SERIES 60 (LEVEL 68)

MULTICS SORT/MERGE

SUBJECT

Detailed Description of the Generalized File Sorting and File Merging Capability, Including Details of the Commands and Subroutines Involved

SOFTWARE SUPPORTED

Multics Software Release 4.0

ORDER NUMBER

AW32, Rev. 0

July 1976

Honeywell

PREFACE

This manual describes the generalized file sorting and file merging capability on the Multics system. The Sort or Merge is specialized for execution by user-supplied parameters, for example to specify key fields. The Sort/Merge package is particularly designed to support files on any storage medium and in any file organization and also to support large files.

The reader is assumed to be familiar with the Multics system and, in particular, to have access to the Multics Programmers' Manual (MPM). The MPM consists of the following:

Reference Guide	Order No. AG91
Commands and Active Functions	Order No. AG92
Subroutines	Order No. AG93
Subsystem Writers' Guide	Order No. AK92

AW32

File No.: 1L53

CONTENTS

TABLES

SECTION I

FUNCTIONS

The Sort/Merge package provides a generalized file sorting and merging capability, which is specialized for execution by user supplied parameters. The package contains two components, the Sort and the Merge.

The basic function of the Sort is to read one or more input files of records which are not ordered, sort those records according to the values of one or more key fields, and write a single output file of ordered (or "ranked") records.

The basic function of the Merge is to read one or more input files of records which are in order according to the values of one or more key fields, merge (collate) those files, and write a single output file of ordered records.

Thus the primary difference between the Sort and the Merge is that the Sort processes files which are not in order, while the Merge processes files which are in order.

The Sort/Merge package has the following general capabilities:

Input and output files may be on any storage medium and in any file organization;

Very large files, such as multisegment files, can be sorted or merged;

Multiple key fields and most PL/I string and numeric data types may be specified;

Exits to user supplied subroutines are permitted at several points during the sorting or merging process.

The Sort/Merge package can be invoked either as a command or as a subroutine. The functions provided are almost identical in the two modes.

See Section II, "Commands," for specifications of the sort and merge commands. See Section III, "Subroutines," for specifications of the sort_ and merge_ subroutines.

In this manual, the term "Sort/Merge" is used to refer to functions or specifications which are the same for both the Sort and the Merge. The term "Sort" is used to refer to functions or specifications which apply only to the Sort component, but which are available either from command level or from subroutine level. Similarly, the term "Merge" is used to refer to functions or specifications which apply only to the Merge component, but which are available either from command level or from subroutine level. For functions or specifications which apply only to a specific command or subroutine, the specific command name (sort or merge) or the specific subroutine name (sort_ or merge_) is used.

In addition to arguments to the command or subroutine, other information is necessary to specialize the Sort/Merge for a particular execution. This information is called the Sort/Merge Description (described in detail in Section IV of this manual).

INPUT AND OUTPUT

The user can specify the input and output files. Input and output files are specified in the arguments to the command or subroutine. In this environment, the Sort/Merge reads the input files and writes the output file. Each input or output file may be stored on any medium and in any file organization supported by an I/O module through iox_. The I/O module may be one of the Multics system I/O modules (such as tape_ansi_), or one supplied by a specific installation, or one written by a user. An input or output file is specified either by a pathname or by an attach description.

Alternatively, for the Sort the user can supply either an input_file procedure or an output_file procedure (or both). For the Merge, input_file and output_file procedures are not permitted. An input_file procedure is responsible for reading input and releasing records to the Sort. An output_file procedure is responsible for retrieving records which have been ranked by the Sort and writing output.

In all cases, records may be either fixed length or variable length.

The maximum amount of input data (total size of all input files) which can be accepted by the Sort is approximately one billion (10**9) bytes. There is essentially no limit for the Merge.

KEY FIELDS

The user can specify the key fields to be used in ranking records. Refer to "Keys Statement" or "keys Structure" in Section IV for more detail on key descriptions. Up to 32 key fields may be specified. Any PL/I string or numeric data type -- except varying string, complex, or pictured -- may be specified for a given key field. Ranking may be ascending, descending, or mixed. For a character string field, the collating sequence is that of the Multics standard character set. For the Merge, the records of each input file must be in order according to those key fields.

Alternatively, the user can specify a user supplied compare procedure, which is then used to rank records. For the Merge, the records of each input file must be in order according to the algorithm of that user supplied compare procedure.

The original order of records with equal keys is preserved (FIFO order). Original input order is defined as follows:

1. If two equal records come from different input files, then the record from the file which is specified earlier (in the command or subroutine argument) is first.

2. If two equal records come from the same input file, then the record which is earlier in the file is first.

EXITS

The Sort/Merge provides exits to user supplied procedures at specific points during the sorting or merging process. Refer to "Exits Statement" or to "exits Structure" and "io_exits Structure" in Section IV for more detail on exit procedures. The following exit points are provided:

input_file	To obtain input records and release them one by one to the sorting process. For the Merge, the input_file exit is not provided.
output_file	To retrieve ranked records one by one from the sorting process and output them. For the Merge, the output_file exit is not provided.
input_record	To perform special processing for each input record, such as deleting, inserting, or altering records to be input to the Sort. For the Merge, the input_record exit is not provided.
output_record	To perform special processing for each output record, such as deleting, inserting, or altering records to be output from the Sort or Merge; or summarizing data by accumulating it into a summary record.
compare	To compare two records; that is, to rank them for the sorting or merging process.

WORK REQUIREMENTS

The Sort/Merge requires that its work files be allocated in the Multics storage system. Thus the user must have sufficient quota for the work files, in addition to that required for the output file if it is to be in the storage system.

There are two groups of work files required.

Sort Work Files

The Sort function requires a number of large segments, which are allocated in the directory specified by the user. As a first approximation, the space required by these segments is between 1.05 and 1.15 times the total size of all the input files.

A closer approximation to the size of the Sort work files is:

F + 64*sqrt(F)

where F is the total amount of data input to the Sort, in bytes.

The Merge function does not require these work files.

Process Directory Work Files

Both the Sort and the Merge functions require a small number of small segments, which are always allocated in the user's process directory. As a first approximation, the space required by these segments is from 3 to 6 storage system (1024 word) records.

A closer approximation for the process directory work files is:

One buffer segment for each input file, each segment as large as the largest record in that input file;

If the output_record exit is specified, two additional buffer segments, both as large as the largest output record;

Up to two segments of one storage system (1024 word) record each, for processing the Sort/Merge Description.

SECTION II

COMMANDS

This section describes both the sort command and the merge command. Additional information necessary for executing the sort or merge commands with user supplied exit procedures is contained in Section V, "Exit Procedures," of this manual.

The conventions used below for describing arguments are the same as those used in the Multics Programmers' Manual, Commands and Active Functions, Order No. AG92.

Usage

```
sort   input_specs  output_spec  -control_args-

merge  input_specs  output_spec  -control_args-
```

where:

1. input_specs

   The user is specifying the input files. Up to 10 input files may be specified. Each input file specification (each input_spec) may be supplied in one of the following forms:

   -input_file pathname, -if pathname

   If an input file is in the Multics storage system and its file organization is either sequential or indexed, then it may be specified by its pathname. The file may be either a single segment or a multisegment file. The star convention can not be used.

   An input file specified by a pathname will be attached using the attach description "vfile_ pathname".

   -input_description "attach_desc", -ids "attach_desc"

   If an input file is not in the Multics storage system or its file organization is neither sequential nor indexed, then it must be specified by an attach description. The attach description must be quoted. The target I/O module specified via the attach description must support the sequential_input opening mode and the iox_ entry point read_record.

   Pathnames and attach descriptions can be intermixed in the input_specs argument.

   For the sort command, if the user is supplying an input_file exit procedure then the input_specs argument must be omitted and the input_file exit procedure must be named in the Exits statement of the Sort Description. For the merge command, an input_file exit procedure cannot be specified.

2. output_spec
the user is specifying the output file. Only one output file can be specified. The output file specification (output_spec) may be supplied in one of the following forms:

-output_file pathname, -of pathname
If the output file is in the Multics storage system and its file organization is sequential, then it may be specified by its pathname. The file may be either a single segment or a multisegment file.

The equals convention may be used. If it is, it is applied to the pathname of the first input file and the first input file must be specified by a pathname, not by an attach description.

An output file specified by a pathname will be attached using the attach description "vfile_ pathname". Thus if the file does not exist, it will be created. If it does exist, it will be overwritten.

-output_file -replace, -of -rp
For the sort command, the output file is to replace the first input file. That input file will be overwritten during the merge phase of the Sort. If -replace is specified, the first input file must be specified by a pathname, not by an attach description. For the merge command, the -replace option cannot be specified.

-output_description "attach_desc", -ods "attach_desc"
If the output file is not in the Multics storage system or its file organization is not sequential, then it must be specified by an attach description. The attach description must be quoted. The target I/O module specified via the attach description must support the sequential_output opening mode and the iox_ entry point write_record.

For the sort command, if the user is supplying an output_file exit procedure then the output_spec argument must be omitted and the output_file exit procedure must be named in the Exits statement of the Sort Description. For the merge command, an output_file exit procedure cannot be specified.

3. control_args
must be chosen from the following:

-console_input, -ci
The Sort/Merge Description is read via the I/O switch user_input (which normally is the user's terminal).

-sort_desc sm_path, -sd sm_path, -merge_desc sm_path, -md sm_path
The user is specifying the pathname of the segment containing the Sort/Merge Description.

Either the -console_input argument or the -sort_desc or -merge_desc argument - but not both - must be specified. See "Sort/Merge Description Notes" below.

-temp_dir td_path, -td td_path
For the sort command, specifies the pathname of the directory which is to contain work files for the sorting process. The equals convention can not be used.

For the merge command, this argument is not required and must not be specified.

If the -temp_dir argument is omitted, work files for the Sort will be contained in the user's process directory.

This argument should be specified when the process directory will not be large enough to contain the work files for the Sort. The [wd] active function may be specified for td_path to place these work files in the user's current working directory.

For both the Sort and the Merge, certain small work files are always placed in the user's process directory.

-file_size f

For the sort command, specifies that the total amount of data to be sorted is f millions of bytes. The argument f must be a decimal number.

For the merge command, this argument is not required and must be omitted.

If the -file_size argument is omitted, the default assumption for the Sort is approximately one million bytes (f = 1.0).

This argument is intended for use when some or all of the input files are not in the Multics storage system (that is, are not specified by pathnames) or when an input_file exit procedure is specified. In these cases the Sort cannot determine the amount of input data. The -file_size argument may also be specified when all of the input files are in the Multics storage system but records are to be inserted or deleted through an input_record exit procedure.

If all of the input files are in the Multics storage system and the input_record exit is not specified, then the -file_size argument is ignored and the Sort computes the total amount of input data (using segment bit counts).

The -file_size argument is used for optimization of performance; the actual amount of input data can be considerably larger without preventing the Sort from completing. The maximum amount of data which can be sorted is (in bytes) approximately 60 million times the square root of f.

Sort/Merge Description Notes

Refer to Section IV, "Sort/Merge Description," for complete specifications for writing a Sort/Merge Description.

At the command level, only the source form of the Sort/Merge Description can be supplied. It can be either supplied as a segment or read via the I/O switch user_input (normally the user's terminal).

If the Sort/Merge Description is supplied in a segment, its pathname is specified in the -sort_desc or -merge_desc argument. The segment must be an ASCII segment; that is, an unstructured file in the Multics storage system. The segment must contain only the Sort/Merge Description.

If the Sort/Merge Description is read via the user's terminal, the -console_input argument is specified. The Sort/Merge prints "Input:" via the I/O switch user_output and waits for input. The user then types the Sort/Merge Description. To terminate the Sort/Merge Description, the user types a line consisting of a period (".") followed by a line feed. (This line is not part of the Sort/Merge Description.)

Notes

Arguments can appear in any order, but a pathname or attach description must immediately follow its keyword.

The temporary directory pathname (td_path) is the name of a directory. The Sort/Merge Description pathname (sm_path) is the name of a segment.

Any pathname may be relative (to the user's current working directory) or absolute.

SECTION III

SUBROUTINES

This section describes both the sort_ subroutine and the merge_ subroutine. The specifications for writing a Sort/Merge Description are given in Section IV, "Sort/Merge Description." Additional information necessary for executing the sort_ or merge_ subroutines with user supplied exit procedures is contained in Section V, "Exit Procedures," of this manual.

The conventions used below for describing arguments are the same as those used in the Multics Programmers' Manual, Subroutines, Order No. AG93.

Usage

```
dcl sort_ entry((*)char(*), char(*), (*)ptr, char(*), char(*), float
    bin(27), fixed bin(35));

call sort_ (input_specs, output_spec, sm_desc, temp_dir, user_out_sw,
    file_size, code);

dcl merge_ entry((*)char(*), char(*), (*)ptr, char(*), fixed bin(35));

call merge_ (input_specs, output_spec, sm_desc, user_out_sw, code);
```

where:

1. input_specs

   An array containing the specifications of the input files. Up to 10 input files may be specified. The array extent specifies the number of input files. (Input)

   Input file j is specified in the array element input_specs(j), in one of the following forms:

   -input_file pathname, -if pathname

   If an input file is in the Multics storage system and its file organization is either sequential or indexed, then it may be specified by its pathname. The file may be either a single segment or a multisegment file. The star convention cannot be used.

   An input file specified by a pathname will be attached using the attach description "vfile_ pathname".

   -input_description attach_desc, -ids attach_desc

   If an input file is not in the Multics storage system or its file organization is neither sequential nor indexed, then it must be specified by an attach description. The target I/O module specified via the attach description must support the sequential_input opening mode and the iox_ entry point read_record.

Pathnames and attach descriptions can be intermixed in the input_specs array.

For the sort_ subroutine, if the user is supplying an input_file exit procedure, then input_specs(1), the first input file specification, must be "" (the array extent should be 1) and the input_file exit procedure must be named in the io_exits structure of the Sort Description. For the merge_ subroutine, an input_file exit procedure cannot be specified.

2. output_spec

Specification of the output file. Only one output file may be specified. (Input)

The output file may be specified in one of the following forms:

-output_file pathname, -of pathname

If the output file is in the Multics storage system and its file organization is sequential, then it may be specified by its pathname. The file may be either a single segment or a multisegment file.

The equals convention can be used. If it is, it is applied to the pathname of the first input file and the first input file must be specified by a pathname, not by an attach description.

An output file specified by a pathname will be attached using the attach description "vfile_ pathname". Thus if the file does not exist, it will be created. If it does exist, it will be overwritten.

-output_file -replace, -of -rp

For the sort_ subroutine, the output file is to replace the first input file. That input file will be overwritten during the merge phase of the Sort. If -replace is specified, the first input file must be specified by a pathname, not by an attach description. For the merge_ subroutine, the -replace option cannot be specified.

-output_description attach_desc, -ods attach_desc

If the output file is not in the Multics storage system or its file organization is not sequential, then it must be specified by an attach description. The target I/O module specified via the attach description must support the sequential_output opening mode and the iox_ entry point write_record.

For the sort_ subroutine, if the user is supplying an output_file exit procedure then the output_spec argument must be "" and the output_file exit procedure must be named in the io_exits structure of the Sort Description. For the merge_ subroutine, an output_file exit procedure cannot be specified.

3. sm_desc

An array of pointers to the Sort/Merge Description. See "Sort/Merge Description Notes" below. (Input)

4. temp_dir

For the sort_ subroutine, the pathname of the directory which is to contain work files for the sorting process. The equals convention cannot be used. (Input)

For the merge_ subroutine, this argument is not present.

If the temp_dir argument is "", then work files for the Sort will be contained in the user's process directory.

This argument should be specified when the process directory will not be large enough to contain the work files for the Sort. The get_wdir_ function may be used to obtain the name of the user's current working directory.

For both the Sort and the Merge, certain small work files are always placed in the user's process directory.

5. user_out_sw

destination of both the summary report and diagnostic messages for errors detected in the subroutine arguments or in the Sort/Merge Description. (Input)

This argument may have the following values:

"" = write the summary report and diagnostic messages via the I/O switch user_output.

"-bf" = do not write the summary report and diagnostic messages. If any errors are diagnosed, sort_ or merge_ will return with the status code bad_arg but information about the number and nature of the errors is not available.

switchname = write the summary report and diagnostic messages via the I/O switch named switchname. The switch must be attached and open for stream output.

6. file_size

For the sort_ subroutine, the total amount of data to be sorted, in millions of bytes. (Input)

For the merge_ subroutine, this argument is not present.

If the file_size argument is zero, the default assumption for the Sort is approximately one million bytes (file_size = 1.0).

This argument is intended for use when some or all of the input files are not in the Multics storage system (that is, are not specified by pathnames) or when an input_file exit procedure is specified. In these cases the Sort cannot determine the amount of input data. The file_size argument may also be specified when all of the input files are in the Multics storage system but records are to be inserted or deleted through an input_record exit procedure.

If all of the input files are in the Multics storage system and the input_record exit is not specified, then the file_size argument is ignored and the Sort computes the total amount of input data (using segment bit counts).

The file_size argument is used for optimization of performance; the actual amount of data can be considerably larger without preventing the Sort from completing. The maximum amount of data which can be sorted is (in bytes) approximately 60 million times the square root of file_size.

7. code

Standard Multics status code returned by sort_ or merge_. Possible values are listed below under "Status Codes." (Output)

Status Codes

The following status codes may be returned by sort_ or merge_ (all codes are in error_table_):

0	Normal return (no errors).
bad_arg	One or more arguments specified to sort_ or merge_, including those in the Sort/Merge Description, was invalid or inconsistent. The Sort/Merge will have previously written diagnostic messages as directed by the user_out_sw argument. The sorting process itself has not been started.
fatal_error	The Sort/Merge has encountered a fatal error during the sorting or merging process. The Sort/Merge will have previously generated a specific error message and signalled the sub_error_ condition via the sub_err_ subroutine.
out_of_sequence	The call to sort_ or merge_ is not in the sequence required by the Sort/Merge; that is, sort_ or merge_ has been called after an initiation of the Sort/Merge but before termination of that invocation.

Sort/Merge Description Notes

Refer to Section IV, "Sort/Merge Description," for complete specifications for writing a Sort/Merge Description.

At the subroutine level, either the source form or the internal form of the Sort/Merge Description can be supplied.

If the source form is supplied, it must be supplied as a segment. The sm_desc argument to sort_ or merge_ must have an array extent of 1 and the one pointer must be a pointer to the segment. The segment must be an ASCII segment; that is, an unstructured file in the Multics storage system. The segment must contain only the Sort/Merge Description. The source form is advantageous when the user writes the Sort/Merge Description and supplies it to the procedure which calls sort_ or merge_.

The internal form of the Sort/Merge Description is a set of structures. For sort_, the internal form is one, two, or three structures. The sm_desc argument must have an array extent of exactly 3, and the three pointers are pointers to the three structures. For merge_, the internal form is one or two structures. The sm_desc argument must have an array extent of exactly 2, and the two pointers are pointers to the two structures. Any of the structures can be omitted; in that case the corresponding pointer must be null. The pointers must be specified in the array in the following order:

```
addr(keys)
addr(exits)
addr(io_exits)  -  for sort_ only
```

where the three structures (keys, exits, and io_exits) are defined in Section IV, "Sort/Merge Description." The internal form is advantageous when the procedure calling sort_ or merge_ constructs the Sort/Merge Description.

Notes

The temporary directory pathname (temp_dir argument) is the name of a directory.

Any pathname may be relative (to the user's current working directory) or absolute.

SECTION IV

SORT/MERGE DESCRIPTION

The Sort/Merge Description contains additional information to specialize the Sort/Merge package for a particular execution. The information supplied may be:

Keys - Description of one or more key fields used for ranking records.

Exits - Specification of which exit points are to be used and the names of the corresponding user supplied exit procedures.

A Sort/Merge Description is required. As a minimum, the user must specify how records are to be ranked, either by describing key fields or by naming a compare exit procedure. Other information in the Sort/Merge Description is optional.

The Sort/Merge Description can be supplied in either of two formats, called source form and internal form.

SOURCE FORM

The source form of the Sort/Merge Description can be used either at the command level (sort or merge commands) or at the subroutine level (sort_ or merge_ subroutines).

At the command level, the source form of the Sort/Merge Description can be supplied as a segment or can be read via the I/O switch user_input (normally the user's terminal). At the subroutine level, the source form of the Sort/Merge Description can be supplied only as a segment.

As a segment, the Sort/Merge Description must be an ASCII segment; that is, an unstructured file in the Multics storage system. The segment must contain only the Sort/Merge Description.

If the Sort/Merge Description is to be read via the user's terminal, the Sort/Merge prints "Input:" via the I/O switch user_output and waits for input. The user then types the Sort/Merge Description. To terminate the Sort/Merge Description, the user types a line consisting of a period (".") followed by a line feed. (This line is not part of the Sort/Merge Description.)

Syntax

The source form of a Sort/Merge Description consists of a set of statements. Each statement must begin with a function keyword. The function keyword is followed by the function keyword delimiter colon (":"). The statement itself consists of one or more parameters, separated by parameter delimiters. The parameter delimiters are spaces, commas (","), or (in certain specific cases as specified below) parentheses ("(" and ")"). Each statement must end with the statement delimiter semicolon (";").

In the descriptions below, certain notational conventions are used. A word enclosed between the less than and greater than symbols ("<" and ">") is a notational variable, which must be replaced by an actual word or phrase of the Sort/Merge Description language. A word not enclosed between < and > is an actual word of the Sort/Merge Description language. A phrase enclosed between brackets ("[" and "]") is optional. A phrase enclosed between braces ("{" and "}") and followed by an ellipsis ("...") is required, and may be repeated one or more times.

Keys Statement

The Keys statement specifies key fields used to rank the records of the input files. The format of the Keys statement is:

```
keys: {<key_description>} ... ;
```

The Keys statement consists of a series of one or more <key_description>s. The key descriptions are specified in order, the first describing the major key and the last describing the most minor key. Up to 32 key descriptions may be supplied.

A key description is the specification of a single key field. The format of a <key_description> is:

```
<datatype> (<size>) <position> [descending]
```

where:

1. <datatype>
   is the data type of the key field. This element is required. See Table 4-1 below for the encoding of <datatype>.

2. <size>
   is the size of the key field, expressed in a form which depends on the data type. This element is required.

   For string data types, <size> is the length (characters or bits) of the field. The length is the exact amount of space occupied by the field.

   For arithmetic data types, <size> is the precision (binary or decimal digits) of the field. Scale factor, if any, must not be written (it is not required by the Sort/Merge). The space occupied is determined by the precision in combination with the data type and the alignment. (Alignment is specified via <position>.) For an aligned binary field (fixed or floating), the space occupied is increased if necessary to an integral number of words.

<size> must be a decimal integer. The unit depends on the data type. See Table 4-1 below for the semantics of <size>. (The rules used are the same as those used by Multics PL/I.)

3. <position>
is the offset of the beginning of the key field, relative to the beginning of the record. Consider the record as being aligned on a word boundary, as will be the case for a Multics PL/I structure. This element is required. There are two formats:

<w> where <w> is the word offset. Words are numbered from 0 for the first word of the record. This format specifies that the key field is aligned on a word or (if <w> is even) on a double word boundary.

<w> (<b>) where <w> is the word portion of the offset and <b> is the bit portion of the offset; that is, the bit offset within the word. Bits are numbered from 0 to 35. This format implies that the key field is not aligned on a word boundary. If the key field is aligned on a word boundary but the user specifies a bit offset of 0 anyway, the Sort/Merge will operate correctly although speed of execution may be affected.

<w> and <b> must be expressed in decimal.

The formats for <position> and the values for <w> and <b> are consistent with those shown in Multics PL/I listings or used by debug.

4. descending, dsc
specifies descending order for ranking using this key field. This element may be omitted; the default is ascending order for this key field.

Table 4-1. Datatype Encoding and Semantics of Size (Source Form)

Data Type	Encoding of <datatype>	Semantics of <size> (where <size> = n) Unit	Range	Space Occupied
Character string (Multics ASCII)	char	9 bit character	1 - 4095	n characters
Bit string	bit	1 bit	1 - 4095	n bits
Fixed binary	bin	1 bit	1 - 71	Aligned: $1 \leq n \leq 35$: one word $36 \leq n \leq 71$: two words Unaligned: n + 1 bits
Floating binary	float bin	1 bit	1 - 63	Aligned: $1 \leq n \leq 27$: one word $36 \leq n \leq 63$: two words Unaligned: n + 9 bits
Fixed decimal (leading sign)	dec	9 bit digit	1 - 59	n + 1 digits
Floating decimal	float dec	9 bit digit	1 - 59	n + 2 digits

In addition to the forms shown for <datatype> in the table above, the following variants are also permitted:

The following alternate spellings may be used:

char|character bin|binary dec|decimal

The word "fixed" may be used (or omitted). For example:

fixed bin|bin fixed dec|dec

The words may be written in any sequence. For example:

float bin|bin float

char(10), 0(18)	Character string, Multics ASCII code, length ten characters; starts at bit 18 of word 0.
char(8), 1, descending	Character string, Multics ASCII code, length eight characters; starts at bit 0 of word 1; ranking is descending.
character(4), 2, dsc	Character string, Multics ASCII code, length four characters; starts at bit 0 of word 2; ranking is descending.
bit(16), 0(2)	Bit string, length 16 bits; starts at bit 2 of word 0.
bin(17), 2	Fixed binary, precision 17; since no bit offset is specified, is aligned and thus occupies one word (equivalent to "bin(35), 2").
bin(17), 2(18)	Fixed binary, precision 17; since a bit offset is specified, is unaligned and occupies 18 bits; starts at bit 18 of word 2 (i.e., is in the low order half of word 2).
bin(1), 2(0)	Fixed binary, precision 1; unaligned and thus occupies 2 bits; starts at bit 0 of word 2.
bin(1), 2	Fixed binary, precision 1; aligned and thus occupies one word (equivalent to "bin(35), 2").
bin(36), 2	Fixed binary, precision 36; since no bit offset is specified and precision is greater than 35 and word offset is even, is aligned and occupies two words (equivalent to "bin(71), 2").
dec(6), 0(9)	Fixed decimal, 9 bit digit, precision 6; starts at bit 9 of word 0 and occupies 7 digits including sign (that is, through the end of word 1).
float dec(9), 0(9)	Floating decimal, 9 bit digit, precision 9; starts at bit 9 of word 0 and occupies 11 digits including exponent and sign (that is, through the end of word 2).

Exits Statement

An Exits statement specifies the exit procedures to be used during execution of the Sort/Merge. The format of an Exits statement is:

```
exits:  {<exit_description>} ... ;
```

The Exits statement consists of a set of one or more <exit_description>s. Exit descriptions may be specified in any order.

An exit description is the specification of one exit point and the user supplied exit procedure to be called at that exit point. The format of an <exit_description> is:

```
<exit_name> <user_name>
```

where:

1. <exit_name>
   is the keyword naming the exit point at which the user supplied exit procedure is to be called. Exit names may be chosen from the following list:

   ```
   input_file     - for the Sort only
   output_file    - for the Sort only
   input_record   - for the Sort only
   output_record
   compare
   ```

2. <user_name>
   is the name of the entry point of the user-supplied procedure. This parameter has the same syntax and semantics as a command name. That is:

   User_name can be either a segment name (e.g., segment) or a segment name and an entry point name (e.g., segment$entry_point). In these cases, the user's current search rules are applied to find the procedure. (If some segment has already been initiated by the specified reference name, that segment is used.)

   User_name can also be a pathname; that is, can specify a directory hierarchy location, either relative (to the user's current working directory) or absolute. In this case, the search rules are not applied and the pathname is used to find the procedure. (If some other segment is already known by the specified reference name, that segment is terminated first.)

INTERNAL FORM

The internal form of the Sort/Merge Description can be used only at the subroutine level (sort_ or merge_ subroutines).

The internal form of the Sort/Merge Description is a set of structures. For sort_, the internal form is one, two, or three structures. The sm_desc argument must have an array extent of exactly 3, and the three pointers are pointers to the three structures. For merge_, the internal form is one or two structures. The sm_desc argument must have an array extent of exactly 2, and the two pointers are pointers to the two structures. Any of the structures can be omitted; in that case the corresponding pointer must be null. The pointers must be specified in the array in the following order:

```
addr(keys)
addr(exits)
addr(io_exits)  -  for sort_ only
```

where the three structures (keys, exits, and io_exits) are defined below.

keys Structure

The keys structure is used when the caller describes key fields. The standard key comparison routine of the Sort/Merge will then be used to rank records. If the caller describes keys, then the compare exit must not be specified.

If the caller does not describe keys, then the pointer to the keys structure in the array sm_desc must be null and the compare exit must be specified in the exits structure. The user supplied compare exit procedure will then be used to rank records.

The keys structure is:

```
dcl 1 keys based,
      2 version          fixed bin init(1),
      2 number           fixed bin,
      2 key_desc(user_keys_number refer(keys.number)),
        3 datatype       char(8),
        3 size           fixed bin(24),
        3 word_offset    fixed bin(18),
        3 bit_offset     fixed bin(6),
        3 desc           char(3);
```

where:

1. version
   is the version number of the structure (must be 1).

2. number
   is the number of key fields, established by the value of user_keys_number. Up to 32 key fields can be specified.

3. key_desc
   is an array of key descriptions. Each key description is one element of the array. The key descriptions must be specified in order, the major key first and the most minor key last.

4. datatype
   is the data type of the key field. See Table 4-2 below for the encoding of datatype. The value must be left justified within datatype.

5. size
   is the size of the key field, in units which depend on the data type.

   For string data types, size is the exact length (characters or bits) of the field.

   For arithmetic data types, size is the precision (binary or decimal digits) of the field. The space occupied is determined by precision in combination with the data type. The space occupied is not adjusted for an aligned field. For example, for an aligned fixed binary field of one word, size must be specified as 35; for an aligned floating binary field of two words, size must be specified as 63. See Table 4-2 below for the semantics of size.

6. word_offset
   is the word portion of the offset of the beginning of the key field, relative to the beginning of the record. Consider the record as being aligned on a word boundary, as will be the case for a Multics PL/I structure. Words are numbered from 0 for the first word of the record.

7. bit_offset
   is the bit portion of the offset of the key field; that is, the bit offset within the word in which the key field begins. Bits are numbered from 0 to 35. (If the field is aligned on a word boundary, then bit_offset is 0.)

8. desc
   indicates whether ranking for this key field is to be ascending or descending. Possible values are:

   "" = use ascending ranking.

   "dsc" = use descending ranking.

Table 4-2. Datatype Encoding and Semantics of Size (Internal Form)

Data Type	Encoding of datatype	Semantics of size (where size = n) Unit	Range	Space Occupied
Character string (Multics ASCII)	char	9 bit character	1 - 4095	n characters
Bit string	bit	1 bit	1 - 4095	n bits
Fixed binary	bin	1 bit	1 - 71	n + 1 bits
Floating binary	flbin	1 bit	1 - 63	n + 9 bits
Fixed decimal (leading sign)	dec	9 bit digit	1 - 59	n + 1 digits
Floating decimal	fldec	9 bit digit	1 - 59	n + 2 digits

exits Structure

The exits structure is:

```
dcl 1 exits,
      2 version          fixed bin init(1),
      2 compare          entry,
      2 input_record     entry,
      2 output_record    entry;
```

where:

1. version
   is the version number of the structure (must be 1).

2. compare
   specifies the entry point of a user supplied compare exit procedure. If the caller describes key fields (supplies a keys structure), then this exit must not be specified.

3. input_record
   for the sort_ subroutine, specifies the entry point of a user supplied input_record exit procedure. This exit can be specified whether or not the input_file exit is specified. For the merge_ subroutine, an input_record exit cannot be specified.

4. output_record
   specifies the entry point of a user supplied output_record exit procedure. This exit can be specified whether or not the output_file exit is specified.

io_exits Structure

The io_exits structure is:

```
dcl 1 io_exits,
      2 version          fixed bin init(1),
      2 input_file       entry,
      2 output_file      entry;
```

where:

1. version
   is the version number of the structure (must be 1).

2. input_file
   specifies the entry point of a user supplied input_file exit procedure. If the caller names input files, then this exit must not be specified.

3. output_file
   specifies the entry point of a user supplied output_file exit procedure. If the caller names the output file, then this exit must not be specified.

For the merge_ subroutine, the io_exits structure cannot be specified since neither an input_file nor an output_file exit is provided.

Entry Variables

In the exits and io_exits structures, each exit point is specified via an entry variable. The entry variable must be set (either initialized or assigned) by a user procedure, normally the procedure which calls sort_ or merge_. The entry variable can identify either an internal entry point (that is, an internal procedure) or an external entry point of the procedure which sets the entry variable; or it can identify an external entry point of another user procedure.

If none of the exits declared in either the exits or io_exits structure is to be used, then that structure can be omitted and the corresponding pointer in the array sm_desc must be null. (For the merge_ subroutine, there must not be a pointer in sm_desc for the io_exits structure.) If the structure is included but an exit specified in it is not to be used, then the corresponding entry variable must be set either to sort_$noexit, which is declared:

```
dcl  sort_$noexit  entry external;
```

or to merge_$noexit, which is declared:

```
dcl  merge_$noexit entry external;
```

An exit point may not be altered after the call to sort_ or merge_. Any change to the entry variable thereafter will have no effect. However, certain entry points can be disabled, as specified in the descriptions of the individual exit procedures.

WRITING EXIT PROCEDURES

The exit points to be used during an execution of the Sort/Merge and the names of the corresponding user supplied exit procedures are specified in the Exits statement or in the exits and io_exits structures as described above. The specifications for writing exit procedures (PL/I declare and call statements) and the functional requirements imposed upon exit procedures are given in Section V, "Exit Procedures."

SECTION V

EXIT PROCEDURES

A user supplied exit procedure is called by the Sort/Merge to perform a specified function. The user exit procedure must perform that function, and then must return to the Sort/Merge. The user exit procedure may perform additional functions desired by the user.

Certain exit procedures replace the corresponding standard routine of the Sort/Merge. Other exit procedures supplement the normal functions of the Sort/Merge. This is specified for each individual exit procedure below.

The following exit points are provided:

input_file	-	for the Sort only
output_file	-	for the Sort only
compare		
input_record	-	for the Sort only
output_record		

All exit points may be active during the same invocation of the Sort/Merge.

The entry point names of all user supplied exit procedures are defined by the user. Specific names are shown below only for convenience in discussion.

INPUT FILE EXIT PROCEDURE

Function

An input_file exit procedure replaces the standard input reading function of the Sort. The Sort calls the input_file exit procedure only once during an execution of the Sort.

For the Merge, an input_file exit procedure cannot be specified.

An input_file exit procedure must perform the following function: For each record which is input by the user to the sorting process, the input_file exit procedure must make one call to the entry sort_$release (described later in this section). After the input_file exit procedure has released the last input record to the Sort, it must return to the Sort.

Usage

```
input_file:  proc(code);

dcl  code  fixed bin(35) parameter;
```

where code is a standard Multics status code (in error_table_) which must be returned by the input_file exit procedure. If the value is not 0, then the Sort normally prints the corresponding message and returns to its caller with the status code fatal_error. (Output)

OUTPUT FILE EXIT PROCEDURE

Function

An output_file exit procedure replaces the standard output writing function of the Sort. The Sort calls the output_file exit procedure only once during an execution of the Sort.

For the Merge, an output_file exit procedure cannot be specified.

An output_file exit procedure must perform the following functions: For each record which is to be retrieved in ranked order from the Sort, the output_file exit procedure must make one call to the entry point sort_$return (described later in this section). If sort_$return is called but there are no more records to be retrieved from the sorting process, then sort_$return returns with the status code end_of_info. The output_file exit procedure then must return to the Sort. If the user desires, the output_file exit procedure may terminate retrieval at any time prior to receiving the end_of_info status, but it must still return to the Sort. (The entry sort_$return may return status codes other than end_of_info in case of error.)

Usage

```
output_file:  proc(code);

dcl  code  fixed bin(35) parameter;
```

where code is a standard Multics status code (in error_table_) which must be returned by the output_file procedure. If the value is not 0, then the Sort normally prints the corresponding message and returns to its caller with the status code fatal_error. (Output)

Function

A compare exit procedure replaces the standard key comparison procedure of the Sort/Merge. The Sort/Merge calls the compare exit procedure each time the sorting or merging process is ready to rank two records; that is, to determine which of the two is first in the sorted order.

A compare exit procedure must perform the following function: The compare exit procedure receives as arguments a pointer to each of the two records. The compare exit procedure must determine which of the two records is first - or that they are equal in rank - and must return the corresponding return value to the Sort. The compare exit procedure is invoked as a function.

Usage

```
compare:  proc(rec_ptr_1, rec_ptr_2) returns(fixed bin(1));

dcl (rec_ptr_1, rec_ptr_2) ptr parameter;
dcl  result fixed bin(1);

     ...

     return(result);
end compare;
```

where:

1. rec_ptr_1
   is a pointer to a double word aligned buffer containing the first record of the pair to be compared. This record is always the first of the two according to the original input order. (Input)

2. rec_ptr_2
   is a pointer to a double word aligned buffer containing the second record of the pair to be compared. (Input)

3. result
   is the result of the comparison. (Output) Possible values are:

   0 = the two records rank equal.

   -1 = the record pointed to by rec_ptr_1 ranks first.

   +1 = the record pointed to by rec_ptr_2 ranks first.

Notes

If a compare exit procedure requires the length of either record, it is available in the word preceding that record in the form:

```
dcl  rec_len fixed bin(21) aligned;
```

A compare exit procedure cannot alter either the content or the length of either record.

Function

An input_record exit procedure may be specified whether the Sort's standard input_file procedure or a user supplied input_file exit procedure is used, and supplements that input_file process.

For the Merge, an input_record exit procedure cannot be specified.

The Sort calls the input_record exit procedure:

1. Each time the input_file process releases a record to the Sort, and before that record is entered into the sorting process (if there were no records released to the Sort, this call is omitted);
2. Once more after the last input record has been released to the Sort (end of input);
3. Additionally, each time the input_record exit procedure returns with an action of insert.

The Sort gives the input_record exit procedure access to the current record, the record about to be entered into the sorting process.

An input_record exit procedure need not perform any processing. If it does not, then the Sort will accept the current record into the sorting process.

An input_record exit procedure may perform the following functions, which are accomplished via the values of arguments returned when the input_record exit procedure returns to the Sort:

Accept the current record. This is accomplished by setting action = 0.

Delete the current record. This is accomplished by setting action = 1.

Insert one or more records before the current record. (At the last call to the input_record exit procedure, records may be inserted at the end of input.) This is accomplished by setting rec_ptr to point to the record to be inserted, setting rec_len appropriately, and setting action = 3.

Alter the current record, before it is entered into the sorting process. This is accomplished by altering the record pointed to by rec_ptr or setting rec_ptr to point to another record, setting rec_len appropriately, and setting action = 0.

Close the exit point so that the input_record exit procedure will not be called again during this execution of the Sort. This is accomplished by setting close_exit_sw = "1".

The input_record exit procedure must return to the Sort each time it is called.

Usage

```
input_record:  proc(rec_ptr, rec_len, action, close_exit_sw);

dcl (rec_ptr        ptr,
     rec_len        fixed bin(21),
     action         fixed bin,
     close_exit_sw  bit(1) )  parameter;
```

where:

1. rec_ptr

   points to a double word aligned buffer containing the current record. The input_record exit procedure may alter the contents of the record or may change the pointer to point to another record. For the actions of accept and insert, the Sort will use the value of rec_ptr returned to it by the input_record exit procedure. (Input/Output)

   At the last call to the input_record exit procedure (either at end of input or if there were no records released to the Sort), then there is no current record and rec_ptr = null().

2. rec_len

   is the length of the current record in bytes. The input_record exit procedure may change the length of the record. For the actions of accept and insert, the Sort will use the value of rec_len returned to it by the input_record exit procedure. (Input/Output)

3. action

   indicates the action to be taken upon return to the Sort. (Input/Output)

   Arguments referred to below are the values returned to the Sort by the input_record exit procedure.

   Possible values of action are:

   0 = accept the current record. The record pointed to by rec_ptr, whose length is given by rec_len, is entered into the sorting process.

   Each time the input_record exit procedure is called, the Sort sets action to this value.

   1 = delete the current record. The current record is not entered into the sorting process.

   3 = insert a record. The record pointed to by rec_ptr, whose length is given by rec_len, is entered into the sorting process. The Sort calls the input_record exit procedure again, so that the current record may be accepted or deleted or an additional record may be inserted. At this next call to the input_record exit procedure, the current record remains the same.

   At the last call to the input_record exit procedure (end of input), if the input_record exit procedure inserts records then they are appended at the end of input. Any other value for action means do not append any records, and the input_record exit will not be taken again.

4. close_exit_sw
   indicates whether the exit is to be closed hereafter. (Input/Output)

   Possible values are:

   "0" = keep this exit open. Each time the input_record exit procedure is called, the Sort sets close_exit_sw to this value.

   "1" = close this exit. The Sort will not call the input_record exit procedure again during this execution of the Sort (even if the action is insert).

Function

An output_record exit procedure may be specified whether the standard output_file procedure of the Sort/Merge or a user supplied output_file exit procedure is used, and supplements that output_file process. The Sort/Merge calls the output_record exit procedure:

1. Each time it has determined the next record in ranked order from the merging process (if there were no records leaving the merging process, this call is omitted);

2. Once more after the last record has been obtained from the merging process (end of output);

3. Additionally, each time the output_record exit procedure returns with an action of insert.

(The term "merging process" is used here to refer either to the merge phase of the Sort or to the Merge function.)

The Sort/Merge gives the output_record exit procedure access to two records:

1. The output record, about to be written to the output file. (If an output_file exit procedure has been specified by the user, this is the record about to be returned to that exit procedure.)

2. The next record, the record leaving the merging process.

An output_record exit procedure need not perform any processing. If it does not, then the output record is accepted for the output file.

An output_record exit procedure may perform the following functions, which are accomplished via the values of arguments returned when the output_record exit procedure returns to the Sort/Merge:

Accept the output record. This is accomplished by setting action = 0.

Delete the output record. This is accomplished by setting action = 1.

Delete the record leaving the merging process. This is accomplished by setting action = 2.

Insert one or more records after the output record. (At the first call to the output_record exit procedure, records may be inserted at the beginning of output. At the last call to the output_record exit procedure, records may be inserted at the end of output.) This is accomplished by setting rec_ptr_2 to point to the record to be inserted, setting rec_len_2 appropriately, and setting action = 3.

Alter the output record, before it is written to the output file. This is accomplished by altering the record pointed to by rec_ptr_1 or setting rec_ptr_1 to point to another record, setting rec_len_1 appropriately, and setting action = 0 to accept (or action = 3 to insert).

Summarize data into the first record of a sequence of records with equal keys, and delete the succeeding records of the sequence. This may be accomplished as follows: At the first call to the output_record exit

procedure, set equal key checking on (equal_key_sw = "1"). At subsequent calls to the output_record exit procedure, if the output record and the record leaving the merging process have equal keys (equal_key = 0), then accumulate data into the output record and delete the record leaving the merging process (action = 2). If the two records have unequal keys (equal_key ≠ 0), then accept the output record (action = 0).

Summarize data into the last record of a sequence with equal keys, and delete the preceding records of the sequence. This may be accomplished as follows: At the first call to the output_record exit procedure, set equal key checking on. At subsequent calls, if the two records have equal keys then accumulate data into a work area and delete the output record (action = 1). If the two records have unequal keys, then alter the output record using the accumulated data and accept that record (action = 0).

Sequence check the output file. This is accomplished by setting seq_check_sw = "1". If the output record will not collate properly with the output file, or does not have its keys in the position specified to the Sort/Merge, then set seq_check_sw = "0".

Close the exit point so that the output_record exit procedure will not be called again during this execution of the Sort/Merge. This is accomplished by setting close_exit_sw = "1".

The output_record exit procedure must return to the Sort/Merge each time it is called.

Usage

```
output_record: proc(rec_ptr_1, rec_len_1, rec_ptr_2, rec_len_2,
                    action, equal_key, equal_key_sw,
                    seq_check_sw, close_exit_sw);

dcl (rec_ptr_1      ptr,
     rec_len_1      fixed bin(21),
     rec_ptr_2      ptr,
     rec_len_2      fixed bin(21),
     action         fixed bin,
     equal_key      fixed bin(1),
     equal_key_sw   bit(1),
     seq_check_sw   bit(1),
     close_exit_sw  bit(1) ) parameter;
```

where:

1. rec_ptr_1

   points to a double word aligned buffer containing the output record. The output_record exit procedure may alter the contents of this record or may change the pointer to point to another record. The Sort/Merge uses the value of rec_ptr_1 returned to it by the output_record exit procedure as specified below in the description of the action argument. (Input/Output)

   At the first call to the output_record exit procedure (beginning of output) or if there were no records merged, then there is no output record and rec_ptr_1 = null().

2. rec_len_1

   is the length of the output record in bytes. The output_record exit procedure may change the length of this record. The Sort/Merge uses the value of rec_len_1 returned to it by the output_record exit procedure as specified below in the description of the action argument. (Input/Output)

3. rec_ptr_2

   points to a double word aligned buffer containing the record leaving the merging process. The output_record exit procedure may not alter the contents of this record. For all actions except insert, the Sort/Merge will ignore the value of rec_ptr_2 returned to it by the output_record exit procedure. If the action is insert, then the output_record exit procedure must change rec_ptr_2 to point to the record to be inserted. (Input/Output)

   At the last call to the output_record exit procedure (end of output) or if there were no records merged, then there is no record leaving the merging process and rec_ptr_2 = null().

4. rec_len_2

   is the length of the record leaving the merging process. The output_record exit procedure may not change the length of this record. For all actions except insert, the Sort/Merge will ignore the value of rec_len_2 returned to it by the output_record exit procedure. If the action is insert, then the output_record exit procedure must set rec_len_2 to the length of the record to be inserted. (Input/Output)

5. action

   indicates the action to be taken upon return to the Sort/Merge. (Input/Output)

   Possible values of action are:

   0 = accept the output record. The output record is written to the output file. The Sort/Merge uses the returned values of rec_ptr_1 and rec_len_1 to identify the record to be written. At the next call to the output_record exit procedure, the record leaving the merging process becomes the new output record, and a new record leaving the merging process has been obtained.

   Each time the output_record exit procedure is called, the Sort/Merge sets action to this value.

   1 = delete the output record. No record is written to the output file. The Sort/Merge ignores the returned values of rec_ptr_1 and rec_len_1. At the next call to the output_record exit procedure, the record leaving the merging process becomes the new output record, and a new record leaving the merging process has been obtained.

   2 = delete the record leaving the merging process. (This action should be used for summarization into the output record.) No record is written to the output file. At the next call to the output_record exit procedure, the output record remains the same, and a new record leaving the merging process has been obtained. The Sort/Merge uses the returned values of rec_ptr_1 and rec_len_1 to identify the output record for that next call to the output_record exit procedure.

   3 = insert a record after the output record. The output record is written to the output file. The Sort/Merge uses the returned values of rec_ptr_1 and rec_len_1 to identify the record to be written. The Sort/Merge calls the output_record exit procedure again, so that the inserted record may be accepted or an additional record may be inserted. At this next call to the output_record exit procedure, the inserted record becomes the new output record, and the record leaving the merging process remains the same. The Sort/Merge uses the returned values of rec_ptr_2 and rec_len_2 to identify the inserted record.

At the last call to the output_record exit procedure (end of output), if the output_record exit procedure inserts records then they are appended at the end of output. Any other value for action means do not append any records, and the output_record exit will not be taken again.

6. equal_key

indicates whether the output record and the record leaving the merging process have equal keys. (Input)

Possible values are:

0 = the two records rank equal.

±1 = the two records do not rank equal. At the first and last calls to the output_record exit procedure (beginning of input and end of input), only one record is present and the Sort/Merge sets equal_key to this value.

If the user supplied key descriptions, then the value of equal_key is determined only by those key fields; the original input order of the two records is not used to resolve key equality. If the user supplied a compare exit procedure, then the Sort/Merge uses the result of that compare exit procedure to set the value of equal_key. (In either case, if the two records rank equal then rec_ptr_1 points to the record which is first according to the original input order of the two records.)

7. equal_key_sw

indicates whether or not equal key checking is to be performed. (Input/Output)

Possible values are:

"0" = do not check for equal keys. At the first call to the output_record exit procedure (beginning of output), the Sort/Merge sets equal_key_sw to this value.

"1" = check for equal keys before the next call to the output_record exit procedure.

Since equal key checking takes time, the user should set equal_key_sw = "1" only when required for actions such as summarization.

8. seq_check_sw

indicates whether or not sequence checking is to be performed. (Input/Output)

Possible values are:

"0" = do not sequence check.

"1" = sequence check. At the first call to the output_record exit procedure (beginning of output), the Sort/Merge sets seq_check_sw to this value.

Sequence checking means comparing the output record to the record previously written to the output file. (If the user specified an output_file exit procedure, the output record is compared to the record previously returned to that exit procedure.) Sequence checking is performed after the output_record exit procedure returns to the Sort/Merge, and only if a record is to be written to the output file (that is, only if the action is accept or insert). If the user supplied key descriptions, then the standard key comparison routine of the Sort/Merge is used. If the user supplied a compare exit procedure, then that exit procedure is called.

If the output record is out of sequence with the previous record, then the status code fatal_error is returned to the caller of sort_; see the specifications of the sort_ and merge_ subroutines in Section III, "Subroutines" above. (If the user specified an output_file exit procedure, then the status code data_seq_error is returned to that exit procedure; see the sort_$return entry below.)

All records written to the output file, including inserted records, can be sequence checked.

9. close_exit_sw
indicates whether the exit is to be closed hereafter. (Input/Output)

Possible values are:

"0" = keep this exit open. Each time the output_record exit procedure is called, the Sort/Merge sets close_exit_sw to this value.

"1" = close this exit. The Sort/Merge will not call the output_record exit procedure again during this execution of the Sort/Merge (even if the action is insert).

NOTES ON EXIT PROCEDURES

Record Areas and Pointers

Record areas used by an input_record or output_record exit procedure must be declared as static, not automatic. Also, such areas cannot be shared with input_file or output_file exit procedures.

Since the Sort/Merge aligns each record in a buffer on a double word boundary, if an exit procedure applies a based declaration of the record to the pointer(s) then correct alignment is ensured.

Original Input Order (FIFO)

For the compare and output_record exit procedures, rec_ptr_1 always points to the record whose original input order was prior to the record pointed to by rec_ptr_2. If a compare exit procedure returns with an equal ranking for the two records, then this original input order is preserved. Original input order has been defined above under "Key Fields" in Section I.

ENTRY: sort_$release

Function

The sort_$release entry is called each time the user releases a record to the sorting process. Calls to sort_$release are made from a user supplied input_file exit procedure. The caller specifies the location and length of the record. The Sort accepts the record and stores it in its own work area.

The sort_$release entry does not apply to the Merge, since for the Merge an input_file exit procedure cannot be specified.

Usage

```
dcl  sort_$release entry(ptr, fixed bin(21), fixed bin(35));

call sort_$release (buff_ptr, rec_len, code);
```

where:

1. buff_ptr
   is a pointer to a byte aligned buffer containing the record. (Input)

2. rec_len
   is the length of the record in bytes. (Input)

3. code
   is a standard Multics status code returned by the Sort. Possible values are listed below under "Status Codes." (Output)

Status Codes

The following status codes may be returned by the sort_$release entry point (all codes are in error_table_):

0	Normal return (no error).
out_of_sequence	The call to sort_$release is not in the sequence required by the Sort; e.g., sort_$release has been called before sort_.
fatal_error	The Sort has encountered a fatal error during the sorting process. The Sort will have previously generated a specific error message and signalled the sub_error_ condition via the sub_err_ subroutine.
long_record	This input record is longer than the maximum supported. The record is ignored by the Sort, and the caller may continue to release records to the Sort.
short_record	This input record is shorter than the minimum required to contain the key fields. The record is ignored by the Sort, and the caller may continue to release records to the Sort.

ENTRY: sort_$return

Function

The sort_$return entry is called each time the user retrieves an output record, in ranked order, from the Sort. Calls to sort_$return are made from a user supplied output_file exit procedure. Upon return from sort_$return, the caller is given the location and length of the record.

If sort_$return is called but there are no more records to be retrieved, then sort_$return returns to the caller with the status code end_of_info.

The sort_$return entry does not apply to the Merge, since for the merge an output_file exit procedure cannot be specified.

Usage

```
dcl sort_$return entry(ptr, fixed bin(21), fixed bin(35));

call sort_$return (buff_ptr, rec_len, code);
```

where:

1. buff_ptr
   is a pointer to a double word aligned buffer containing the record. (Output)
2. rec_len
   is the length of the record in bytes. (Output)
3. code
   is a standard Multics status code returned by the Sort. Possible values are listed below under "Status Codes." (Output)

Notes

The Sort aligns each record on a double word boundary in a work area. Thus if the caller applies a based declaration of the record to the pointer then correct alignment is ensured.

Status Codes

The following status codes may be returned by the sort_$return entry point (all codes are in error_table_):

0	Normal return (not end of information, no error).
end_of_info	There are no more records to be retrieved from the Sort. This is the normal end of data indication. No record is returned to the caller.

out_of_sequence	The call to sort_$return is not in the sequence required by the Sort; e.g., sort_$return has been called before sort_$release.
fatal_error	The Sort has encountered a fatal error during the sorting process. The Sort will have previously generated a specific error message and signalled the sub_error_ condition via the sub_err_ subroutine.
data_loss	End of data has been reached, but the number of records previously returned is less than the number of records released to the Sort. No record is returned to the caller.
data_gain	The number of records returned (including this record) is now larger than the number of records released to the Sort. The current record is returned to the caller, and the caller may continue to retrieve records from the Sort.
data_seq_error	A ranking error has occurred in the records returned to the caller; that is, the current record is out of order. The current record is returned to the caller, and the caller may continue to request records from the Sort.

SECTION VI

EXAMPLES

EXAMPLES OF COMMAND LEVEL

```
sort  -input_file sort.in  -output_file =.out  -console_input
Input.
key:  char(10), 0;
.
```

In this example, the arguments of the command state that there is one input file, whose pathname is sort.in; the output file pathname is sort.out; the Sort Description is input via the user's terminal; and by default the work files are contained in the user's process directory.

The Sort Description states that there is one key, a character string of length 10 characters, starting at word 0 bit 0 of the record. There are no exits specified.

```
sort  -temp_dir >udd>pool  -sort_desc sd
```

In this example the arguments of the command state that the work files are contained in the directory >udd>pool; and the Sort Description is contained in the segment named sd.

Assume that the segment sd contains:

```
keys:   fixed bin(35) 0, char(8) 1;
exits:  input_file  user$input,
        output_file user$output;
```

The Sort Description states that there are two keys. The major key is an aligned fixed binary field of precision 35, contained in word 0 of the record. The minor key is a character string of length 8, contained in words 1 and 2 of the record.

There are two exits, an input_file procedure exit and an output_file procedure exit. The input_file exit procedure entry point is named user$input; the output_file exit procedure entry point is named user$output. These exits must be specified because the command did not specify either an input file or an output file.

```
sort  -if sort_in  -of -replace  -td [wd]  -sd sort_desc
```

In this example the arguments of the command state that the input file is named sort_in; the output file is to replace the input file; work files are contained in the user's current working directory; and the Sort Description is contained in the segment sort_desc.

```
sort -input_description "tape_ansi_ vol_1 -name a" -if b  \
     -output_description "vfile_ c -extend" -ci
```

In this example there are two input files. The first input file is specified by an attach description for the I/O module tape_ansi_ with the attach argument "vol_1 -name a". The second input file is specified by the pathname b, and thus must be a sequential or indexed file in the storage system. The output file is specified by an attach description for the I/O module vfile_ with the attach argument "b -extend". For the I/O module vfile_, this means that the pathname is c and the file is to be extended; that is, output records from the Sort will be written at the end of the file c (if it already exists).

(A \ followed by a line feed is used to continue the command arguments onto the second line.)

The Sort Description (not shown) will be read via the user's terminal.

```
merge  -input_file in_1 -if in_2  -of out_1  -merge_desc md
```

In this example, the arguments of the command state that the input files are named in_1 and in_2; the output file is named out_1; and the Merge Description is contained in the segment named md.

Assume that the segment md contains:

```
exits:  compare  user$compare,
        output_record  user$output;
```

There are two exits, a compare procedure exit and an output_record procedure exit.

```
merge  -ids "record_stream_ -target vfile_ a"  \
       -ids "syn_ user_switchname"  -of c  -console_input
```

In this example, assume that the first input file is an unstructured file in the storage system, with the pathname a. This input file has been specified by an attach description using the I/O module record_stream_, which will transform the record I/O operations requested by the Merge into the appropriate stream I/O operations for the target file a. The second input file is attached using the I/O module syn_ to the I/O switch user_switchname, which must be attached and closed.

EXAMPLE OF SUBROUTINE LEVEL

```
call sort_(input_specs, "-of =.out", sm_desc, "",
          "", 2.5, code);


dcl  input_specs(2) char(16) init("-if a.in", "-ids syn_ sw"),
     sm_desc(3) ptr init(addr(keys, addr(exits, null()),
     code fixed bin(35);


dcl 1 keys,
      2 version       fixed bin init(1),
      2 number        fixed bin init(1),
      2 key_desc(1),
        3 datatype    char(8) init("char"),
        3 size        fixed bin(24) init(7),
        3 word_offset fixed bin(18) init(1),
        3 bit_offset  fixed bin(6) init(0),
        3 desc        char(3) init("dsc");


dcl 1 exits,
      2 version       fixed bin init(1),
      2 compare       entry init(sort_$noexit),
      2 input_record  entry init(sort_$noexit),
      2 output_record entry init(summarize_into_first);
```

In this example, there are two input files. The first has the pathname a.in; the second is attached through the I/O module syn_ to the I/O switch sw, which must be attached and closed. The output file will have the pathname a.out.

The Sort Description is supplied in internal form. The keys and exits structures are present; the io_exits structure is omitted.

The keys structure describes one key, a character string of length 7 characters starting at bit o in word 1 (the second word) of the record. Ranking is descending.

The exits structure specifies only an output_record exit procedure, whose entry point is summarize_into_first.

Assume the output_record exit procedure is:

```
summarize_into_first:  proc(rec_ptr_1, rec_len_1, rec_ptr_2, rec_len_2,
      action, equal_key, equal_key_sw, seq_check_sw, close_exit_sw);
dcl (rec_ptr_1     ptr,
     rec_len_1     fixed bin(21),
     rec_ptr_2     ptr,
     rec_len_2     fixed bin(21),
     action fixed  bin,
     equal_key     fixed bin(1),
     equal_key_sw  bit(1),
     seq_check_sw  bit(1),
     close_exit_sw bit(1) ) parameter;
dcl 1 record based,
      2 data fixed bin(35),
      2 key  char(7),
      2 rest char(69);
if rec_ptr_1 = null() & rec_ptr_2 = null() then do;
     /*  no data in file  */
     end;
else if rec_ptr_1 = null() then do;
     /*  beginning of file  */
     equal_key_sw = "1"b;  /*  check for duplicates  */
     end;
else if rec_ptr_2 = null() then do;
     /*  end of file  */
     end;
else do;
     if equal_key = 0 then do;   /*  duplicate key  */
          rec_ptr_1->data = rec_ptr_1->data + rec_ptr_2->data;
          action = 2;
          end;
     end;
end summarize_into_first;
```

This output_record exit procedure retains only the first record from each group of consecutive records with equal keys, accumulating into the retained record the value of the field data from the deleted records.

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

CUT ALONG LINE

TITLE: SERIES 60 (LEVEL 68) MULTICS SORT/MERGE

ORDER NO. AW32, REV. 0

DATED JULY 1976

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

CUT ALONG LINE

FOLD ALONG LINE

FOLD ALONG LINE

Honeywell

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Shepperd Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

21484, 3C878, Printed in U.S.A.

AW32, Rev. 0