on the command line but you must define each INFOS output file in an INFOS Output File Declaration (described in Chapter 7), within the command file. If the AOS output file does not exist, the utility creates it according to the standards previously described for the basic sort output file. If the AOS output file does exist, you should use the /O switch, which allows you to sort, merge, or copy a file into itself, or to direct your output to an AOS-defined file like the line printer. If the AOS output file exists and you forget the /O switch, and if that file is not also an input file, the utility tells you that the file exists (if you asked for interaction by using the /C switch). It also gives you the option of telling the utility to delete and recreate the output file automatically, or of storing your command file in a trail file so you can edit it.

You may name one or more AOS input files on the command line but you must define all INFOS input files in INFOS Input File Declarations (described in Chapter 7). The AOS input files must conform to the standards previously described for the basic sort input files; otherwise, you must define them in AOS Input File Declarations (described in Chapter 4).

If you are going to name one or more input files on a command line, note that you must first specify the name of the output file. That is, you cannot name input files on the command line and define the output file in a command file. Furthermore, you cannot use a command file to further describe *any* files (either input or output) which you name on the command line. For example:

```
SORT/O INTO OFILE1 FROM IFILE1 IFILE2
```

is legal, but

```
SORT/C=CFILE1/O FROM IFILE1 IFILE2
```

is *not.* Also,

```
SORT/C=CFILE1/O INTO OFILE1 FROM IFILE1 IFILE2
```

is *not* legal if CFILE is an output file.

End of Chapter

Chapter 3
Command File Format and Syntax

A command file consists of an optional series of declarations followed by a required directive-group, followed by an END statement; in other words:

[DECLARATION]... DIRECTIVE-GROUP END.

The declarations allow you to specify the names and characteristics of AOS or INFOS input files, an AOS or INFOS output file, tables, work files, and sort, merge, or copy keys. The declarations sequence is:

$$\left[\left\{\begin{matrix}\textit{AOS INPUT FILE}\\ \textit{INFOS INPUT FILE}\end{matrix}\right\}\right]\ldots\left[\left\{\begin{matrix}\textit{AOS OUTPUT FILE}\\ \textit{INFOS OUTPUT FILE}\end{matrix}\right\}\right]\ \textit{[TRANSTABLE]}\ \ldots\ \textit{[KEY]}\ \ldots\ \textit{[WORKFILE]}\ \ldots$$

The directive-group consists of a series of optional input record massage statements followed by a single required imperative, followed by a series of optional output massage statements. (Remember that the directive-group is followed by END.) The directive-group is:

$$\left[\textit{[REFORMAT] [REPLACE] [INSERT] [PAD\_TO] [TRANSLATE] [COMPRESS] [IF]}\right]\ldots\left\{\begin{matrix}\text{SORT}\\ \text{STABLE SORT}\\ \text{TAG SORT}\\ \text{STABLE TAG SORT}\\ \text{MERGE}\\ \text{COPY}\end{matrix}\right\}$$

$$\left[\textit{[REFORMAT] [REPLACE] [INSERT] [PAD\_TO] [TRANSLATE] [COMPRESS] [IF]}\right]\ldots$$

Thus, in its simplest form, a command file may consist of a single imperative followed by the END statement.

When entering the command file at your console, you must terminate each unique declaration, massage statement, imperative, and the END statement with a period, but each may span several lines, if you need them, or if you would find the file easier to read by doing so. Also note that you may separate declarations phrases and massage statements with spaces, tabs, form feeds, and new lines.

General Information

In the following chapters, this format indicates that you must specify a range:

$$\left\{\begin{matrix}\text{integer/integer}\\ \text{integer/LAST}\end{matrix}\right\}$$

This range encompasses the record from its first character to its last. We count each character position in a record, even if it contains a blank or null character. The record's first character is numbered 1; each succeeding character's number is one greater, counting in decimal.

The integer you specify to the right of the / must be equal to or greater than the integer you specify to the left. Thus, in an 80-character record, a five-character field may begin at character position 41 and end in character position 45. To specify that field, you'd enter 41/45. To specify that field's first character, you'd enter 41/41. To specify the entire record, you would enter 1/80 or 1/LAST. To specify the last 10 characters, you would enter 71/80 or 71/LAST. The integer/LAST form is particularly useful when you are specifying variable-length fields or records.

Use the following form to single-out one character for processing:

{ integer }
{ 'literal' }

In this form, integer represents the character's corresponding decimal value. For example, the decimal value of an ASCII $ is 36. Therefore, to specify the $ you could enter 36 or '$'. When you use integer to represent a scalar value, as in:

RECORDCOUNT=integer

the largest integer you can specify is 65,535 decimal.

You must always delimit literals with apostrophes or quotation marks. You can also represent a literal by its octal value enclosed in angle brackets < >. All the following are valid ways of representing the ASCII character A.

'A' "A" '<101>' "<101>"

To specify one of the literal delimiters as a literal, delimit it with the alternative delimiter or use the angle bracket notation. For example,

"'" or '"' or "<047>" or "<042>"

End of Chapter

Chapter 4
Command File Declarations

You'll use command file declarations to define:

- one or more input files,
- a single output file,
- any tables you'll want to use,
- the sort or merge keys, and
- one or more work files.

What you specify in your command file's declaration portion depends on what you have specified on the command line and what you want to accomplish.

Note that the input and output file declarations described in this chapter are for AOS files only. We describe how to define INFOS files in Chapter 7.

AOS Input File Declaration

To define one or more AOS input files in your command file, you must define each one in a separate Input File declaration. The order in which you define input files is the order in which the utility processes them. Note that if you name some input files on the command line and others in your command file, the utility will process those named on the command line first.

You may define any AOS input file in an Input File declaration. However, you must use AOS Input File declarations to define AOS input files:

- that contain variable-length records,
- that contain data-sensitive records exceeding 136 characters, or
- that contain data-sensitive records not delimited by nulls, line feeds, carriage returns, or form feeds.

The format of an AOS Input File declaration is:

```
INPUT FILE IS 'name'

                    ( integer { CHARS      }                                                     )
                    (         { CHARACTERS }                                                     )
  , { RECS    } ARE ( { DATA SENS      } [DELIMITED BY 'literal'] UPTO integer { CHARS      }    ) .
    { RECORDS }     ( { DATA SENSITIVE }                                       { CHARACTERS }    )
                    (                                                                            )
                    ( VARIABLE UPTO integer { CHARS      }                                       )
                    (                       { CHARACTERS }                                       )
```

In the above format, integer represents record length in characters. The RECORDS ARE clause is optional only if:

- the input records are fixed-length, or
- the input records are data-sensitive, do not exceed 136 characters, and are delimited by the null, line feed, form feed, or carriage return character.

To specify the length of fixed-length records, use the first form of the RECORDS ARE clause:

```
, { RECS    } ARE integer { CHARS      }
  { RECORDS }             { CHARACTERS }
```

For example,

```
,RECS ARE 90 CHARS
```

If the input file has data-sensitive records, use the second form of the RECORDS ARE clause to specify the length of the longest record:

```
, { RECS    } ARE { DATA SENS      } [DELIMITED BY 'literal'] UPTO integer { CHARS      }
  { RECORDS }     { DATA SENSITIVE }                           { CHARACTERS }
```

For example,

```
,RECS ARE DATA SENS UPTO 95 CHARS
```

You specify an upper bound to the length of data-sensitive records with the phrase:

```
UPTO integer { CHARS      }
             { CHARACTERS }
```

If you know the length of the longest data-sensitive record, specify that. Otherwise, specify an upper bound which will be adequate, yet won't waste too much space. If your data-sensitive records are not delimited by one of the AOS default delimiters (null, carriage return, new line, or form feed) you must define the input file in an AOS Input File declaration, specifying the delimiters in the phrase:

```
DELIMITED BY 'literal'
```

For example,

```
DELIMITED BY '/;'
```

tells the utility to delimit records when it finds a slash or a semicolon. However, note that if you specify your own delimiters, you automatically exclude the AOS defaults. To include some of them as well as your own delimiters, you must specify each one you want. For example, to use a null, a carriage return, and a comma as delimiters, you would specify:

```
DELIMITED BY '<0> <012>,'
```

The utility builds a data-sensitive delimiter table consisting of each character you specify in 'literal'.

If your file has variable-length records you must define it in an AOS Input File declaration, using the clause:

```
, { RECS    } ARE VARIABLE UPTO integer { CHARS      }
  { RECORDS }                           { CHARACTERS }
```

For example,

```
,RECS ARE VARIABLE UPTO 86 CHARS.
```

If you know the length of the longest variable-length record, specify that. Otherwise, specify an upper limit which seems to be adequate, yet is practical from both the system's and your viewpoint. That is, if you think that your largest record will be about 75 characters, you could specify 85 characters to be safe. (Note, however, that if you give an inadequate bound for the record length, the system will return an execution time error.)

The following illustrates three AOS Input File declarations. The first, for FILE_1, specifies fixed-length records 40 characters long. The second, for FILE_2, specifies 132-character data-sensitive records delimited by the #, /, or @ characters. Note that no characters except #, /, and @ appear in the DELIMITED BY phrase. If you separate the delimiters in the command line, (with spaces or commas, for example,) the utility will assume that those separators are also delimiters and will place them in the data-sensitive delimiter table. The third example, for FILE_3, specifies variable-length records, the largest of which is 512 characters.

```
INPUT FILE IS 'FILE_1',
   RECORDS ARE 40 CHARACTERS.
INPUT FILE IS 'FILE_2',
   RECS ARE DATA SENS DELIMITED BY '#/@' UPTO 132 CHARS.
INPUT FILE IS 'FILE_3',
   RECORDS ARE VARIABLE UPTO 512 CHARACTERS.
```

Note that a period terminates each declaration. The indentation and formatting are up to you. For example, it is syntactically correct to define FILE_2 with:

```
INPUT FILE IS 'FILE_2',
   RECS ARE DATA SENS
   DELIMITED BY '#/@'
   UPTO 132 CHARACTERS.
```

AOS Output File Declaration

You'll define one AOS output file (which may or may not exist) in your command file. If the output file does exist, you should specify the global switch /O on the command line. This allows you to sort, merge, or copy an input file into itself, to delete and recreate an existing file which is not an input file, or to direct your output to an AOS-defined file, such as the line printer or a magnetic tape.

If the AOS output file exists and is not also one of the input files, and if you don't specify the global switch /O on the command line, the utility tells you (if you specified /C) that the file exists and asks whether it should be deleted and recreated or not. If the output file does not exist, the utility creates it with the appropriate record format (as described in Chapter 2) for the basic sort output file. The format of an AOS Output File declaration is:

```
OUTPUT FILE IS 'name'.
```

Table Declaration

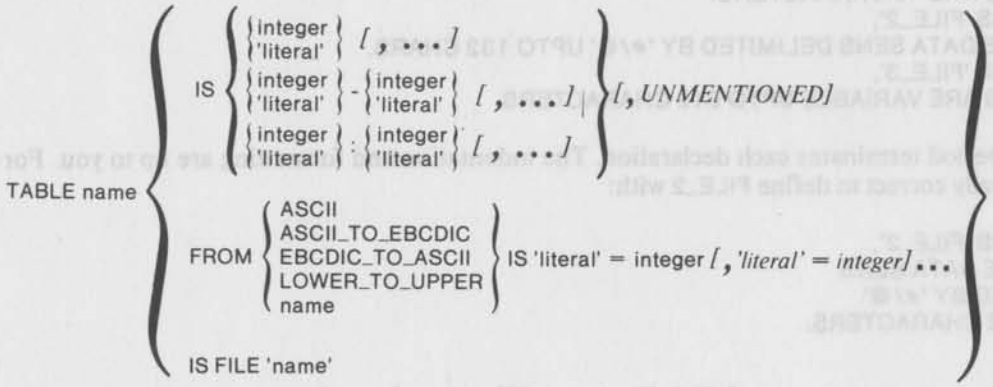
In general, the cases for which you'll define a table are:

- When your records are not recorded in a standard code such as ASCII or EBCDIC.
- When you want to alter the normal collating sequence of your output.
- When you want to ignore one or more characters in a sort or merge key.

If your records are recorded in ASCII and you want them translated to EBCDIC, or vice versa, you need not define a table in your command file. You simply specify the TRANSLATE massage statement, naming an appropriate utility-supplied translation table (as described in Chapter 5).

The utility treats your record's characters as 8-bit bytes (which ASCII and EBCDIC characters are), and collates your output in ascending sequence according to the character's collating value. The utility assumes your records are in ASCII code unless you specify otherwise. Moreover, if you want to alter the collating sequence from ascending to descending, you needn't define a table. You can simply specify the descending sequence in your Key declaration, described in the next section.

While the collating sequence of a character set need not have any correspondence to its recording code, it often does. For example, in ASCII the decimal equivalent of a character's binary representation determines that character's position in the ASCII collating sequence. For example, in ASCII the internal representation of A is 01000001, which translates to 65 decimal, and 9 is 00111001, which translates to 57 decimal. Consequently, in the ascending sequence, a record whose sort_key begins with 9 appears in the output before one whose sort key begins with A. The format of the Table declaration is:



[TABLE name ... .] ...

The name you specify in the TABLE name phrase must consist of only uppercase letters, digits, and the underline. Note that the name is not a literal. Consequently, do not delimit it with ' or "; if you use these delimiters you'll get an error. The name you specify represents a 256-byte table which the utility constructs for you. All tables you define exist only for the duration of the current evocation. You cannot save them.

Each Table declaration form is suited to a particular need. The first form is provided so you can alter the standard ASCII collating sequence. You can use this form to assign your own collating values to one or more ASCII characters. In the phrase:

$$\left\{\begin{matrix}\text{integer}\\ \text{'literal'}\end{matrix}\right\}$$

you can specify a 'literal' consisting of one or more ASCII characters. However, in the phrases:

$$\left\{\begin{matrix}\text{integer}\\ \text{'literal'}\end{matrix}\right\} - \left\{\begin{matrix}\text{integer}\\ \text{'literal'}\end{matrix}\right\} \text{ and } \left\{\begin{matrix}\text{integer}\\ \text{'literal'}\end{matrix}\right\} : \left\{\begin{matrix}\text{integer}\\ \text{'literal'}\end{matrix}\right\}$$

your 'literal' must consist of a single ASCII character. The integer must be in the range 0 to 255 decimal, since it represents an ASCII character's collating value.

Appendix B shows a character set with 128 characters. It also shows each character's decimal equivalent for ASCII code and its hexadecimal equivalent for EBCDIC code. You'll notice that the ASCII collating positions for the illustrated character set are numbered in decimal from zero to 127. Table 4-1 is a simplification of the table in Appendix B. All the examples in this chapter will use characters shown in Table 4-1.

Table 4-1. Simplified ASCII Collating Table

Collating Position Number (Decimal)	ASCII Character
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z

When you use the Table declaration form:

TABLE name IS { integer / 'literal' } [, ...]

the order in which you specify characters determines the collating value the utility assigns to them. For example, you can assign the letter J collating value 0 and the digit 5 collating value 1 by specifying any of the following:

```
TABLE TEMP_1 IS 'J', '5'.
TABLE TEMP_1 IS 'J', 53.
TABLE TEMP_1 IS 74, 53.
TABLE TEMP_1 IS 74, '5'.
```

As you can see, it doesn't matter whether you specify a literal or an integer. What matters is the order in which you specify them. To assign the letters A, B, and C the collating value 0 and X, Y, and Z the collating value 1 you could specify:

```
TABLE TEMP_1 IS 'ABC', 'XYZ'.
```

To assign digits 0 through 9 the collating values 0 through 9, you could specify:

```
TABLE TEMP_1 IS 48, 49, 50, 51, 52, 53, 54, 55, 56, 57.
```

An easier way is to use:

TABLE name IS { integer / 'literal' } - { integer / 'literal' }

Thus, to assign the digits 0 through 9 the collating values 0 through 9, you could specify either of the following.

```
TABLE TEMP_1 IS 48-57.
TABLE TEMP_1 IS '0'-'9'.
```

Using this form, you assign a *string* of characters a *range* of values. To assign a *string* of characters a *single* collating value you'll use the form:

TABLE name IS { integer / 'literal' } : { integer / 'literal' }

For example, if you want to assign all the digits the collating value 0 you could specify:

```
TABLE TEMP_1 IS '0':'9'.
```

You can use all three clauses in a single Table declaration when you use this form of the TABLE name phrase. For example, to assign the character $ the collating value 0, the digits 0 through 9 the collating values 1 through 10, and all the uppercase ASCII letters the single collating value 11, you could specify:

```
TABLE TEMP_1 IS '$', '0'-'9', 'A':'Z'.
```

In all these examples, the utility automatically assigns each ASCII character its normal sequential position, *beginning in the next available collating position in your table,* unless you explicitly specify otherwise.

Use the UNMENTIONED phrase to assign a particular collating value to all the characters you do not mention in your table specification. For example, let's assume you are sorting a file of numeric keys. Since you are only concerned with digits, you might specify:

```
TABLE TEMP_1 IS '0'-'9', UNMENTIONED.
```

In this case, you assigned 0 the collating value 0, 1 the collating value 1, etc., and 9 the collating value 9. You assigned all other ASCII characters the collating value 10. If you specify:

```
TABLE TEMP_1 IS UNMENTIONED, '0'-'9'.
```

you assign all unmentioned ASCII characters the collating value 0. You assign the digit 0 the collating value 1, 1 the value 2, etc. You may specify the UNMENTIONED clause anywhere in your phrase. For example:

```
TABLE TEMP_1 IS 'A'-'Z', UNMENTIONED, '0':'9'.
```

In this case, you assigned the uppercase ASCII letters the collating values 0 through 25 and all the digits 0 through 9 have the single collating value 27. You assigned all unmentioned characters the collating value 26.

NOTE: When you use this form of the Table declaration you may specify each ASCII character and the UNMENTIONED phrase only once. That is, while you can assign a single collating value to more than one characer, you cannot assign more than one value to any single character.

The second form of the Table declaration:

```
                  ⎧ ASCII           ⎫
                  ⎪ ASCII_TO_EBCDIC ⎪
TABLE name FROM   ⎨ EBCDIC_TO_ASCII ⎬ IS 'literal' = integer  [ , 'literal' = integer] ... .
                  ⎪ LOWER_TO_UPPER  ⎪
                  ⎩ name            ⎭
```

is most useful when you intend to specify the COMPRESS massage statement described in Chapter 5. In this form, you alter the collating sequence of a previously defined table. The utility provides four predefined tables: ASCII, ASCII_TO_EBCDIC, EBCDIC_TO_ASCII, and LOWER_TO_UPPER. (When you use the LOWER_TO_UPPER table you assign each lowercase ASCII character the collating value of its corresponding uppercase letter.) Also note that during a single evocation of the utility you can alter the collating sequence of a table you defined in a preceding Table declaration.

When you use this form, the utility builds a 256-byte table, gives it the name you specify, and assigns the collating value you specify as integer to the character you specify as 'literal'. For example:

```
TABLE TEMP_2 FROM ASCII IS 'A'=48, 'B'=49, 'C'=50.
```

When the utility builds TEMP_2 it assigns both the uppercase letter A and the digit 0 the collating value 48. It assigns letter B and the digit 1 the collating value 49, and C and 2 the value 50. The utility assigns all other ASCII characters their normal ASCII collating numbers.

Note, too, that 'literal' can be a string of characters; the utility assigns all characters in the string the collating value you specify as integer. For example:

```
TABLE TEMP_2 FROM ASCII IS 'ABC'=48, 'XYZ'=49.
```

In this case the utility assigns the letters A, B, and C and the digit 0 the collating value 48 and X, Y, and Z and the digit 1 the collating value 49. The utility assigns the remaining ASCII characters their normal collating values.

NOTE: There is an important distinction between this form of the Table declaration and the first form we described: In this form the table is effectively mapped into itself, except for those characters you specify. In the first form the characters you do not specify remain in their normal collating sequence, which begins in the collating position immediately after the last character you specify.

In the third form of the Table declaration:

[TABLE name ... .] ...

you specify the name of a table which you want the utility to build table which you want the utility to build and the name of a *file* which contains the collating values of the characters. You must specify an AOS pathname (delimited by ' or ") to the existing file, which must contain 256 bytes. We provide this form of the Table declaration primarily for use by programmers who are writing a language interface to the utility. We do not recommend that you use it if you are using the utility to sort, merge, or copy files.

Key Declaration

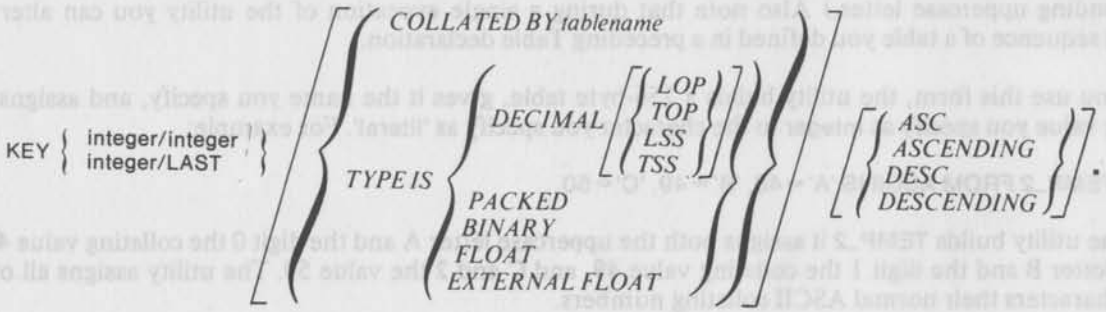
Whenever you invoke the utility to specify one of the sorts or the merge operation, you may specify the key on which you want the sort or merge to take place. You'll also specify a key when copying to an INFOS output file. (See Chapter 7.)

You may specify a primary key and one or more secondary keys. For example, suppose you want to sort an input file of personnel records currently ordered by employee social security number. You want your output file to be ordered by employee name: last name first, first name, middle initial. Your first Key declaration, therefore, describes the primary key: employee last name. Your second declaration describes the second most important key: (first name), and your next declaration describes the least important key (middle initial).

The order in which you specify keys is the order in which the utility uses them to resolve conflicts. For example, if you have employees named Jane M Smith, John Q Smith, James T Smith, and Jane B Smith and you specify the primary and secondary keys mentioned above, the utility orders your output file as follows:

SMITH JAMES T - SMITH JANE B - SMITH JANE M - SMITH JOHN Q

The format of the Key declaration is:



[KEY ... .]...

The only required phrase in a Key declaration is the *location* phrase.

{ integer/integer | integer/LAST }

A key's location corresponds to some field in the record after the utility completes the input massaging you specify. For the utility's purposes, a record's first *character position* is numbered one, the next two, etc. in a monotonically increasing sequence to the last character in the record. Each character position contains a character; for example, the space (blank), the binary zero (null), or a printing character (a letter, number, or special symbol like $ or @).

When specifying a key's location you *must* count each character position up to the first character of the key field. For example, if you have records in the format below (the symbol □ represents the space character):

```
    5    10   15    20    25    30   35    40    45
    |    |    |     |     |     |    |     |     |
□□□□159-28-1936□□□□JAMES□□□□□T□SMITH□□□□□□□□□□ ...
```

you'd specify the locations of the primary and secondary keys as:

```
KEY 32/46.
KEY 20/29.
KEY 30/30.
```

Note that the second integer must always be equal to or greater than the first integer. If the record's last field contains employee last name, you could use the form:

```
KEY 32/LAST.
```

You'll use the TYPE IS phrase to specify key type. The default character set is ASCII and the default key type is *character,* indicating that the keys consist of one or more ASCII characters. The character key type corresponds, roughly, to the COBOL alphanumeric data type described in the *ECLIPSE-LINE COBOL Reference Manual* (093-000180). In fact, we provide the TYPE IS phrase primarily for those interfacing to the utility from a COBOL program.* If you intend to use the TYPE IS phrase you must use the integer/integer form of the location phrase (rather than the integer/LAST form), because variable-length special data types aren't allowed.

Normally, the utility writes your output in ascending sequence, based on the internal character code described in the Table declaration section. You can alter the normal collating sequence in two ways. You can simply specify that you want your output in descending sequence. For example:

```
KEY 32/46 DESCENDING.
```

Or you can define a table (in a Table declaration) and specify that table's name in the COLLATED BY phrase. For example:

```
KEY 32/46 COLLATED BY TEMP_1.
```

Work File Declaration

Normally, the utility builds two work files in the current working directory, which it uses for intermediate storage and deletes when it completes execution. We provide the optional Work File declaration so that if you have direct access devices with a high transfer rate or short access times (such as fixed-head disks), you can define portions or one or more of them as work files, thus speeding up the operation. The format of the Work File declaration is:

```
WORK FILE IS 'filename'.
```

The 'filename' you specify must be an AOS pathname, delimited by ' or ''. If the file does not exist, the utility creates one. If the file does exist, the utility deletes it and recreates it. Note that the utility does not automatically delete work files which you define when execution is completed. If you define more than one work file, the utility attempts to use them alternately (i.e., in a round-robin manner).

End of Chapter

---

* Note, however, that those of you who are not familiar with COBOL may also use decimal, binary, floating point or external floating point keys. See the "Commercial Instruction Addressing Set" section of Chapter 2 in the *ECLIPSE C-Series Computers* manual (number 015-000047) for further details.

Chapter 5
Command File Directive Group

You will use the directive group portion of your command file to specify massage statements and an imperative. Your specifications will have the following format:

- First, specify one or more massage statements for the *input* records (if desired);
- Second, specify an imperative
- Third, specify one or more massage statements for the *output* records (if desired).

That is, you must specify massage statements for the input records *before* specifying an imperative, but, to massage output records, you specify one or more massage statements *after* specifying an imperative. This means that you may massage both input and output files in a single evocation and that you may specify any number of each kind of massage statement (input/output) in any order. Be aware, however, that the order in which you specify the statements is the order in which the utility executes the statements.

Massage Statements

REFORMAT

The REFORMAT statement lets you rearrange a record's fields, delete fields from a record, and/or repeat a record's fields. The REFORMAT statement's format is:

REFORMAT { integer/integer | integer/LAST } [, ...] .

Suppose you have a record in the following format (the □ represents a blank space):

```
          5    10   15   20   25   30   35   40   45   50  55   60
          |    |    |    |    |    |    |    |    |    |   |    |
1A9□28□19B6□□□DEPAF□ENTERPRISES□INC.□□□MAIN□ST.□U.S.A.□□□□□□
```

To move the field 1A9□28□19B6 to the end of the record, you would specify:

REFORMAT 12/LAST, 1/11.

To delete the field ENTERPRISES you'd specify:

REFORMAT 1/20, 32/LAST.

To repeat the field ENTERPRISES you'd specify:

REFORMAT 1/20, 21/31, 21/32, 32/LAST.

Note that if your records are data-sensitive and you use the integer/LAST form, LAST represents the data-sensitive delimiter character position. Consequently, you cannot insert anything after that character. You can, however, replace the delimiter with a REPLACE statement, reformat the record with a REFORMAT statement, and insert a new delimiter with an INSERT statement.

Since the utility massages the input records *before* it executes the imperative, you should take care that:

- The key you specify in a Key declaration corresponds to the appropriate field after the input massage is done;
- The massaged record is the length that you specify in the INFOS Input File declaration RECORD IS phrase. (See Chapter 7.)

This is especially important to Key declarations, INFOS output qualifiers (see Chapter 7), and subsequent massage statements. If you ignore these cautions you may get the RECORD TOO SHORT FOR RANGE or the RECORD TOO SHORT FOR KEY error message.

You may specify more than one REFORMAT statement. Therefore, to delete the ENTERPRISES field you could also specify:

```
REFORMAT 1/20.
REFORMAT 32/LAST.
```

REPLACE

You make global changes to your records with the REPLACE statement. Its format is:

REPLACE [{ *ALL* / *ANY* }] 'literal' IN { integer/integer / integer/LAST } WITH 'literal'.

Suppose that you want to replace the blank spaces in the field 1A9□28□19B6 (in the previous example record) with the slash (/). You could specify:

```
REPLACE '<040>' IN 1/11 WITH '/'.
REPLACE '<040>' IN 1/11 WITH '/'.
```

When you do not specify ANY or ALL, the utility scans the field from left to right, replacing the first blank space it encounters with the /. That is, the utility replaces the first occurrence of the first 'literal' with the second 'literal' then the scan stops. The utility makes one scan for each REPLACE statement you specify.

In the preceding example, you've simply replaced one character with another. However, you may replace any number of characters with any other number of characters. For example:

```
REPLACE '<040>' IN 1/11 WITH '***'.
REPLACE '<040>' IN 1/11 WITH '***'.
```

replaces each blank space in 1A9□28□19B6 with three asterisks, as follows:

```
1A9***28***19B6
```

This, of course, adds four characters to the final record, making its length 64 characters rather than the original 60. Conversely, you could specify:

```
REPLACE '1A9 28 19B6' IN 1/11 WITH '1A92819B6'.
```

which removes the blank spaces, reducing the record to 58 characters.

However, the ANY phrase replaces the blanks more easily. When you specify the ANY phrase, the utility scans the entire field you specify in the *location* phrase from left to right, replacing each occurrence of the first 'literal' with the second 'literal'. Thus:

```
REPLACE ANY '<040>' IN 1/11 WITH '/'.
```

accomplishes in one statement the task previously carried out with two statements. Also:

```
REPLACE ANY '<040>' IN 1/11 WITH '***'.
```

is a simpler way of doing:

```
REPLACE '<040>' IN 1/11 WITH '***'.
REPLACE '<040>' IN 1/11 WITH '***'.
```

The ALL phrase has a significantly different effect. When you specify ALL, the utility scans the field from left to right, replacing the first occurrence of the first 'literal' with the second 'literal', and the scan stops. The utility then *rescans* the field (again from left to right), replacing the first occurrence of the first 'literal' with the second 'literal' and the scan stops again. The utility continues to scan the field until there are no more occurrences of the first 'literal'. Thus, using the same example record as before, if you specify:

```
REPLACE ALL '<040><040>' IN 1/LAST WITH '<040>'.
```

the results are;

```
    5    10    15    20    25    30    35    40    45    50
    |     |     |     |     |     |     |     |     |     |
1A9□28□19B6□DEPAF□ENTERPRISES□INC.□MAIN□ST.□U.S.A.□
```

In contrast, if you specify:

```
REPLACE ANY '<040><040>' IN 1/LAST WITH '<040>'.
```

the results are:

```
    5    10    15    20    25    30    35    40    45    50    55
    |     |     |     |     |     |     |     |     |     |     |
1A9□28□19B6□□DEPAF□ENTERPRISES□INC.□□MAIN□ST.□U.S.A.□□□
```

The first 'literal' you specify can never be an empty string. That is, you cannot specify '' as the first 'literal'. You can, however, specify an empty string as the second 'literal'. Thus, to remove the field INC.□□□ from the original example record you could specify either:

```
REPLACE 'INC.  ' IN 33/39 WITH ''.
```

or

```
REPLACE ANY 'INC.  ' IN 33/39 WITH ''.
```

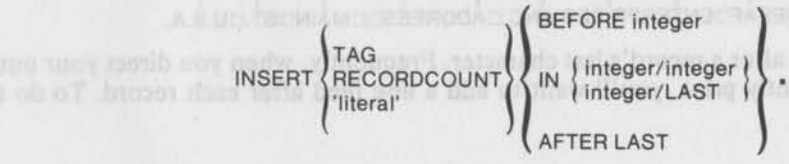
Both REPLACE statements result in the record:

```
    5    10    15    20    25    30    35    40    45    50
    |     |     |     |     |     |     |     |     |     |
1A9□28□19B6□□□DEPAF□ENTERPRISES□MAIN□ST.□U.S.A.□□□□□□
```

As with the REFORMAT statement, we caution you to be careful when using the REPLACE statement on input records.

INSERT

You use the INSERT statement to add information anywhere in a record. Its format is:



If you specify TAG the utility generates a six-character binary tag. Its first two characters are the file's number and its last four characters are the record's logical disk address. Note that:

- Inserting a tag is meaningful only for disk input files, and then, only if you do not specify either Tag Sort imperative.
- You cannot insert a tag for magnetic tape input files, because the last four characters of the tag represent the record's logical disk address.

- When you define the input files, either on the command line or in your command file, the first file number is 0, the second is 1, the third 2, etc.
- We caution you against inserting tags into an input file with data-sensitive records. A tag might contain a data-sensitive delimiter, NULL for example, in which case the utility would truncate your output records.

If you specify RECORDCOUNT, the utility generates an eight-character ASCII decimal number that represents the record's ordinal position. On input, the utility numbers the records sequentially beginning with one for the first record on the first input file. It gives the first record on the second input file a record number one greater than the last record on the first input file. On output, the utility numbers the records sequentially as it writes them to the output file.

If you use the 'literal' form, your 'literal' may be a string of one or more characters or an empty string. The utility inserts the information you specify as 'literal' at the position you indicate in the *location* phrase. For example, if you want to insert NAME and ADDRESS into the example record we've been using, you would specify:

```
INSERT 'NAME' BEFORE 13.
INSERT 'ADDRESS' BEFORE 42.
```

After the first insertion the record's contents are:

```
          5    10    15    20    25    30    35    40    45    50    55    60
          |     |     |     |     |     |     |     |     |     |     |     |
1A9□28□19B6□NAME□□DEPAF□ENTERPRISES□INC.□□□MAIN□ST.□U.S.A.□□□□□□
```

After the second insert the record's contents are:

```
          5    10    15    20    25    30    35    40    45    50    55    60  65    70
          |     |     |     |     |     |     |     |     |     |     |     |   |     |
1A9□28□19B6□NAME□□DEPAF□ENTERPRISES□INC.□ADDRESS□□MAIN□ST.□U.S.A.□□□□□□
```

There is a simple difference between the BEFORE form and the IN form: if you use the IN form, the utility deletes the characters you specify in your *location* phrase and inserts the information you specify (TAG, RECORDCOUNT, or 'literal') in their place. If you use the BEFORE form, the utility does not delete anything from the record. For example, to strip off 1A9□28□19B6 and the trailing blanks from the record shown immediately above, you'd specify:

```
INSERT '' IN 1/12.
INSERT '' IN 54/LAST.
```

The first of these insertions results in:

```
          5    10    15    20    25    30    35    40    45    50    55
          |     |     |     |     |     |     |     |     |     |     |
NAME□□DEPAF□ENTERPRISES□INC.□ADDRESS□□MAIN□ST.□U.S.A.□□□□□□
```

The second insertion produces:

```
          5    10    15    20    25    30    35    40    45    50
          |     |     |     |     |     |     |     |     |     |
NAME□□DEPAF□ENTERPRISES□INC.□ADDRESS□□MAIN□ST.□U.S.A.
```

You can also insert information after a record's last character. Frequently, when you direct your output to a disk file which you'll subsequently print, you'll want to add a line feed after each record. To do this you specify:

```
INSERT '<012>' AFTER LAST.
```

Again, remember that we caution you to be careful when massaging input records.

PAD

You use the PAD statement to convert variable-length records to fixed-length records. Its format is:

```
PAD TO integer {CHARS      } WITH {'literal'} .
               {CHARACTERS }      {integer  }
```

The length you specify as integer must be at least as long as the longest variable-length record in the file. If the utility encounters a record longer than the length you specify, you'll get a runtime error. You may pad records by specifying a single character 'literal' or by specifying a decimal integer that represents a character's collating value, (shown in Appendix B). Therefore, if you specify an integer it must be in the range 0 to 255 decimal. For example, to pad a variable-length record to 132 characters using the blank space you could specify any of the following:

```
PAD TO 132 CHARS WITH '<040>'.
PAD TO 132 CHARS WITH ' '.
PAD TO 132 CHARS WITH 32.
```

Note that padding variable-length records to a fixed length does not alter the record format characteristic of the output file. The utility automatically creates the output file with the AOS variable-length record characteristic. Also, padding begins after the last character in a record. In data-sensitive records, this last character is one of the data-sensitive delimiters. Therefore, if you have data-sensitive records and you want all your output records to have the same length, you should replace this delimiter before you PAD, then insert a new delimiter AFTER LAST.

TRANSLATE

You use the TRANSLATE statement to translate your records from one recording code set to another, or to convert lowercase ASCII letters to their uppercase equivalents. Remember: the utility assumes that your records are in ASCII code and collates your output in ascending ASCII sequence. The format of the TRANSLATE statement is:

```
TRANSLATE {integer/integer} USING {ASCII_TO_EBCDIC  }
          {integer/LAST   }       {EBCDIC_TO_ASCII  }
                                  {LOWER_TO_UPPER   }
                                  {tablename        } .
```

To translate from ASCII to EBCDIC, from EBCDIC to ASCII, or from lowercase to uppercase ASCII you do not need to specify a Table declaration (described in Chapter 4). You simply specify the name of a utility-supplied translation table. For example:

To tranlsate from ASCII to EBCDIC, from EBCDIC to ASCII, or from lowercase to uppercase ASCII you do not need to specify a Table declaration (described in Chapter 4). You simply specify the name of a utility-supplied translation table. For example:

```
TRANSLATE 1/LAST USING EBCDIC_TO_ASCII.
```

To translate to or from a character code that is neither ASCII nor EBCDIC, you must create a table, using a form of the Table declaration (described in Chapter 4), and specify that table's name in a TRANSLATE statement.

In most cases, you should perform your translation on input. For example, if you are sorting a file of EBCDIC records and you intend to output them in ASCII, you should specify the TRANSLATE statement before specifying the SORT imperative. This ensures that the utility will sort the records according to the ASCII equivalent of the sort key and write the output in the proper sequence. If you specify the SORT imperative first, the utility sorts the records according to the EBCDIC equivalent of the sort key, then translates them. Though the output will be ASCII, it may not be in the proper ASCII sequence.

COMPRESS

You'll use the COMPRESS statement to remove unwanted characters in a character string. When you specify the COMPRESS statement, the utility removes characters whose collating value is zero. Use the Table declaration (described in Chapter 4) to build a table in which you assign the collating value zero to unwanted characters. You specify that table's name in a COMPRESS statement. The format of a COMPRESS statement is:

COMPRESS { integer/integer / integer/LAST } { LEFT [{ integer / 'literal' } FILLED] / RIGHT [{ integer / 'literal' } FILLED] / VARIABLE } USING tablename .

You use the *location* phrase integer/integer or integer/LAST to specify the string you want compressed. Note that you can compress the entire record by specifying 1/LAST.

There are a number of ways to control the length and contents of the compressed string. If you simply specify LEFT or RIGHT the utility justifies (to either the left or right) the compressed string in the original field. If you specify LEFT, it appends nulls to the end of the string; if you specify RIGHT, it inserts them before the string's first character. In either case, the compressed string will be exactly as long as the original string. Note, however, that null insertion in data-sensitive records is dangerous because null is one of the system's default delimiters.

If you don't want nulls to appear in the string, use the form:

[{ integer / 'literal' } FILLED]

You can specify a single character 'literal' or its decimal equivalent. For example, you could specify A or 65.

If you use the VARIABLE option, the utility compresses the string left-justified in the field and truncates the field to the length of the justified string. That is, the record's next field begins in the character position immediately following the last character of the compressed string.

Assume that you want to sort a file of personnel records, using employee surname as the sort key. This field occupies character positions 11 through 25 in the records. Now let's say you have employees named ODonnel, OMalley, and O'Leary. You'd normally expect these names to be in alphabetical order in the output. But, because the apostrophe has a lower collating value than any letter, O'Leary will appear in the output before any other name beginning with O. To overcome this problem, define a table in which you assign the collating value zero to the apostrophe, then specify that table's name in a COMPRESS statement. For example:

```
TABLE NAMES FROM ASCII IS " ' "=0.
.
.
.
COMPRESS 11/25 LEFT '<040>' FILLED USING NAMES.
```

This COMPRESS statement compresses the string O'Leary to:

```
11          25
|           |
OLEARY□□□□□□□□□□
```

If you specify:

```
COMPRESS 11/25 RIGHT '*' FILLED USING NAMES.
```

O'Leary becomes:

```
11          25
*********OLEARY
```

If you specify:

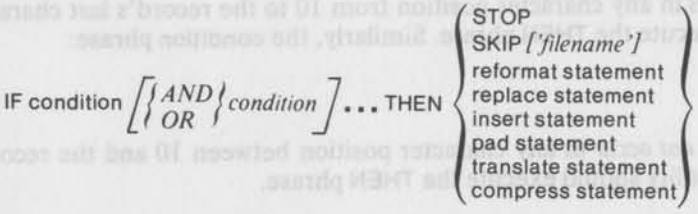
```
COMPRESS 11/25 VARIABLE USING NAMES.
```

it becomes:

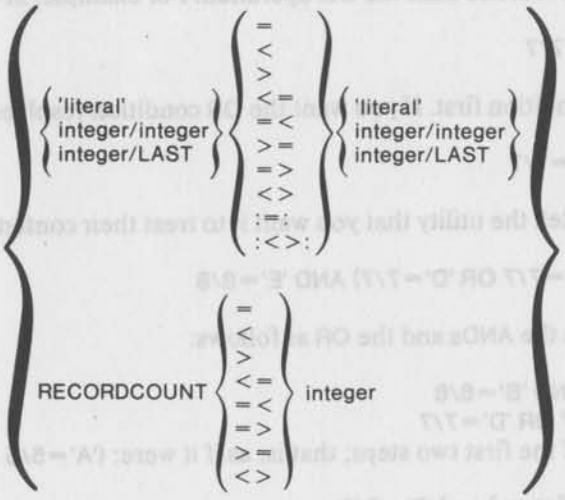
```
11      16
OLEARY
```

IF

You use the IF statement to establish conditions which control the utility's output, or which initiate massaging operations. The IF statement's format is:



The format of the *condition* phrase is:



The condition phrase works like an equation. If the equation is true, the utility executes the THEN phrase; if it's false, the utility doesn't execute the THEN phrase. You should specify a 'literal' on one side of the condition phrase *operator* and a range on the other side, or specify ranges on both sides.

In general, you should specify the same length 'literal' or range on each side of the operator. If you specify different length 'literals' or ranges, the utility pads the shorter to the length of the longer, using the blank character (<040>) as the padding character. Two operators, however, let you do *floating* compares: :=: and :<>:. If you use either of these, the utility does not pad the shorter 'literal' or range.

The IF statement uses the following operators:

Operator	Means
=	is equal to
<	is less than
>	is greater than
=< or <=	is less than or equal to
=> or >=	is greater than or equal to
<>	is not equal to
:=:	occurs in (used for floating compares)
:<>:	does not occur in (used for floating compares)

The condition phrase:

```
'A' :=: 10/LAST
```

means that *if an A occurs* in any character position from 10 to the record's last character, *the condition is true* and the utility should execute the THEN phrase. Similarly, the condition phrase:

```
'A' :<>: 10/LAST
```

means that *if an A does not occur* in any character position between 10 and the record's last character, *the condition is true* and the utility should execute the THEN phrase.

If you specify RECORDCOUNT in a condition phrase, you must specify it to the left of the operator. If you use this form, the integer you specify to the right of the operator cannot exceed 65,535 decimal.

You can logically combine condition phrases using ANDs and ORs, and parenthesizing them. In general, the AND operation has greater precedence than the OR operation. For example, in the series:

```
'A'=5/5 OR 'B'=6/6 AND 'C'=7/7
```

the utility resolves the AND condition first. If you want the OR condition resolved first, specify:

```
('A'=5/5 OR 'B'=6/6) AND 'C'=7/7
```

Your parentheses may simply tell the utility that you want it to treat their contents as an entity. For example:

```
'A'=5/5 AND 'B'=6/6 AND ('C'=7/7 OR 'D'=7/7) AND 'E'=8/8
```

In this case, the utility resolves the ANDs and the OR as follows:

- First, it performs 'A'=5/5 AND 'B'=6/6
- Second, it performs 'C'=7/7 OR 'D'=7/7
- Third, it ANDs the results of the first two steps; that is, as if it were: ('A'=5/5 AND 'B'=6/6) AND ('C'=7/7 OR 'D'=7/7).
- Fourth, it ANDs the result of step 3 and 'E'=8/8.

The STOP option tells the utility that it has reached the *end of the input* you want processed. If you specify RECORDCOUNT, the utility does not write the last input record to the output file. For example, if you simply want to copy the first 500 records of an input file you could specify:

```
IF RECORDCOUNT > 500 THEN STOP.
```

When the utility encounters input record number 501, it stops and does not write that record to your output file. Suppose you have a file of records sorted by date, with a six character date field in character positions 11 through 16. The dates are in the form 770101 for 1977, January first. To copy records for January, February, and March, you could specify:

```
IF 11/16 > '770331' THEN STOP.
```

Then, when the utility encounters an input record whose date field begins with 7704 or greater, it stops and does not write that record to your output file. Note that the date is specified as a 'literal'.

You decide whether to specify an IF statement before or after the imperative based on which THEN phrase option you choose. You'll use the STOP option primarily when you are massaging input. You should not use it on output because it will probably not produce meaningful results. You may specify any number of IF statements, as long as you conform to the STOP convention.

The SKIP option allows you to ignore input (or output) records, thereby keeping them from the output file. You can also write the ignored records to a different file, which we call a *skip file*. For example, let's say you are sorting a transaction file by department number. If you don't want to include the transactions for department A101 in the sorted output, you can specify:

```
IF 1/4='A101' THEN SKIP.
```

Then, the utility will not pass any records whose first four characters are A101 to the next internal operation. If you want to skip these input records and also save them for processing later, you can specify:

```
IF 1/4='A101' THEN SKIP 'REJECTS'.
```

As before, the utility won't pass any input record whose first four characters are A101. But, the utility will create a file named REJECTS and write the skipped records to it. You must specify an AOS pathname for the file; the file must not exist when you invoke the utility. You may also specify the same skip file in more than one IF statement (even though the utility will create only one skip file with that name), or you may name different skip files in different IF statements.

Your THEN phrase can contain any of the previously described massage statements. Note, however, that you cannot use an IF statement as a THEN phrase, because, you can't nest IF statements. Therefore, you can do any of the following:

```
IF 'ABC' =10/12 THEN REFORMAT 1/9, 13/LAST, 10/12.
IF 1/3='XYZ' THEN REPLACE 'TIME' IN 26/29 WITH 'DATE'.
IF 81/81='<000>' THEN INSERT '<021>' IN 81/81.
IF 1/1='A' THEN PAD TO 132 CHARACTERS WITH '<040>'.
IF 1/1='<077>' THEN TRANSLATE 1/LAST USING ASCII_TO_EBCDIC.
IF 12/14 <>'999' THEN COMPRESS 16/35 VARIABLE USING TEMP_1.
```

Imperatives

Imperatives define the operation you want executed. Each command file must contain *exactly one* imperative. The imperatives are:

```
SORT.
STABLE SORT.
TAG SORT.
STABLE TAG SORT.
MERGE.
COPY.
```

Sort and Stable Sort

These imperatives both sort the input records and order them according to the sort key you specify. A standard Sort is faster than a Stable Sort; however, the essential difference between them is how they treat duplicate sort keys. When a utility performing a Stable Sort encounters two or more sort keys which are exactly equal in value, it writes the output records in the exact order in which it encountered the duplicate keys. However, if you use a standard Sort, the output is in sorted order; but, records whose sort keys are exactly equal may or may not be in the order in which the utility encountered them.

The utility writes the sorted output in ascending ASCII sequence based on the value of the sort key, unless you specify differently in your Key declaration.

Tag Sort and Stable Tag Sort

The essential difference between a Tag Sort and a Stable Tag Sort is the same as between a standard Sort and a Stable Sort. The difference between a Tag Sort and a non-tag Sort concerns the amount of data the utility carries throughout the operation. A utility performing a non-tag sort carries the entire input record in all phases of its operation. However a utility performing a Tag Sort extracts the input record's sort key and logical disk address, combining them into a tag which it carries during the sorting and merging phases. During its output phase, the utility retrieves the entire record and writes it to the output file. The utility writes the sorted output in ascending ASCII order, unless you specify otherwise in your Key declaration.

Changes caused by input massage are not reflected in the output records of a Tag Sort. However, if you massage the key field on which the Tag Sort takes place, that massage will affect the order of the output records, but not their content. Note that massaged records written to a skip file will reflect the results of any massage statements. You may use both the SKIP and STOP options of the IF statement when doing a Tag Sort.

You should choose a Tag Sort over a non-tag Sort when your disk space (for work files) is limited, or when you are sorting a large number of large input records. Tag Sorts are especially attractive when the sum of the Key sizes is much smaller than the sum of the record sizes and the file is very large.

Merge

A Merge merges a minimum of two input files into a single output file. The input files must have been sorted using the key on which you base the merge. The utility writes the merged records to the output file based on the value of the merge key you specify.

Copy

A Copy writes the records in one or more input files to a single output file in the order in which it encounters them in the input files. If the output file is an AOS file, you need not specify a key since the utility neither sorts nor merges the records. If the output file is an INFOS file, however, you must specify the INFOS key which you want the utility to write to the output file's index. (See Chapter 7.) You do this with a Key declaration, described in Chapter 4. You do not enable either sorting or merging by specifying an INFOS key. The utility writes the input records to the database in the sequence in which it encounters them in the input files. It extracts the INFOS key from each input record and writes it to the file's index.

End of Chapter

Chapter 6
Summary Examples

Suppose you have a disk file containing 80-character records which you want to sort sending the output to the line printer. Specify:

```
SORT/O INTO @LPT FROM INFILE
```

The utility treats each 80-character input record as an 80-byte sort key. It sorts all the input records and writes the output in ascending ASCII sequence. If the last 10 characters of the input records contain a key on which you want to base the sort, you might specify:

```
FILTER/C/O INTO @LPT
INPUT FILE IS 'INFILE',
   RECORDS ARE 80 CHARACTERS.
KEY 71/LAST.
SORT.
END.
```

Suppose the records in this file are in EBCDIC. You want to sort them using the key field and, since your output is to the line printer, you'll translate the records to ASCII. To do this, you specify:

```
FILTER/C/O INTO @LPT
INPUT FILE IS 'INFILE',
   RECORDS ARE 80 CHARACTERS.
KEY 71/LAST.
TRANSLATE 1/LAST USING EBCDIC_TO_ASCII.
SORT.
END.
```

Note that there was no need to define a table in this example. You simply specified the TRANSLATE statement and used the utility-supplied table. Note, too, that you performed the translation prior to the sort.

Suppose you have two transaction files, TR_1 and TR_2 whose records are variable-length and up to 132 characters long. The key field on which you want to sort is in character positions 21 through 35. You want to move the key field to the beginning of the output records, and to pad the output records so they will all be 132 characters long. And you want to name the output file MASTER. Specify:

```
FILTER/C INTO MASTER
INPUT FILE IS 'TR_1',
   RECORDS ARE VARIABLE UPTO 132 CHARACTERS.
INPUT FILE IS 'TR_2',
   RECORDS ARE VARIABLE UPTO 132 CHARACTERS.
KEY 21/35.
SORT.
REFORMAT 21/35, 1/20, 36/LAST.
PAD TO 132 CHARACTERS WITH '*'.
END.
```

Here, the utility sorts the records before reformatting and padding them. Consequently, you specify a key based on its position in the *input* record. If you had specified the reformatting first, then the sort, you would have to specify the first 15 characters as the key field, as the example below illustrates.

```
FILTER/C INTO MASTER
INPUT FILE IS 'TR_1',
   RECORDS ARE VARIABLE UPTO 132 CHARACTERS.
INPUT FILE IS 'TR_2',
   RECORDS ARE VARIABLE UPTO 132 CHARACTERS.
KEY 1/15.
REFORMAT 21/35, 1/20, 36/LAST.
SORT.
PAD TO 132 CHARACTERS WITH '*'.
END.
```

In both of these examples the utility creates the output file, MASTER, with the AOS variable-length record characteristic.

Suppose you have a file, TEMP_1, containing transaction records for three different departments whose codes are A12, SO3, and W59. The records are variable up to 60 characters, with the first three character positions containing the department code. You want to create subsets of these records by department code and to reformat the records for department A12, moving the department code to the end of the record. Specify:

```
FILTER/C
INPUT FILE IS 'TEMP_1',
   RECORDS ARE VARIABLE UPTO 60 CHARACTERS.
OUTPUT FILE IS 'A12'.
IF 1/3='SO3' THEN SKIP 'SO3'.
IF 1/3='W59' THEN SKIP 'W59'.
COPY.
REFORMAT 4/LAST, 1/3.
END.
```

Since you wanted to reformat the records for department A12, you directed those records to the output file, saving yourself an IF statement. You might also have specified a third IF statement in the format:

```
IF 1/3 = 'A12' THEN REFORMAT 4/LAST, 1/3.
```

In the above example the utility performs the two specified filtering operations before the copy, writing the records to file SO3 or W59 depending on the filter conditions you specified. However, it writes the records for department A12 to the output file, after reformatting them as you specified.

Now let's say you want to sort the records for department SO3. The keys you'll use are:

- Item number (in character positions 4 through 8).
- Dollar amount (in character positions 9 through 14).
- Style (in character positions 15 through 17).
- Color (in character position 18).

The input records are variable-length up to 60 characters. You want to pad them to 95 characters, inserting a line feed in position 96. You also want the largest dollar amount for each item to appear first in the output. Specify:

```
FILTER/C/O INTO SO3
INPUT FILE IS 'SO3',
   RECORDS ARE VARIABLE UPTO 60 CHARACTERS.
KEY 4/8.
KEY 9/14 DESCENDING.
KEY 15/17.
KEY 18/18.
SORT.
PAD TO 95 CHARACTERS WITH '<040>'.
INSERT '<012>' AFTER LAST.
END.
```

You now have a sorted file of variable-length records. But, each record is 96 characters long, with the line feed as the ninety-sixth character. Consequently, you can process this file as if it had data-sensitive records.

Now let's say you want to print the contents of file SO3 and to format the dollar amount field for printing. Assume this field has the format:

```
    9    14
...000.00...
```

You want to place a dollar sign before the field and to suppress leading zeroes within the field. To do so, specify:

```
FILTER/C INTO @LPT
INPUT FILE IS 'SO3',
   RECORDS ARE VARIABLE UPTO 96 CHARACTERS.
COPY.
INSERT '$' BEFORE 9.
REPLACE '00' IN 10/11 WITH '<040><040>'.
REPLACE '0' IN 10/10 WITH '<040>'.
END.
```

The INSERT statement adds a character to the record, making the output records 97 characters long. The dollar amount field in the output record now looks like:

```
    9     15
...$000.00...
```

The first REPLACE statement replaces the leading zeros in character positions 10 and 11 with two blank spaces; thus, the printing of all single digit dollar amounts looks like:

```
... $ 9.99...
```

The second REPLACE statement replaces a single leading zero in character position 10 with a blank; thus, the printing of double digit dollar amounts looks like:

```
...$ 29.99...
```

Note that these changes do not affect the input records. They have the same format before and after the operation.

If you don't want to specify this full set of instructions every time you want to perform this operation, specify a trail file when you invoke the utility and create the command file. For example:

```
FILTER/C/T=DAILY_SO3/O INTO @LPT
INPUT FILE IS 'SO3',
   RECORDS ARE VARIABLE UPTO 96 CHARACTERS.
COPY.
INSERT '$' BEFORE 9.
REPLACE ANY '00' IN 10/11 WITH '<040><040>'.
REPLACE ANY '0' IN 10/10 WITH '<040>'.
END.
```

Now, when you want to perform this task, you can simply specify:

```
FILTER/C=DAILY_SO3/O INTO @LPT
```

End of Chapter

Chapter 7
INFOS Files

The utility lets you sort, merge, and copy INFOS files as well as AOS files. However, to process INFOS files you must specify a command file and define the input and output files in appropriate declarations. When you process INFOS files you may use the Table, Key, and Work File declarations described in Chapter 4. You must, however, use the Input and Output File declarations described in this chapter. You may also use all the massaging statements and all the imperatives described in Chapter 5.

INFOS Input File Declaration

The format of the INFOS Input File declaration is shown below.

```
INPUT INFOS INDEX IS 'name'
[, PATH IS { DOWN
           | *
           | 'literal'
           | 'literal' : 'literal'
           | 'literal' -
           | - 'literal'
           | 'literal' + } [, RECORD] [, HEADER] [, [TRIMMED] PARTIAL RECORD]
[, KEY [PADDED TO integer {CHARS | CHARACTERS} WITH {integer | 'literal'}] [, IGNORE LOGICAL DELETES] [, ...]]
{, RECS | , RECORDS} ARE { integer {CHARS | CHARACTERS}
                          | VARIABLE UPTO integer {CHARS | CHARACTERS} }.
```

The PATH IS clause is always optional but the RECORDS ARE clause is always required. The PATH IS clause specifies a traversal path through the input file's index, telling what information is to be extracted at each subindex level. The RECORDS ARE clause defines the database record's format so the system can construct an appropriate AOS output file.

If you do not specify a PATH IS clause, you are, in effect, telling the system to retrieve the records associated with the main index keys. A PATH IS clause consists of at least one key phrase usually followed by one or more qualifier phrases.

You use the *key phrases* to specify the path you want the utility to traverse through an INFOS index. There are two classes of key phrases: those dealing with entire subindex levels (DOWN and *), and those dealing with a key or range of keys in a particular subindex ('literal', 'literal'+, 'literal':'literal', 'literal'-, and -'literal').

The key phrases and their meanings are:

- DOWN

  You want the utility to pass completely through a subindex level (without visiting any of its keys) to the subindex linked to the lowest-value key in the first subindex. You cannot qualify a DOWN key phrase.

  Note that when you specify DOWN, the first key in the first subindex through which you are passing must be linked to a lower level subindex. For example, if DOWN is the first key phrase you specify, the first key (lowest value) in the main index must be linked to a lower level subindex. If it's not, you'll get a runtime error.

- * (asterisk)

  You want the utility to visit every INFOS key in each subindex at a given subindex level. However, if * is immediately preceded by DOWN, 'literal', or 'literal'+, the utility visits only the keys in the subindex it moves down to. You may qualify a * key phrase.

- 'literal'

  Within a particular subindex, you want the utility to visit only the specified INFOS key. You may qualify a 'literal' key phrase.

- 'literal':'literal'

  You want the utility to visit each INFOS key in the inclusive range you specify. That is, you want it to visit both the keys you specify and all the keys between them in a particular subindex. You must write a lower ASCII value INFOS key first, then the colon, then the higher ASCII value INFOS key. You may qualify a 'literal':'literal' key phrase.

- 'literal'-

  You want the utility to visit the INFOS key specified and each higher (ASCII) value INFOS key in a particular subindex. You may qualify a 'literal'- key phrase.

- -'literal'

  You want the utility to visit the lowest value INFOS key and each higher (ASCII) value key up to and including the INFOS key specified. You may qualify a -'literal' key phrase.

- 'literal'+

  You are specifying a generic INFOS key. A *generic key* is the leading portion of an actual INFOS key in a particular subindex. The actual INFOS key that the utility visits is the first key whose leading portion exactly matches the generic key specified. You may qualify a 'literal'+ key phrase.

  You may also substitute 'literal'+ wherever a 'literal' is used; for example, you may specify 'literal'+:'literal'.

Use the *qualifier phrases* to specify the information you want the utility to extract either from within a particular subindex or for each key at a subindex level. You may specify any combination of qualifier phrases (i.e., none, some, or all) for each unique key phrase you specify; however, you may never qualify the DOWN key phrase.

Note that if you use the IGNORE LOGICAL DELETES phrase in conjunction with a key phrase, you must specify it first, before specifying any other qualifier phrases for that particular key phrase.

The qualifier phrases and their meanings are:

- IGNORE LOGICAL DELETES

  If you intend to use this phrase in addition to other qualifier phrases for a given key phrase, then you must specify this phrase first, before specifying any other qualifier phrases.

  Normally, if a key or data record is marked as logically deleted, the utility ignores that key or data record. Therefore, if a key marked as logically deleted is also linked to a lower level subindex, it may interrupt your traversal path unless you specify IGNORE LOGICAL DELETES.

- RECORD

  You want the utility to retrieve the record for each INFOS key it visits for the corresponding key phrase.

- HEADER

  You want the utility to generate a two-word, nonprinting binary header for each INFOS key it visits for the corresponding key phrase. The first word of the header contains the length of the record associated with the INFOS key; the first byte of the second word contains the partial record length for that key; and the second byte of the second word contains the length of the key.

- PARTIAL RECORD

  You want the utility to retrieve the partial record associated with each INFOS key it visits for the corresponding key phrase. If you also specify the TRIMMED option, the utility strips all trailing nulls from the partial records it retrieves.

- KEY

  You want the utility to retrieve each INFOS key it visits for the corresponding key phrase. If you also specify the PADDED TO option, the utility pads each key it retrieves to the length specified with the character specified.

In the INFOS Input File declaration, the RECORDS ARE clause is always required. If the database associated with the index you named has fixed-length records, use the first form of the RECORDS ARE clause:

```
{,RECS   }
{,RECORDS} ARE integer {CHARS     }
                       {CHARACTERS}
```

If the database has variable-length records, use the second form:

```
{,RECS   }
{,RECORDS} ARE VARIABLE UPTO integer {CHARS     }
                                     {CHARACTERS}
```

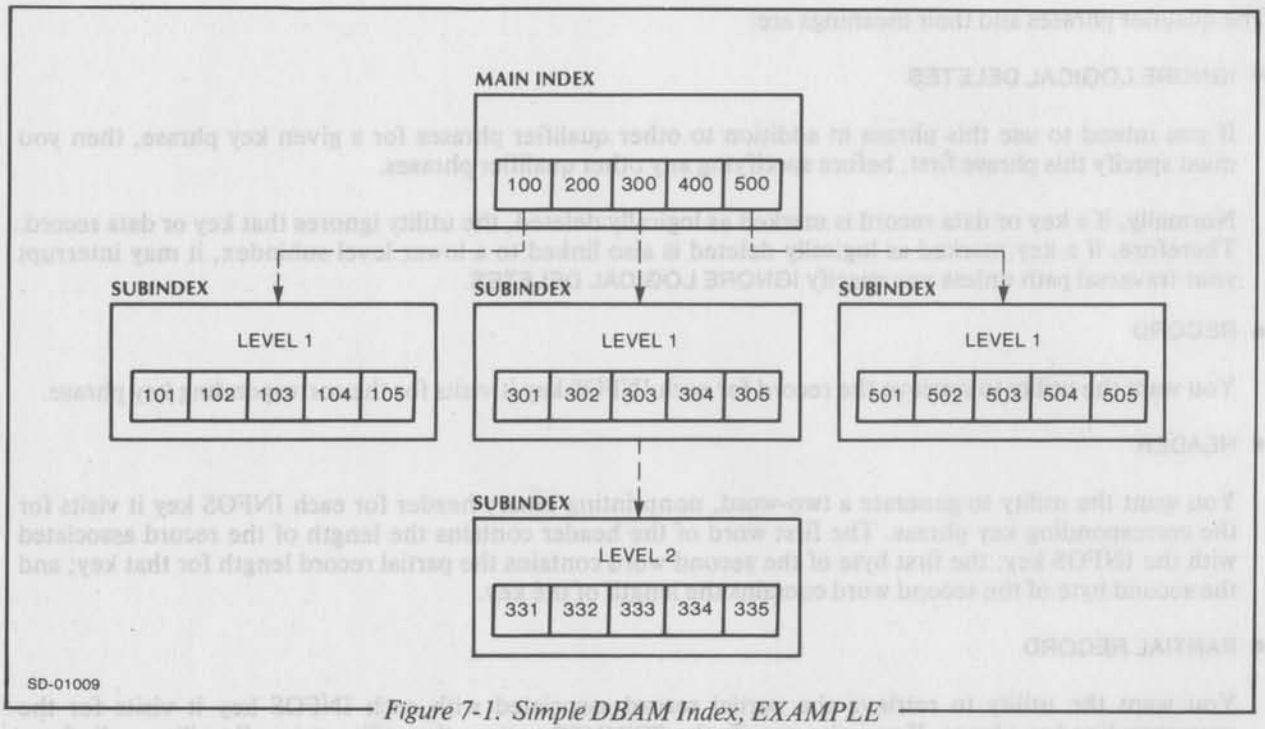


Figure 7-1. Simple DBAM Index, EXAMPLE

Figure 7-1 represents a simple DBAM index, named EXAMPLE, on which we base the following examples.

If you want to do a preordered traversal of this index, extracting the record associated with each key visited, specify:

```
INPUT INFOS INDEX IS 'EXAMPLE',
   PATH IS *,RECORD,
           *,RECORD,
           *, RECORD,
   RECORDS ARE VARIABLE UPTO 512 CHARACTERS.
```

Note the PATH IS clause. The first key phrase (*) tells the utility to visit each key in the main index. The first qualifier phrase (RECORD) tells the utility to retrieve the data record for each key it visits. The next key phrase - qualifier phrase pair (*, RECORD) specifies the same thing for all level one subindexes; and the last key phrase - qualifier phrase pair specifies the same for all level two subindexes.

In a preordered traversal of this index, the utility visits the keys in the following order (an arrow indicates a change in subindex level):

100 → 101, 102, 103, 104, 105 → 200 → 300, 301, 302, 303 → 331, 332, 333, 334, 335 → 304, 305 → 400 → 500 → 501, 502, 503, 504, 505

Suppose that key 303 is marked logically deleted. Then the utility visits the keys in the following order.

100 → 101, 102, 103, 104, 105 → 200 → 300, 301, 302, 304, 305 → 400 → 500 → 501, 502, 503, 504, 505

Thus, the utility does not retrieve the record associated with key 303 or the records for the keys 331 through 335. If you want to ensure that all records are retrieved for a preordered traversal of this index, specify:

```
INPUT INFOS INDEX IS 'EXAMPLE',
   PATH IS *, IGNORE LOGICAL DELETES, RECORD,
           *, IGNORE LOGICAL DELETES, RECORD,
           *, IGNORE LOGICAL DELETES, RECORD,
   RECORDS ARE VARIABLE UPTO 512 CHARACTERS.
```

If you want to retrieve the keys in the subindex under key 500, padding them to 10 characters with the minus sign (-), specify:

```
INPUT INFOS INDEX IS 'EXAMPLE',
   PATH IS '500', *, KEY PADDED TO 10 CHARACTERS WITH '-',
   RECORDS ARE VARIABLE UPTO 512 CHARACTERS.
```

Again, note the PATH IS clause. The first key phrase ('500') is not qualified. It tells the utility to visit key 500 but not extract any information associated with it. The second key phrase (*) tells the utility to visit every key in the subindex under key 500. Its qualifier phrase (KEY PADDED TO 10 CHARACTERS WITH '-') tells the utility to retrieve each key in the subindex, and force each to a fixed length of 10 characters by adding minus signs. Therefore, key "505" will be retrieved as "505-------". If you want to retrieve the keys, partial records, and data records for all keys in the level one subindexes, specify:

```
INPUT INFOS INDEX IS 'EXAMPLE',
   PATH IS *,
           *, KEY, PARTIAL RECORD, RECORD,
   RECORDS ARE VARIABLE UPTO 512 CHARACTERS.
```

The first key phrase (*) is not qualified, telling the utility to visit each key in the main index but not extract any information for those keys. You use the * key phrase so that the utility can locate those keys in the main index which are linked to level one subindexes. The second key phrase (*) tells the utility to visit every key in every level one subindex. Its qualifier phrase (KEY, PARTIAL RECORD, RECORD) tells the utility to extract the key, its partial record, and its associated data record, in that order, for each key visited.

If you want to retrieve the records for the keys in the level two subindex, specify:

```
INPUT INFOS INDEX IS 'EXAMPLE',
   PATH IS '300', '303', *, RECORD,
   RECORDS ARE VARIABLE UPTO 512 CHARACTERS.
```

Since the first two key phrases ('300' and '303') are not qualified, the utility does not extract any information for those keys. The combination *, RECORD tells the utility to visit every key in the subindex under key 303 and to retreive their data records.

If you want to retrieve the records for the first three keys in the subindex under key 100, specify:

```
INPUT INFOS INDEX IS 'EXAMPLE',
   PATH IS DOWN, -'103', RECORD,
   RECORDS ARE VARIABLE UPTO 512 CHARACTERS.
```

The key phrase DOWN tells the utility to pass through the main index without visiting any of its keys. The key phrase - '103' tells the utility to visit all keys (in the subindex) whose value is less than 103 and key 103. Its corresponding qualifier phrase RECORD tells the utility to retrieve the records for the keys it visits.

NOTE: While the DOWN key phrase tells the utility to pass through the main index, the first (lowest value) key in the main index must be linked to a lower level subindex. If it isn't, you'll get a runtime error.

If you want to retrieve the records for the last three keys in the subindex defined under key 500, specify:

```
INPUT INFOS INDEX IS 'EXAMPLE',
   PATH IS '500', '503'-, RECORD,
   RECORDS ARE VARIABLE UPTO 512 CHARACTERS.
```

The key phrase '503'- tells the utility to visit key 503 and all higher value keys in the subindex.

To retrieve the partial records for the keys 332, 333, and 334 specify:

```
INPUT INFOS INDEX IS 'EXAMPLE',
  PATH IS '300', '303', '332':'334', PARTIAL RECORD
  RECORDS ARE VARIABLE UPTO 512 CHARACTERS.
```

The key phrase '332':'334' tells the utility the range of keys you want it to visit.

INFOS Output File Declaration

The utility cannot create INFOS output files. Consequently, any INFOS output file you specify must exist when you invoke the utility. However, note that the utility's operation is not affected by the presence or absence of the global switch /O, so you need not specify it when your output is an INFOS file.

The format of the INFOS Output File declaration is:

```
OUTPUT INFOS { INDEX     } IS 'name'
             { INVERSION }

[ [, RECORD IS { integer/integer }] [, PARTIAL RECORD IS { integer/integer }] [, TRIM [{ integer   } FROM] KEYS] [, PATH IS { DOWN              } /, .../ ] ]
               { integer/LAST    }                       { integer/LAST    }            { 'literal' }                            { *                 }
                                                                                                                                 { 'literal' [ + ]   }
                                                                                                                                 {           [ - ]   }
```

Notice that in the OUTPUT INFOS clause you must choose one of two options, INDEX or INVERSION. The INDEX option indicates that the output INFOS file has no relationship to any of the input files. Therefore, you can use AOS or INFOS files as input files. The INVERSION option, however, indicates that the INFOS output file index is linked to the same database as all the indexes of all the input files you specify. Thus, if you specify:

```
OUTPUT INFOS INDEX IS 'name'.
```

you can use one or more AOS input files and/or one or more INFOS input files. But, if you specify:

```
OUTPUT INFOS INVERSION IS 'name'.
```

you can use only INFOS input files whose indexes are linked to the same database as the index of the output file you specify as 'name'. For example, you can have three indexes and a single database, as shown in Figure 7-2.

To link INDEX_3, you could specify:

```
INPUT INFOS INDEX IS 'INDEX_1',
  PATH IS *, RECORD,
  RECORDS ARE VARIABLE UPTO 512 CHARACTERS.
INPUT INFOS INDEX IS 'INDEX_2',
  PATH IS *, RECORD,
  RECORDS ARE VARIABLE UPTO 512 CHARACTERS.
OUTPUT INFOS INVERSION IS 'INDEX_3',
  ...
  ...
```

Use the PATH IS clause to describe the key path you want the utility to follow in the output index; and, optionally, what you want written to the output index (e.g., a partial record).

NOTE: Do not use a PATH IS clause to specify the key you want written to the output index; instead, use a Key declaration, as described in Chapter 4. Also, remember that the pathnames and output subindexes must exist before you attempt inversion.

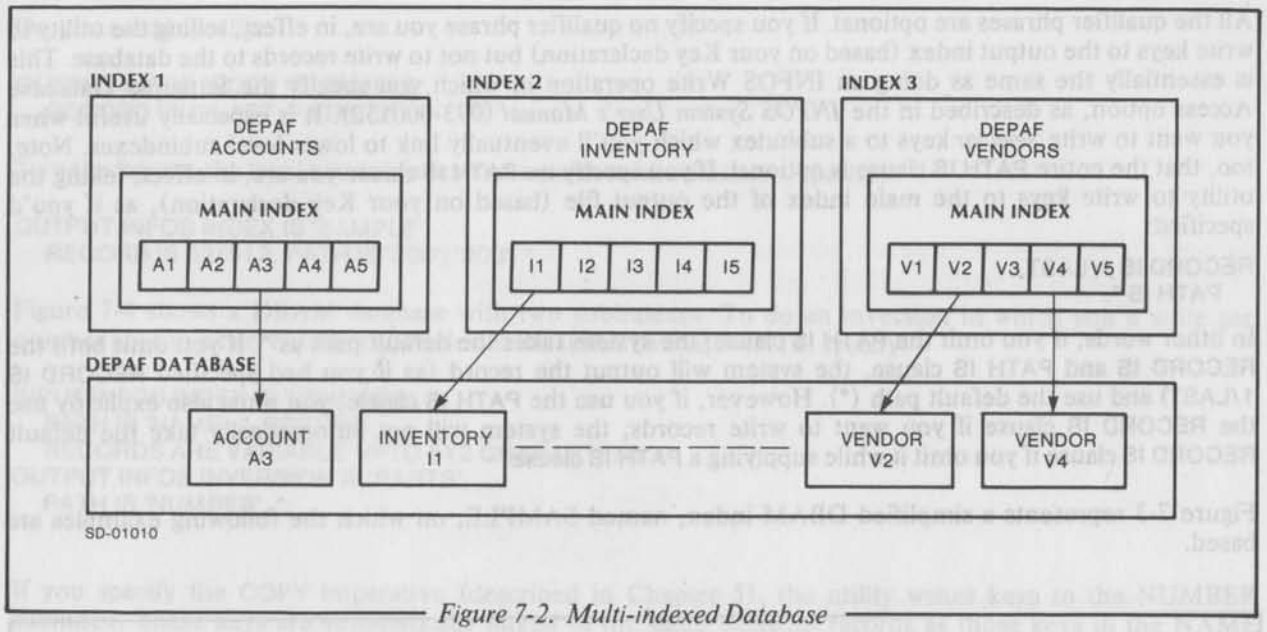


Figure 7-2. Multi-indexed Database

In the PATH IS clause, the *key phrases* (DOWN, *, 'literal', and 'literal'+) have the same meanings described for the INFOS Input File declaration PATH IS clause. You must specify * as the last key phrase in an output path; and it is the only key phrase which you may qualify. No key phrase except the last one in an output path may be qualified.

The *qualifier phrases* tell the utility what information to write to the output file. The qualifier phrases and their meanings are:

- RECORD IS

  You want the utility to write a data record to the output database. You specify the number of characters in the record in integer/integer or in integer/LAST. These ranges refer to character positions in the record after all massaging is done.

  NOTE: If you specify the INVERSION option, you must not specify the RECORD IS qualifier phrase, because the database record already exists. You are simply going to link another key to it.

- PARTIAL RECORD IS

  You want the utility to write the specified partial record for the key. You specify the number of characters in the partial record in integer/integer or in integer/LAST. These ranges refer to character positions in the record after all massaging is done.

- TRIM KEYS

  You want the utility to trim the character you specify from the ends of the keys it writes to the output index. Specifying TRIM KEYS is the same as specifying TRIM '<000>' FROM KEYS, which strips off trailing nulls. Since INFOS does not allow zero length keys, the utility will never trim a key to less than one character.

  NOTE: If you use the integer form of the option, the integer you specify represents the decimal equivalent of the character you want trimmed, (as shown in Appendix B). For example, specifying TRIM '*' FROM KEYS is the same as specifying TRIM 42 FROM KEYS.

All the qualifier phrases are optional. If you specify no qualifier phrase you are, in effect, telling the utility to write keys to the output index (based on your Key declaration) but not to write records to the database. This is essentially the same as doing an INFOS Write operation in which you specify the Suppress Database Access option, as described in the *INFOS System User's Manual* (093-000152). It is especially useful when you want to write *selector* keys to a subindex which you'll eventually link to lower level subindexes. Note, too, that the entire PATH IS clause is optional. If you specify no PATH IS clause you are, in effect, telling the utility to write keys to the main index of the output file (based on your Key declaration), as if you'd specified:

```
RECORD IS 1/LAST,
   PATH IS *.
```

In other words, if you omit the PATH IS clause, the system takes the default path as *. If you omit both the RECORD IS and PATH IS clause, the system will output the record (as if you had specified RECORD IS 1/LAST) and use the default path (*). However, if you use the PATH IS clause, you must also explicitly use the RECORD IS clause if you want to write records; the system will not automatically take the default RECORD IS clause if you omit it while supplying a PATH IS clause.

Figure 7-3 represents a simplified DBAM index, named SAMPLE, on which the following examples are based.

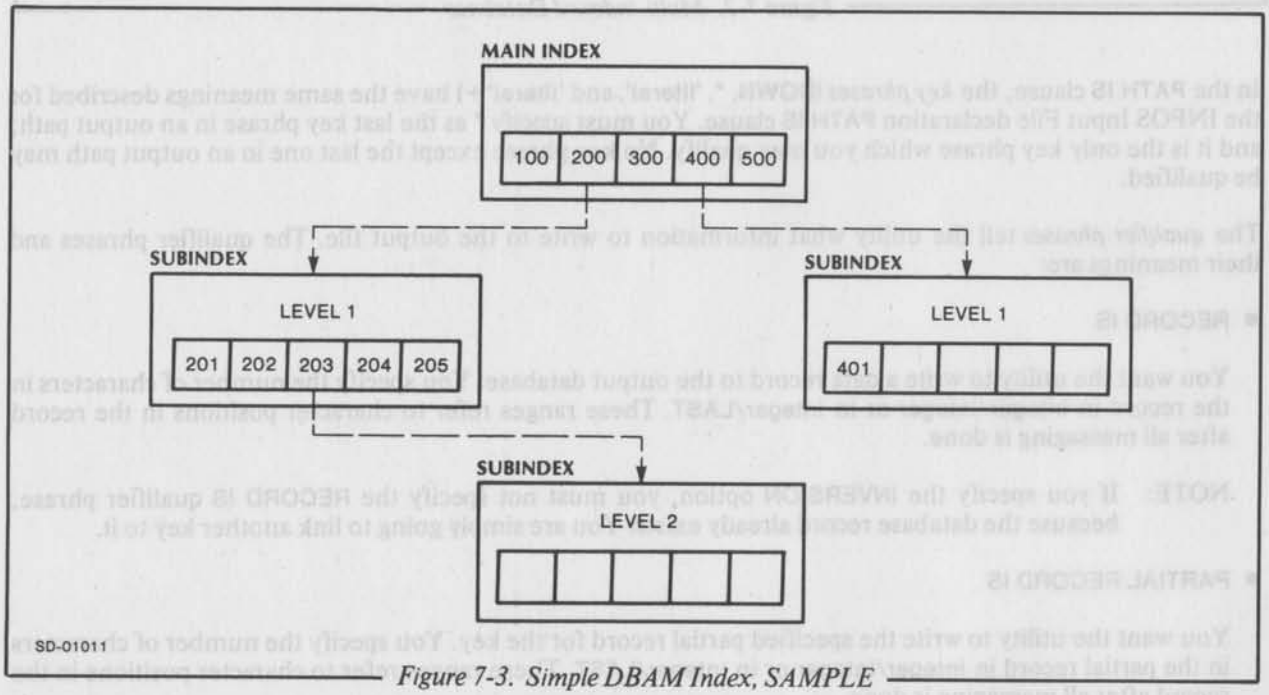


Figure 7-3. Simple DBAM Index, SAMPLE

NOTE: In the following, you'll define the key that you want the utility to write to the index by using a Key declaration as described in Chapter 4.

To write keys and partial records to the subindex defined under key 400, and records to the database, specify:

```
OUTPUT INFOS INDEX IS 'SAMPLE',
   RECORD IS 1/LAST,
      PARTIAL RECORD IS 20/50, PATH IS '400', *.
```

To write keys to the main index and records to the database, specify:

```
OUTPUT INFOS INDEX IS 'SAMPLE'.
```

Or, you could specify:

```
OUTPUT INFOS INDEX IS 'SAMPLE',
   RECORD IS 1/LAST, PATH IS *.
```

To write keys to the level two subindex and records to the database, specify:

```
OUTPUT INFOS INDEX IS 'SAMPLE',
   RECORD IS 52/512, PATH IS '200', '203', *.
```

Figure 7-4 shows a DBAM database with two subindexes. To do an inversion in which you'll write part number keys to a Part Number subindex in the index named PARTS, specify:

```
INPUT INFOS INDEX IS 'VENDORS',
   PATH IS 'NAME', *, RECORD,
   RECORDS ARE VARIABLE UPTO 512 CHARACTERS.
OUTPUT INFOS INVERSION IS 'PARTS',
   PATH IS 'NUMBER', *.
```

If you specify the COPY imperative (described in Chapter 5), the utility writes keys to the NUMBER subindex. These keys are automatically linked to the same database records as those keys in the NAME subindex (in the index named VENDORS).

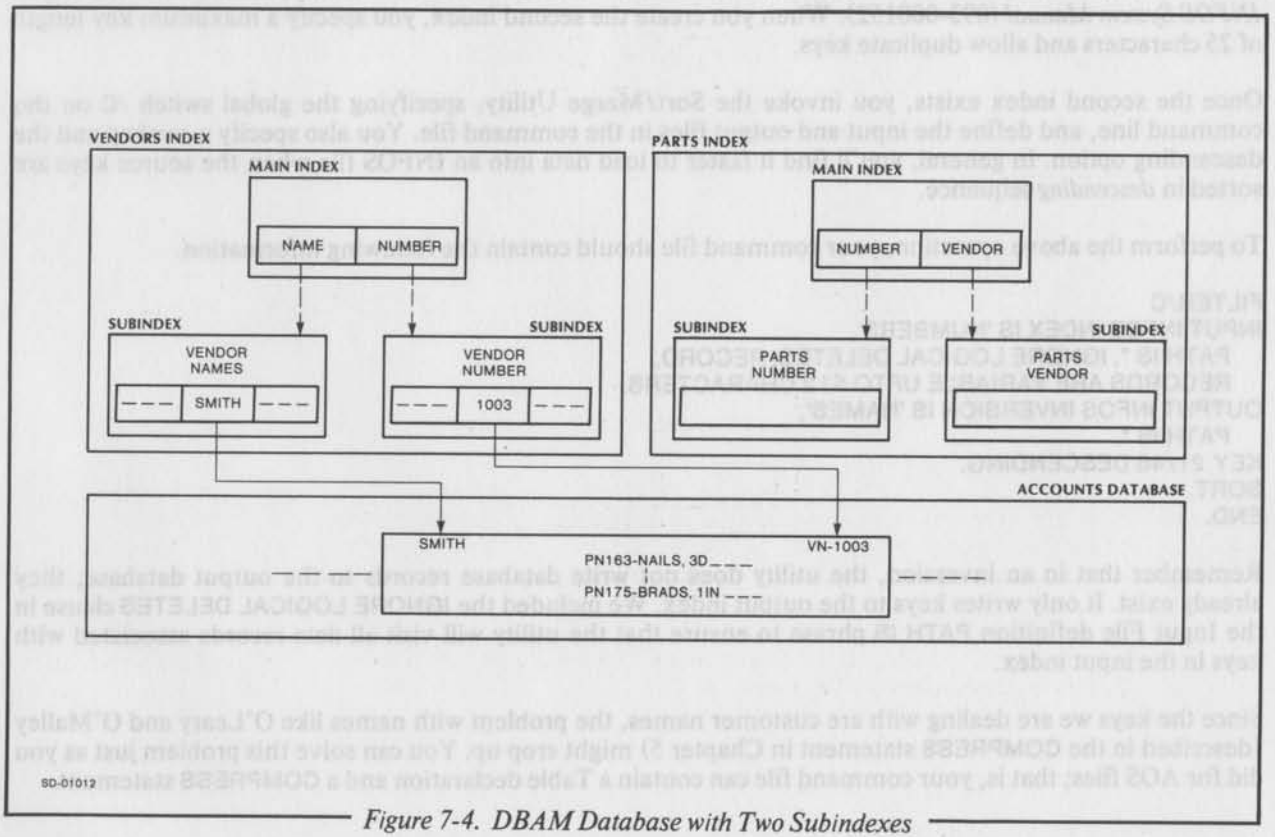
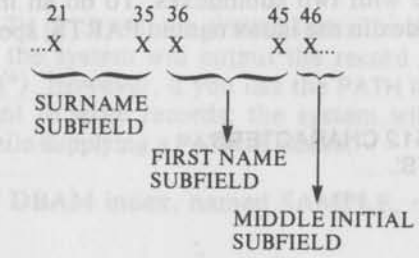


Figure 7-4. DBAM Database with Two Subindexes

Examples

Suppose you have an ISAM file containing customer accounts, keyed by account number. The index of this file is named NUMBERS and its database is named ACCOUNTS. As you know, INFOS keys are automatically maintained in each unique subindex and in the main index, from lowest ASCII value to highest value. This is analogous to always having the keys in any subindex, or main index, sorted in ascending sequence. However, in the ISAM database the records may be in any order.

Assume the database records are variable and up to 512 characters. The customer name field in these records is in character positions 21 through 46 in the following format.



If a name does not occupy an entire subfield, that subfield is padded with blanks.

Let's assume that you want to write customer name keys to a second index. The easiest way to create the second index, which you'll call NAMES, is to invoke the INFOS ICREATE Utility, described in the *AOS INFOS System Manual* (093-000152). When you create the second index, you specify a maximum key length of 25 characters and allow duplicate keys.

Once the second index exists, you invoke the Sort/Merge Utility, specifying the global switch /C on the command line, and define the input and output files in the command file. You also specify a *sort* key and the descending option. In general, you'll find it faster to load data into an INFOS file when the source keys are sorted in *descending* sequence.

To perform the above operation, your command file should contain the following information.

```
FILTER/C
INPUT INFOS INDEX IS 'NUMBERS',
   PATH IS *, IGNORE LOGICAL DELETES, RECORD,
   RECORDS ARE VARIABLE UPTO 512 CHARACTERS.
OUTPUT INFOS INVERSION IS 'NAMES',
   PATH IS *.
KEY 21/46 DESCENDING.
SORT.
END.
```

Remember that in an inversion, the utility does not write database records to the output database; they already exist. It only writes keys to the output index. We included the IGNORE LOGICAL DELETES clause in the Input File definition PATH IS phrase to ensure that the utility will visit all data records associated with keys in the input index.

Since the keys we are dealing with are customer names, the problem with names like O'Leary and O'Malley (described in the COMPRESS statement in Chapter 5) might crop up. You can solve this problem just as you did for AOS files; that is, your command file can contain a Table declaration and a COMPRESS statement.

```
FILTER/C
INPUT INFOS INDEX IS 'NUMBERS',
   PATH IS *, IGNORE LOGICAL DELETES, RECORD,
   RECORDS ARE VARIABLE UPTO 512 CHARACTERS.
OUTPUT INFOS INVERSION IS 'NAMES',
   PATH IS *.
TABLE TEMP FROM ASCII IS " ' "=0.
KEY 21/46 DESCENDING.
COMPRESS 21/35 LEFT '<040>' FILLED USING TEMP.
SORT.
END.
```

Figure 7-5 represents a simple multikey ISAM file (an INFOS DBAM file) named MASTER, on which the next two examples are based.

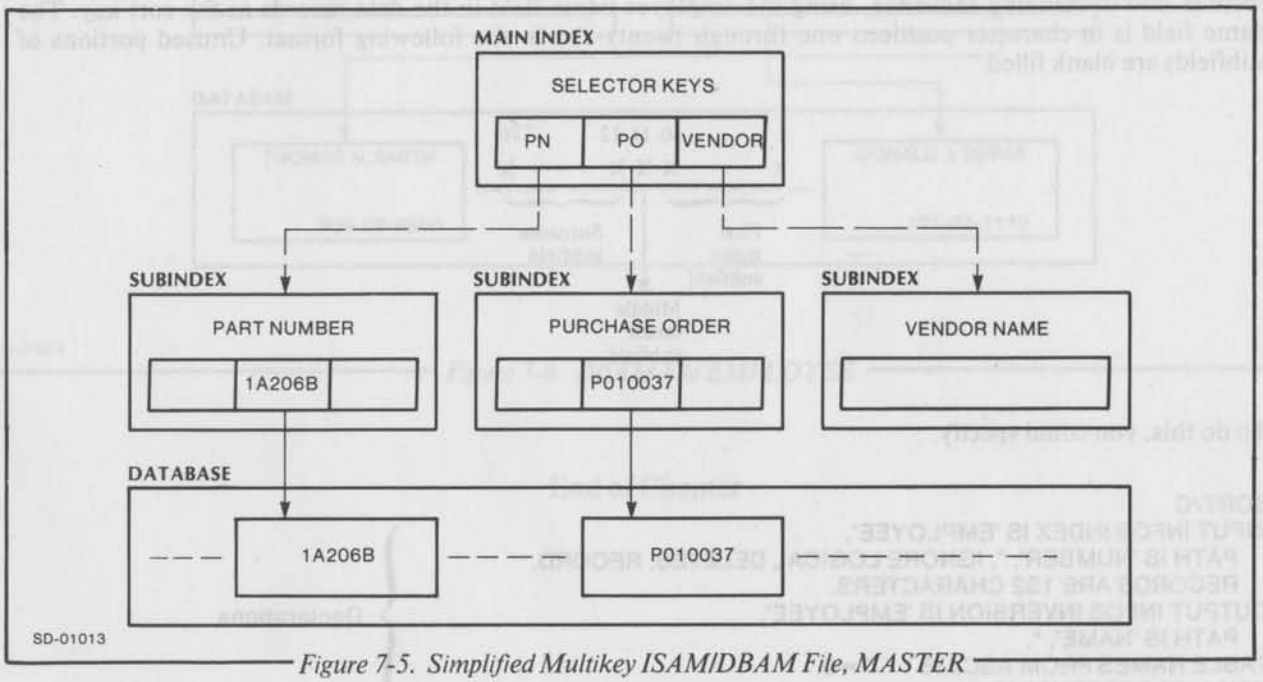


Figure 7-5. Simplified Multikey ISAM/DBAM File, MASTER

Let's say you have a previously sorted disk file named TEMP containing vendor records. This file was sorted in descending sequence, based on vendor name keys. Its records are 80 characters, with the name field occupying character positions 61 through 80. You want to copy the records from this file to the database of the MASTER file, writing keys and partial records to the Vendor Name subindex. Assume that the information for the partial records is in character positions 11 through 25 of the source records. To perform this task, you specify:

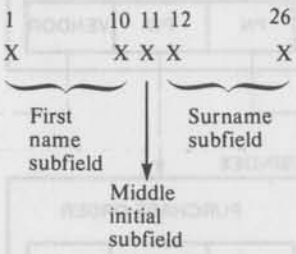
```
FILTER/C
INPUT FILE IS 'TEMP',
   RECORDS ARE 80 CHARACTERS.
OUTPUT INFOS INDEX IS 'MASTER',
   RECORD IS 1/LAST,
   PARTIAL RECORD IS 11/25,
   PATH IS 'VENDOR', *.
KEY 61/LAST.
COPY.
END.
```

In this example, the Key declaration tells the utility the location in the input record of the key you want to write. The clauses in the Output File declaration tell the utility what portion of the source record to write to the database, where the partial record you want it to write is located, and (in the PATH IS clause) where to write the keys.

Now let's assume you want to copy the database records associated with the keys in the Purchase Order subindex to an existing disk file named TEMP. The database records are variable and up to 132 characters. Specify:

```
FILTER/C/O INTO TEMP
INPUT INFOS INDEX IS 'MASTER',
   PATH IS 'PO, *, IGNORE LOGICAL DELETES, RECORD
   RECORDS ARE VARIABLE UPTO 132 CHARACTERS.
COPY.
END.
```

Suppose you have a DBAM file of employee records, named EMPLOYEE, currently keyed by employee social security numbers, as shown in Figure 7-6. You want to write name keys to the NAMES subindex, linking them to the existing data records. The data records are 132 characters long. You should sort the data records into descending sequence, using the employee name field in the data records as the sort key. The name field is in character positions one through twenty-six, in the following format. Unused portions of subfields are blank filled.



To do this, you could specify:

```
SORT/C
INPUT INFOS INDEX IS 'EMPLOYEE',
   PATH IS 'NUMBER', *, IGNORE LOGICAL DELETES, RECORD,
   RECORDS ARE 132 CHARACTERS.
OUTPUT INFOS INVERSION IS 'EMPLOYEE',                          } Declarations
   PATH IS 'NAME', *.
TABLE NAMES FROM ASCII IS " ' "=0.
KEY 1/26 DESCENDING.
REFORMAT 12/26, 1/11, 27/LAST.
IF 2/2=" ' " THEN COMPRESS 1/26 LEFT '<040>' FILLED USING NAME. } Directive-Group
SORT.
END.
```

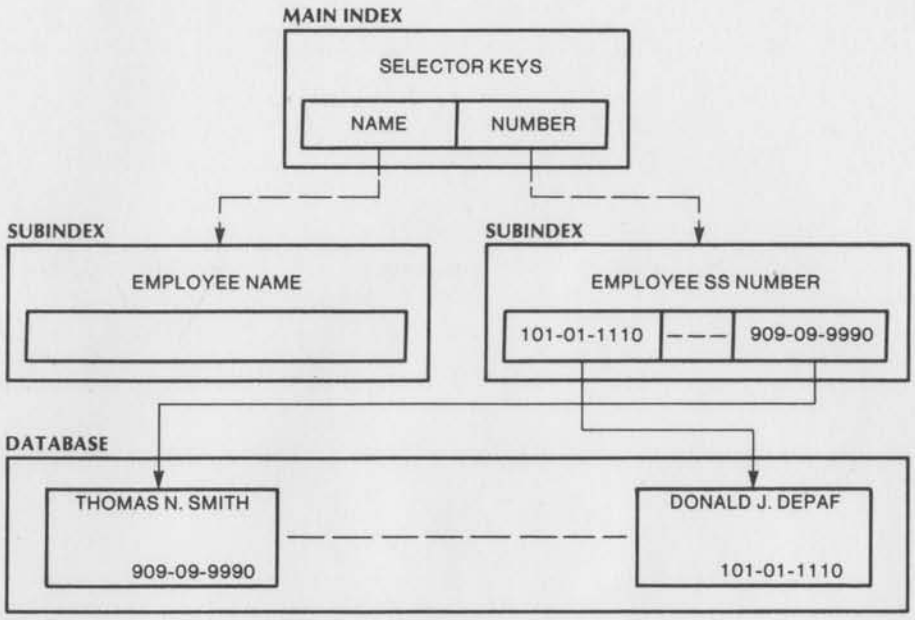


Figure 7-6. DBAM File EMPLOYEE

End of Chapter

Appendix A
Statistical Information Returned by the Utility

The utility normally generates a file of statistics for the current evocation. If you don't want the statistical output for the current evocation, specify the global switch /S on the CLI command line. Table A-1 shows the statistics generated for each stage of the utility's operation.

Table A-1. Statistics Produced per Operational Stage

During this Stage of the Utility's Operation	The Utility Performs these Operations	And Generates these Statistics
Setup and Validation	Verifies that there are no syntactical errors in the command line and/or Command File. Sets up the modules it needs to execute the requested operations.	*SETUP AND VALIDATION PHASE TIME* *OUTPUT FILE RECORD FORMAT*
Input	Reads the input records and massages them as specified. During this stage, the utility may skip certain input records, depending on the input IF statements you've specified.	*TOTAL NUMBER OF INPUT RECORDS* *MINIMUM INPUT RECORD LENGTH* *MAXIMUM INPUT RECORD LENGTH* *TOTAL NUMBER OF SKIPPED RECORDS*
Replacement Selection	Sorts the input records passed to this stage. The results of this stage are one or more *runs*. A run is a sorted string of records. The utility passes runs to the merge stage as input.	*REPLACEMENT SELECTION PHASE TIME* *TOTAL NUMBER OF OUTPUT RUNS* *RECORDS PASSED TO THE MERGE* *SIZE OF SELECTION TREE* *BIAS FACTOR*

Table A-1. Statistics Produced per Operational Stage (continued)

During this Stage of the Utility's Operation	The Utility Performs these Operations	And Generates these Statistics
Merge	Merges the contents of the runs produced in the replacement selection stage. Massages records as specified. During this stage, the utility may skip certain records depending on the output IF statements you've specified.	*REQUIRED NUMBER OF MERGE PASSES* *MAXIMUM ORDER OF MERGE* *PER PASS MERGE TIME* *TOTAL SKIPPED RECORDS (MERGE PHASE)*
Output	Writes the records to the output file.	*FINAL NUMBER OF RECORDS OUTPUT* *MINIMUM OUTPUT RECORD LENGTH* *MAXIMUM OUTPUT RECORD LENGTH* *TOTAL ELAPSED TIME*

If you invoke one of the Sorts, you'll get the full range of statistics shown in Table A-1. If you invoke the Merge operation, you'll get all the statistics shown except those for the replacement selection stage. If you invoke the Copy operation, you'll get the statistics not directly related to replacement selection or merge.

The statistical output is in decimal integers. If a fraction is encountered, the utility rounds it up to the next integer. The statistics and their meanings are as follows.

SETUP AND VALIDATION PHASE TIME

The time, in seconds, it takes the utility to verify that there are no syntactical errors in the command line and/or Command File, and to set up its modules to execute the functions you've requested.

OUTPUT FILE RECORD FORMAT

The utility generates an output file with variable-, or fixed-length, or data-sensitive records, depending on the input record's format. (See Chapter 2.)

TOTAL NUMBER OF INPUT RECORDS

The sum of all the records the utility encounters in all the input files.

MINIMUM INPUT RECORD LENGTH

The length in characters of the shortest input record the utility encounters.

MAXIMUM INPUT RECORD LENGTH

The length in characters of the longest input record the utility encounters.

TOTAL NUMBER OF SKIPPED RECORDS

The number of records the utility does not write to the output file due to IF statements in which the SKIP option is specified.

REPLACEMENT SELECTION PHASE TIME

The amount of time in seconds it takes the utility to sort the input records.

TOTAL NUMBER OF OUTPUT RUNS

The number of sorted strings the utility produces in the replacement selection phase.

RECORDS PASSED TO THE MERGE

The total number of records in all the runs produced by the replacement selection phase.

SIZE OF SELECTION TREE

The maximum number of records the utility places in a single run.

BIAS FACTOR

A measure of the relative disorder of the input. A bias factor of 2.0 indicates that the input files had an average amount of randomness to their records. If the bias factor is less than 2.0, the input files were more disordered than normal; a bias factor of 1.0 indicates that the file was ordered descending, rather than ascending. A bias factor of more than 2.0 indicates a relatively well ordered input file.

REQUIRED NUMBER OF MERGE PASSES

The number of times the utility has to merge runs. We'll explain more about this later.

MAXIMUM ORDER OF MERGE

This statistic reflects the largest number of runs the utility merges to form a single run. We'll explain more about this later.

PER PASS MERGE TIME

The time in seconds it took the utility to merge the input runs into output runs. More about this later.

TOTAL SKIPPED RECORDS (MERGE PHASE)

The number of records the utility does not place into an output run due to IF statements in which you specified the SKIP option for output massage.

FINAL NUMBER OF RECORDS OUTPUT

The actual number of records the utility writes to the output file.

MINIMUM OUTPUT RECORD LENGTH

The length in bytes of the shortest record the utility wrote to the output file.

MAXIMUM OUTPUT RECORD LENGTH

The length in bytes of the longest record the utility wrote to the output file.

TOTAL ELAPSED TIME

The actual amount of time from the moment you invoke the utility until the utility writes the last output record.

The output of the replacement selection phase is one or more runs, where a run is a string of sorted records. The utility passes runs to the merge phase for merging into longer runs. The utility merges in phases, until there is a single output run which it writes to the output file. If you invoke the merge operation, the utility considers the input files which you specify as runs to be input to the merge phase; consequently, there is no replacement selection phase.

The utility can merge a maximum of 15 runs into a single run. If the replacement selection phase produces eight runs, a single merge pass is all that's required to merge them into a single final run, which the utility writes to the output file. In the unlikely case that the replacement selection phase produces 32 output runs, the utility merges a maximum of 15 runs into a single intermediate run, a maximum of 15 different runs into another intermediate run, and two runs into a third intermediate run. It does all this in a single merge pass, as shown in Figure A-1.

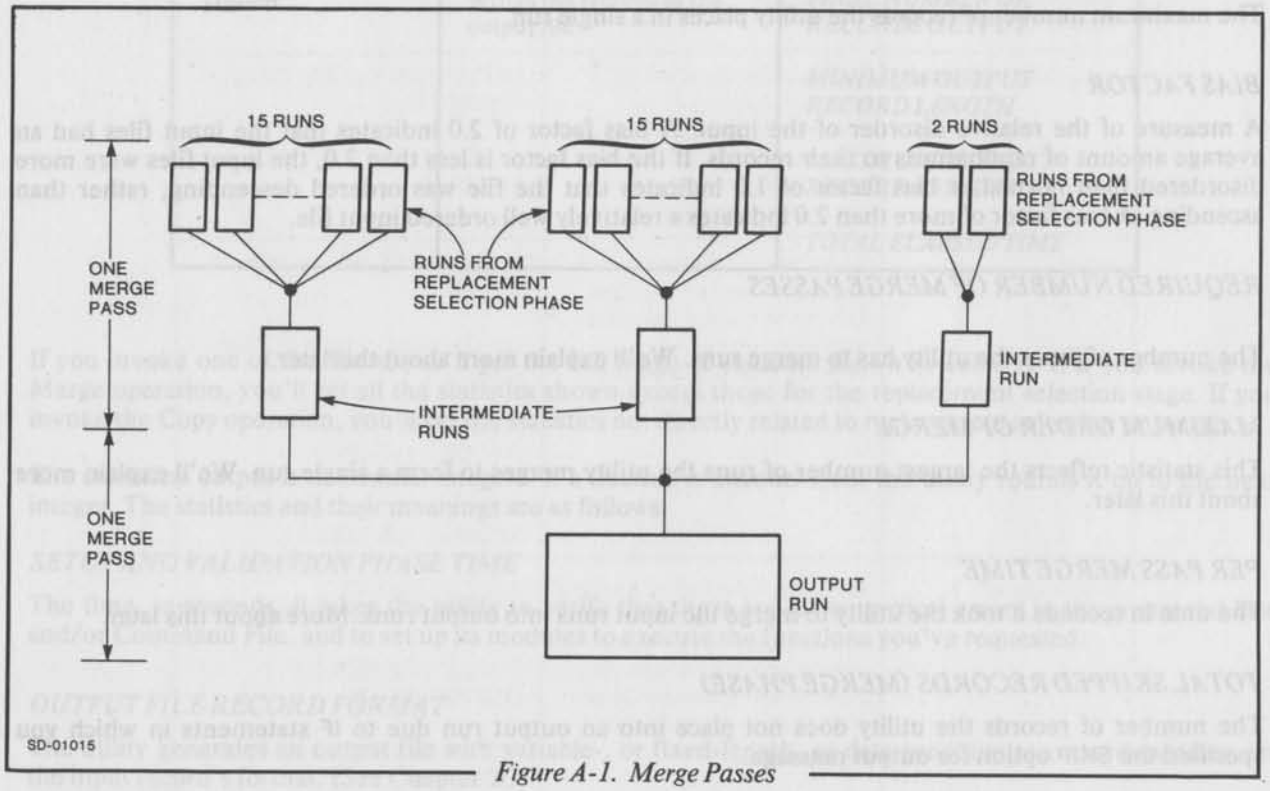


Figure A-1. Merge Passes

The utility merges the three intermediate runs (in a three-way merge) in a second merge pass, producing a single final run. It writes the contents of the final run to the output file. Note that in some cases it's possible for the utility to simply rename the final output run to the name of the output file, thus avoiding writing the output file.

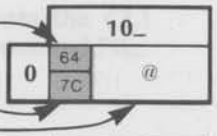
End of Appendix

Appendix B
Character Set

To find the *octal* value of a character, locate the character, and combine the first two digits at the top of the character's column with the third digit in the far left column.

LEGEND:

Character code in decimal
EBCDIC equivalent hexadecimal code
Character



OCTAL	00_		01_		02_		03_		04_		05_		06_		07_	
0	0 00	NUL	8 16	BS (BACK-SPACE)	16 10	DLE ↑P	24 18	CAN ↑X	32 40	SPACE	40 4D	(	48 F0	0	56 F8	8
1	1 01	SOH ↑A	9 05	HT (TAB)	17 11	DC1 ↑Q	25 19	EM ↑Y	33 5A	!	41 5D	)	49 F1	1	57 F9	9
2	2 02	STX ↑B	10 15	NL (NEW LINE)	18 12	DC2 ↑R	26 3F	SUB ↑Z	34 7F	'' (QUOTE)	42 5C	*	50 F2	2	58 7A	:
3	3 03	ETX ↑C	11 0B	VT (VERT. TAB)	19 13	DC3 ↑S	27 27	ESC (ESCAPE)	35 7B	#	43 4E	+	51 F3	3	59 5E	;
4	4 37	EOT ↑D	12 06	FF (FORM FEED)	20 3C	DC4 ↑T	28 1C	FS ↑\	36 5B	$	44 6B	, (COMMA)	52 F4	4	60 4C	<
5	5 2D	ENQ ↑E	13 0D	RT (RETURN)	21 3D	NAK ↑U	29 1D	GS ↑]	37 6C	%	45 60	-	53 F5	5	61 7E	=
6	6 2E	ACK ↑F	14 0E	SO ↑N	22 32	SYN ↑V	30 1E	RS ↑↑	38 50	&	46 4B	. (PERIOD)	54 F6	6	62 6E	>
7	7 2F	BEL ↑G	15 0F	SI ↑O	23 26	ETB ↑W	31 1F	US ↑←	39 7D	' (APOS)	47 61	/	55 F7	7	63 6F	?

OCTAL	10_		11_		12_		13_		14_		15_		16_		17_	
0	64 7C	@	72 C8	H	80 D7	P	88 E7	X	96 79	` (GRAVE)	104 88	h	112 97	p	120 A7	x
1	65 C1	A	73 C9	I	81 D8	Q	89 E8	Y	97 81	a	105 89	i	113 98	q	121 A8	y
2	66 C2	B	74 D1	J	82 D9	R	90 E9	Z	98 82	b	106 91	j	114 99	r	122 A9	z
3	67 C3	C	75 D2	K	83 E2	S	91 8D	[	99 83	c	107 92	k	115 A2	s	123 C0	{
4	68 C4	D	76 D3	L	84 E3	T	92 E0	\	100 84	d	108 93	l	116 A3	t	124 4F	|
5	69 65	E	77 D4	M	85 E4	U	93 9D	]	101 85	e	109 94	m	117 A4	u	125 D0	}
6	70 C6	F	78 D5	N	86 E5	V	94 5F	↑ or ~	102 86	f	110 95	n	118 A5	v	126 A1	~ (TILDE)
7	71 C7	G	79 D6	O	87 E6	W	95 6D	← or _	103 87	g	111 96	o	119 A6	w	127 07	DEL (RUBOUT)

SD-00217 Character code in octal at top and left of charts.

↑ means CONTROL

End of Appendix

Appendix C
Command Summary

This appendix summarizes the utility's commands. Each command's format is shown, from the CLI command you use to invoke the utility to the END statement you use to signal the end of the Command File.

CLI Command Lines

1. SORT [{ /L / /L=*filename* }] *[/N] [/O] [/S]* INTO outfile FROM infile ...

2. MERGE [{ /L / /L=*filename* }] *[/N] [/O] [/S]* INTO outfile FROM infile$_1$ infile$_2$ ...

3. { SORT / MERGE / FILTER } { /C *[/T=filename]* / /C=filename } [{ /L / /L=*filename* }] *[/N] [/O] [/S]* [*INTO outfile [FROM infile ...]*]

Command File Format

[DECLARATION] ... DIRECTIVE-GROUP END .

Declaration Sequence

[{ *AOS INPUT FILE* / *INFOS INPUT FILE* }] ... [{ *AOS OUTPUT FILE* / *INFOS OUTPUT FILE* }] *[TRANSTABLE]* ... *[KEY]* ... *[WORKFILE]* ...

Directive Group

[*[REFORMAT] [REPLACE] [INSERT] [PAD_TO] [TRANSLATE] [COMPRESS] [IF]*] ... { SORT / STABLE SORT / TAG SORT / STABLE TAG SORT / MERGE / COPY }

[*[REFORMAT] [REPLACE] [INSERT] [PAD_TO] [TRANSLATE] [COMPRESS] [IF]*] ...

End Statement

END .

AOS Input File Declaration

INPUT FILE IS 'name'

```
                  ⎧ integer {CHARS | CHARACTERS}                                                                   ⎫
, {RECS | RECORDS} ARE ⎨ {DATA SENS | DATA SENSITIVE} [DELIMITED BY 'literal'] UPTO integer {CHARS | CHARACTERS}  ⎬ .
                  ⎩ VARIABLE UPTO integer {CHARS | CHARACTERS}                                                    ⎭
```

INFOS Input File Declaration

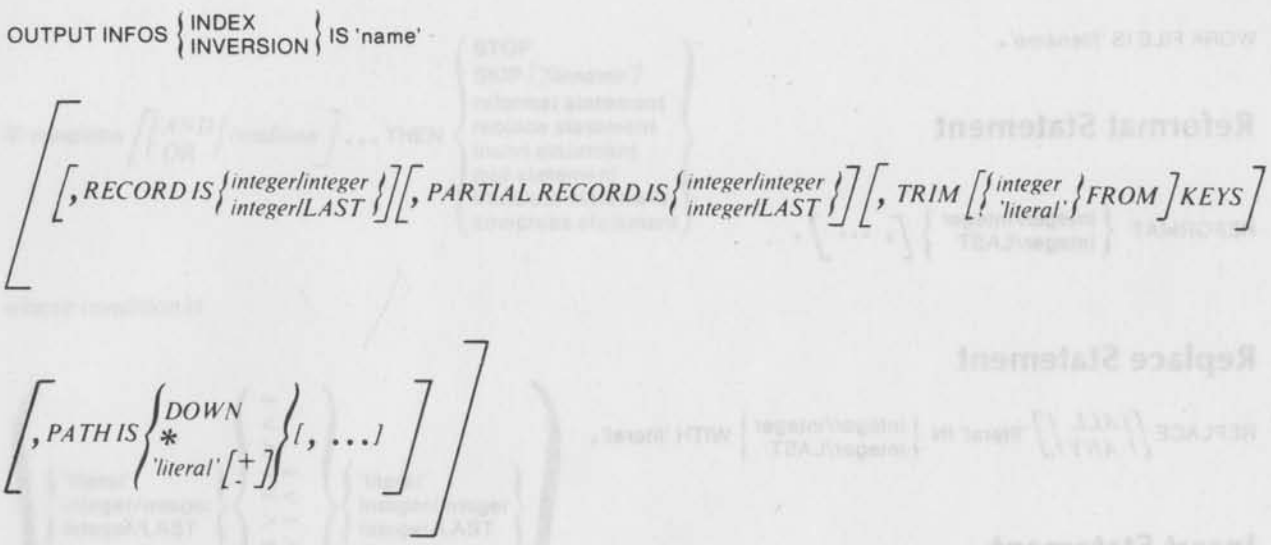
INPUT INFOS INDEX IS 'name'

```
[, PATH IS {{DOWN | * | 'literal' | 'literal' : 'literal' | 'literal' - | - 'literal' | 'literal' +}} [, RECORD] [, HEADER] [, [TRIMMED] PARTIAL RECORD]

[, KEY [PADDED TO integer {CHARS | CHARACTERS} WITH {integer | 'literal'}]] [, IGNORE LOGICAL DELETES]} [, ...]]

{, RECS | , RECORDS} ARE {integer {CHARS | CHARACTERS} | VARIABLE UPTO integer {CHARS | CHARACTERS}} .
```

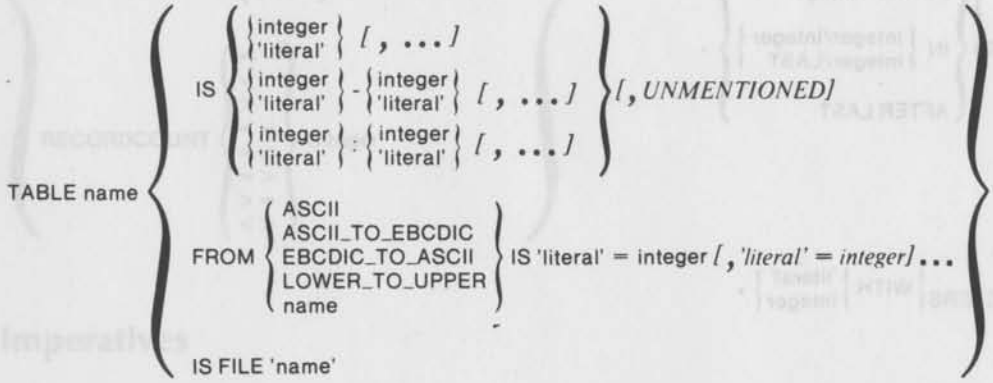
AOS Output File Declaration

OUTPUT FILE IS 'name' .

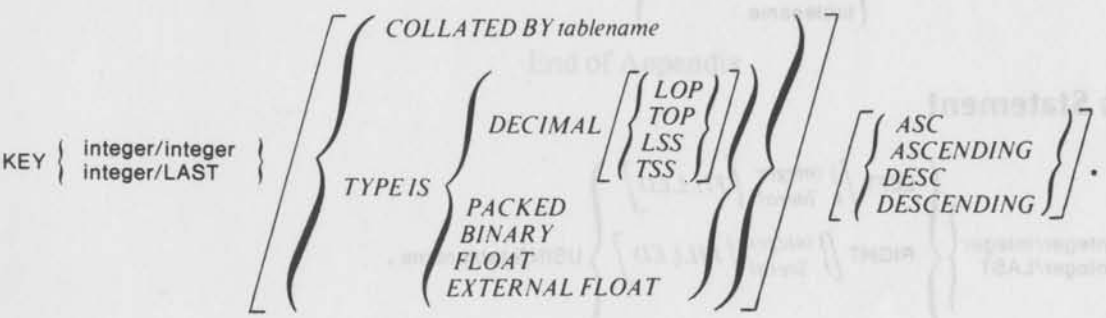
INFOS Output File Declaration



Translation Table Declaration



Key Declaration



Work File Declaration

WORK FILE IS 'filename'.

Reformat Statement

REFORMAT { integer/integer | integer/LAST } [, ...].

Replace Statement

REPLACE [{ ALL | ANY }] 'literal' IN { integer/integer | integer/LAST } WITH 'literal'.

Insert Statement

INSERT { TAG | RECORDCOUNT | 'literal' } { BEFORE integer | IN { integer/integer | integer/LAST } | AFTER LAST }.

Pad Statement

PAD TO integer { CHARS | CHARACTERS } WITH { 'literal' | integer }.

Translate Statement

TRANSLATE { integer/integer | integer/LAST } USING { ASCII_TO_EBCDIC | EBCDIC_TO_ASCII | LOWER_TO_UPPER | tablename }.

Compress Statement

COMPRESS { integer/integer | integer/LAST } { LEFT [{ integer | 'literal' } FILLED] | RIGHT [{ integer | 'literal' } FILLED] | VARIABLE } USING tablename.

If Statement

IF condition [{ AND / OR } condition] ... THEN { STOP / SKIP ['filename'] / reformat statement / replace statement / insert statement / pad statement / translate statement / compress statement }

where *condition* is

{ { 'literal' / integer/integer / integer/LAST } { = / < / > / <= / =< / >= / => / <> / :=: / :<>: } { 'literal' / integer/integer / integer/LAST } / RECORDCOUNT { = / < / > / <= / =< / => / >= / <> } integer }

Imperatives

SORT.
STABLE SORT.
TAG SORT.
STABLE TAG SORT.
MERGE.
COPY.

End of Appendix

Index

Within this index, f or ff after a page reference means "and the following page (or pages)." Primary references to topics are indicated by *italic* page numbers.

AFTER LAST phrase 5-5
altering collating sequence 1-2, 4-7, 4-9
AND operations 5-8
apostrophes (as delimiters) *3-2*
ASCII character set Appendix B
  simplified 4-5

/C (control command file) *2-2*, 2-4
character position 4-8
character set (ASCII/EBCDIC) Appendix B
  simplified ASCII 4-5
character type key 1-3
CLI (AOS) 1-1
  command lines *2-1*, C-1
COBOL
  data types 1-3
  interfacing 4-9
COLLATED BY phrase 4-9
collating sequence, alter 1-2, *4-4*, 4-9
collating table *4-5*, 4-7
command file 1-1, 2-4, *3-1ff*
  controlling (/C) *2-2*, 2-4
  declarations 3-1, *4-1ff*
  directive group *5-1ff*
  format *3-1ff*, C-1
commands (summary) Appendix C
compares, floating 5-8
COMPRESS statement 1-2, 4-7 *5-6ff*, C-4
controlling utility's output (IF) 5-7
conventions, documentation iii
convert lowercase to uppercase 5-5
COPY 1-1, 2-4, *5-10*
  example 7-9
create new fields 1-2
CTRL-D 2-4

data-sensitive records 1-3, 4-1f, 5-1
  in basic sort 2-2
  inserting tags *5-4*
  padding 5-5
DBAM example 7-4, 7-8
decimal type key 1-3
declarations 3-1, C-1
  input *4-2f*, C-2
  output *4-3*, C-2
  table *4-4*, C-3
  key *4-8*, C-3
  INFOS input *7-1f*, C-2
  INFOS output *7-6f*, C-3
defaults
  character set 4-9
  key type 4-9
default delimiters (AOS) 2-2, 4-2
deleting fields 1-2
delimiters
  AOS default 2-2, 4-2
  literal 3-2
  tables 4-2
device types 1-3
directive group 3-1, *5-1ff*, C-1
documentation conventions iii
DOWN phrase 7-2
duplicate keys 7-10

editing records 1-1
END *3-1*, C-2
examples, summary, of utility 6-1ff
execution, controlling (IN) 2-2

fields
  creating new 1-2
  deleting 1-2
files
  command 1-1, *3-1ff*
  input 1-1, *4-1f*
  master 1-1
  output 1-1, *4-3*
  skip 1-2, *5-9*
  @ LIST 2-1
  trail 2-2
FILTER 1-1, *2-1ff*
fixed-length records 1-3, 4-2
  in basic sort 2-2
  in INFOS input declaration 7-3
floating compares 5-8

generic keys 7-2

HEADER phrase 7-3

ICREATE utility (INFOS) 7-10
IF statement *5-7f,* C-5
IGNORE LOGICAL DELETES 7-2f
imperatives *5-10,* C-5
INDEX option 7-6
INFOS
  output file 2-4, *7-6f*
  input file 2-4, 5-1, *7-1f*
    example 7-4f, 7-10f
input files 1-1
  declarations 2-2, *4-1f,* C-2
  INFOS 5-1, *7-1ff,* C-2
input statistic A-1
INSERT 1-2, 5-1, *5-3f,* C-4
inserting new fields 1-2
INVERSION option 7-6

key phrases 7-1, 7-7
keys 1-3, *4-8*
  declaration *4-8,* C-3
  duplicate 7-10
  generic 7-2
  primary 4-8
  secondary 4-8
  selector 7-8

@ LIST file 2-1
/L (control output listing) 2-1
LEFT (justify) 5-6
literals 3-1f
location phrase 4-8
LOGICAL DELETES, IGNORE 7-2f
lowercase ASCII to uppercase, converting 5-5

massaging 1-1, *5-1*
MERGE 1-1, *2-1ff,* 5-10
  statistic A-2
  passes A-4

/N (control execution) *2-2*

/O (control output file) *2-2,* 2-4
operators (in IF) 5-8
OR operators 5-8
output, controlling utility's (IF) 5-7
output files 1-1
  controlling (/O) *2-2,* 2-4
  declaration *4-3,* C-2
  INFOS *7-6f,* C-3
output listing
  controlling (/L) 2-1
output statistic A-2

PAD 1-2, *5-5,* C-4
PADDED TO option 7-3
PARTIAL RECORD phrase
  INFOS input 7-3
  INFOS output 7-7
PATH IS clause *7-1,* 7-6
primary keys 4-8

qualifier phrases (INFOS)
  input 7-2f
  output 7-7f
quotation marks (as delimiters) 3-2

ranges 3-1, *4-6*
RECORD phrase 7-3
RECORDCOUNT *5-4,* 5-8
RECORD IS phrase 5-1
RECORDS ARE clause 7-1
record types 1-3
REFORMAT 1-2, *5-1,* C-4
remove unwanted characters 5-6
REPLACE 1-2, 5-1, *5-2f,* C-4
replacement selection statistic A-1
RIGHT (justify) 5-6

/S (control statistics) 2-1
secondary keys 4-8
selector keys 7-8
setup statistic A-1
skip file 1-2, *5-9*
skipping records 1-2

SORT 1-1, *2-1f,* 5-10
  stable 1-1, 2-4, *5-10*
  stable tag 1-1, 2-4, *5-10*
  standard 1-1, 2-4, *5-10*
  tag 1-1, 2-4, *5-10*
stable sort 1-1, *2-4*
stable tag sort 1-1, *2-4*
standard sort 1-1, *2-4*
statistics Appendix A
  suppress (/S) *2-1,* A-1
STOP option 5-9
subindex levels 7-1
suppress database option 7-8
suppress statistics 2-1
switches, global 2-1

/T (using trail file) *2-2,* 2-4
table declaration *4-4,* C-3
TAG 5-3
tag sort 1-1, *2-4,* 5-3
tailoring utility's operation 2-4
terminate your interaction (CTRL-D) 2-4
trail files 2-2
  controlling (/T) *2-2,* 2-4
TRANSLATE 1-2, *5-5,* C-4
  table declaration C-3
traversal, preordered, of INFOS index 7-4
TRIM KEYS phrase 7-7
TRIMMED option 7-3
TYPE IS phrase 4-9

UNMENTIONED clause 4-7

validation statistic A-1
VARIABLE option 5-6
variable-length records 1-3, 4-1, 4-3
  in INFOS input declaration 7-3

ork files 1-3
  declaration *4-9,* C-4

DataGeneral

Software Documentation Remarks Form

How Do You Like This Manual?

Title _____ No. _____

We wrote the book for you, and naturally we had to make certain assumptions about who you are and how you would use it. Your comments will help us correct our assumptions and improve our manuals. Please take a few minutes to respond.

If you have any comments on the software itself, please contact your Data General representative. If you wish to order manuals, consult the Publications Catalog (012-330).

Who Are You?

- ☐ EDP Manager
- ☐ Senior System Analyst
- ☐ Analyst/Programmer
- ☐ Operator
- ☐ Other _____

What programming language(s) do you use? _____

How Do You Use This Manual?

(List in order: 1 = Primary use)

- ____ Introduction to the product
- ____ Reference
- ____ Tutorial Text
- ____ Operating Guide
- ____ _____

Do You Like The Manual?

Yes	Somewhat	No	
☐	☐	☐	Is the manual easy to read?
☐	☐	☐	Is it easy to understand?
☐	☐	☐	Is the topic order easy to follow?
☐	☐	☐	Is the technical information accurate?
☐	☐	☐	Can you easily find what you want?
☐	☐	☐	Do the illustrations help you?
☐	☐	☐	Does the manual tell you everything you need to know?

Comments?

(Please note page number and paragraph where applicable.)

From:

Name _____ Title _____ Company _____

Address _____ Date _____

SD-00742

FOLD DOWN FIRST FOLD DOWN

FIRST
CLASS
PERMIT
No. 26
Southboro
Mass. 01772

BUSINESS REPLY MAIL

No Postage Necessary if Mailed in the United States

Postage will be paid by:

Data General Corporation

Southboro, Massachusetts 01772

ATTENTION: Software Documentation

FOLD UP SECOND FOLD UP

STAPLE