MULTICS TECHNICAL BULLETIN

MTB-253

To: Distribution

From: J. Berson

Date: 02/03/76

Subject: Release 2 of the Multics Sort/Merge

Attached is information about Release 2 of the Multics Sort/Merge, which is scheduled for Multics Release 4.0 in June 1976. There are four write-ups, including sort command, merge command, sort_ subroutine, merge_ subroutine, in the usual form for the Multics Programmers' Manual, and one write-up of additional interfaces to be documented in the PLM.

Comments and criticisms are solicited, whether on technical aspects or on the documentation. They may be sent to Joel Berson at Honeywell Billerica by mail or phone; or via "mail Berson MSORT" on either the MIT or Phoenix Multics systems.

---

1. NEW FUNCTIONS

1.1 A Merge, or file collation, function has been added.

1.2 A subroutine interface for both the Sort and Merge has been added.

1.3 Support for the SORT portion of the ANSI COBOL Sort/Merge Module, Level 2, has been added. (The COBOL MERGE function is not supported by this package).

1.4 Additional data types for keys and multiple key fields are supported. Release 1 supported only character string and a single key field.

1.5 Additional storage media and file organizations are supported for the input and otuput files. Essentially any file can be supported which can be read or written sequentially via iox_ using any available I/O module. Release 1 supported only sequential input and output files in the Multics storage system (using vfile_).

1.6 The following additional user exit points are provided:

input_record exit:	Permits the user to alter, delete, or insert records before they enter the sorting or merging process.
output_record exit:	Permits the user to alter, delete, insert, or summarize records coming out of the sorting or merging process before they are written to the output file.

1.7 Sequence checking for output records has been added.

1.8 A file size argument has been added.

1.9 Command arguments for measurement and testing have been added (-time, -merge_order, and -string_size).

2. CHANGED SPECIFICATIONS

The keyword -sort_desc (-sd) must precede the pathname of the Sort Description (when the Sort Description is supplied in a segment). In Release 1, the pathname of the Sort Description must be the first argument and is not preceded by a keyword.

3. QUESTIONS ABOUT DOCUMENTATION

I would like to raise the following questions about documentation of the Sort/Merge.

3.1 Should the Sort and the Merge be documented in four separate MPM write-ups, as attached; or should the Merge (command and subroutine) be documented in two shorter write-ups which then refer to the two Sort write-ups for details? There is much in common between the Sort and the Merge. On the other hand, noting differences applicable to the Merge in the Sort write-ups may be somewhat complicated and confusing.

3.2 Should there be a separate Users' Guide for the Sort/Merge? If so, what information should go in the MPM and what in the Users' Guide? Some information not presently in the MPM write-ups which might go into a Users' Guide is:

text of error messages

description of the report produced by the Sort/Merge (various counts of records processed; data produced by the -time argument)

I/O usage; e.g. for PL/I I/O, Fortran, record_stream_, syn_, etc.

Relationship between file size, work space required, optimization, etc.

3.3 Should the additional command arguments described in the PLM write-up be documented directly in the MPM Commands write-ups?

sort

Name: sort

The sort command provides a generalized file sorting capability, which is specialized for execution by user supplied parameters. 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 file of ordered (or "ranked") records. The Sort 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;

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 process.

In addition to arguments to the sort command, other information is necessary to specialize the Sort for a particular execution. This information, called the Sort Description, can be supplied either through the user's terminal or in a segment.

The description given here of the sort command is sufficient for situations where the Sort is free standing; that is, where no user supplied procedures are executed. (User supplied procedures are called "exit procedures".) Additional information is necessary for executing the sort command with exit procedures, and is contained in the description of the sort_ subroutine in the Multics Programmers' Manual, Subroutines, Section II.

INPUT AND OUTPUT

The user can specify the input and output files. In this environment, the Sort 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, the user can supply either an input_file procedure or an output_file procedure (or both). An input_file procedure is responsible for reading input and releasing records

to the Sort. An output_file procedure is responsible for retrieving records (ranked by the Sort) from the Sort and writing output.

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

KEY FIELDS

The user can specify the key fields to be used in ranking records. Key fields are described in the Keys statement of the Sort Description. Up to 20 key fields may be specified. Any PL/I string or numeric data type - except 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.

Alternatively, the user can specify a user supplied compare procedure, which is then used to rank records.

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 line 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 provides exits to user supplied procedures at specific points during the sorting process. Exit procedures are named in the Exits statement of the Sort Description. The following exit points are provided:

input_file	To obtain input records and release them one by one to the sorting process.
output_file	To retrieve ranked records one by one from the sorting process and output them.
input_record	To perform special processing for each input record, such as deleting, inserting, or altering records to be input to the Sort.

output_record	To perform special processing for each output record, such as deleting, inserting, or altering records to be output from the Sort; or summarizing data by accumulating it into a summary record.
compare	To compare two records; that is, to rank them for the sorting process.

Usage

sort -input_specs output_spec control_args

where:

1. input_specs indicates that 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 ffile 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.

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.

2. output_spec Indicates that 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

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 used, the first input file must be specified by a pathname, not by an attach description.

-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.

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.

3. control_args must be chosen from the following:

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

-sort_desc sd_path
-sd sd_path
indicates that the user is specifying the pathname of the segment containing the Sort Description.

Either the -console_input or the -sort_desc argument - but not both - must be specified. See the heading Sort Description below.

-temp_dir td_path
-td td_path
indicates that the user is specifying the pathname of the directory which will contain the Sort's work files. The equals convention can not be used.

If this argument is omitted, work files will be contained in the user's process directory.

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

-file_size f
specifies that the total amount of data to be sorted is f millions of bytes. The argument f must be a decimal number. If the -file_size argument is omitted, the default assumption 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 storage system (that is, are not specified by pathnames) or when an input_file exit procedure is used. In these cases the Sort cannot determine the amount of input data. (The Sort does compute the total amount of input data which is in the storage system, using segment bit counts.) The

-file_size argument may also be used when all of the input files are in the storage system but records are to be inserted or deleted through an input_record exit procedure.

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.

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 Description pathname (sd_path) is the name of a segment.

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

Sort Description

The Sort Description contains additional information to specialize the Sort 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 Description is required. As a minimum, the user must specify how records are to be ranked, either by describing key fields in the Keys statement or by naming a compare exit procedure in the Exits statement. Other information in the Sort Description is optional.

The Sort Description may be supplied as a segment or read via the I/O switch user_input (normally the user's terminal).

If the Sort Description is supplied in a segment, its pathname is specified in the -sort_desc argument.

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

SYNTAX OF THE SORT DESCRIPTION

A Sort 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 Description language. A word not enclosed between < and > is an actual word

of the Sort 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 20 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 the table 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). 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 the table 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 to the Sort 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 will operate correctly although speed of execution may be affected.

   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.

DATATYPE ENCODING AND SEMANTICS OF SIZE

	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 word; 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 word; 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

EXAMPLES OF KEY DESCRIPTIONS

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. 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
   output_file
   input_record
   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 is already known 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.)

WRITING EXIT PROCEDURES

The exit points to be used during an execution of the Sort and the names of the corresponding user supplied exit procedures are specified in the Exits statement 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 the description of the sort_ subroutine in Section II of MPM Subroutines.

Examples

```
sort -input_file sort.in -output_file =.out -console_input
Input.
key1 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.

```
sort  -ids "record_stream_ -target vfile_ a"  -of b  -ci
```

In this example assume that the input file is an unstructured file in the storage system, with the pathname a. The 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 Sort into the appropriate stream I/O operations for the target file a.

```
sort  -ids "syn_ user_switchname"  -of b  -ci
```

In this example the input file is attached using the I/O module syn_ to the I/O switch user_switchname, which must be attached and closed.

Name: merge

The merge command provides a generalized file merging capability, which is specialized for execution by user supplied parameters. 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 records according to the values of those key fields, and write a single file of ordered (or "ranked") records. The Merge has the following general capabilities:

(END)

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

Very large files, such as multisegment files, can be 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 merging process.

In addition to arguments to the merge command, other information is necessary to specialize the Merge for a particular execution. This information, called the Merge Description, can be supplied either through the user's terminal or in a segment.

The description given here of the merge command is sufficient for situations where the Merge is free standing; that is, where no user supplied procedures are executed. (User supplied procedures are called "exit procedures".) Additional information is necessary for executing the merge command with exit procedures, and is contained in the description of the merge_ subroutine in the Multics Programmers' Manual, Subroutines, Section II.

INPUT AND OUTPUT

The user specifies the input and output files. The 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.

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

KEY FIELDS

The user can specify the key fields to be used in ranking records. Key fields are described in the Keys statement of the Merge Description. Up to 20 key fields may be specified. Any PL/I string or numeric data type - except complex or pictured - may be specified for a given key field. Ranking may be

(END)

ascending, descending, or mixed. For a character string field, the collating sequence is that of the Multics standard character set. 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. The records of each input file must be in order according to the algorithm of that 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 line 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 Merge provides exits to user supplied procedures at specific points during the merging process. Exit procedures are named in the Exits statement of the Merge Description. The following exit points are provided:

output_record — To perform special processing for each output record, such as deleting, inserting, or altering records to be output from the Merge; or summarizing data by accumulating it into a summary record.

compare — To compare two records; that is, to rank them for the merging process.

(END)

Usage

merge input_specs output_spec control_args

where:

1. input_specs indicates that 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"
-ios "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.

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

-output_file pathname
-of pathname
If the output file is in the Multics

(END)

merge

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_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.

3. control_args must be chosen from the following:

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

-merge_desc md_path
-md md_path
indicates that the user is specifying the pathname of the segment containing the Merge Description.

Either the -console_input or the -merge_desc argument - but not both - must be specified. See the heading Merge Description below.

(END)

NOTES

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

The Merge Description pathname (md_path) is the name of a segment.

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

(END)

Merge Description

The Merge Description contains additional information to specialize the Merge 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 Merge Description is required. As a minimum, the user must specify how records are to be ranked, either by describing key fields in the Keys statement or by naming a compare exit procedure in the Exits statement. Other information in the Merge Description is optional.

The Merge Description may be supplied as a segment or read via the I/O switch user_input (normally the user's terminal).

If the Merge Description is supplied in a segment, its pathname is specified in the -merge_desc argument.

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

SYNTAX OF THE MERGE DESCRIPTION

A 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

(END)

replaced by an actual word or phrase of the Merge Description language. A word not enclosed between < and > is an actual word of the 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 20 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 the table below for the encoding of <datatype>.

2. <size> is the size of the key field. 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 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

(END)

increased if necessary to an integral number of words.

<size> must be a decimal integer. The unit depends on the data type. See the table 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 to the Merge 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 Merge will operate correctly although speed of execution may be affected.

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

(END)

DATATYPE ENCODING AND SEMANTICS OF SIZE

	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 ≤ n ≤ 35: one word; 36 ≤ n ≤ 71: two words; Unaligned: n + 1 bits
Floating binary	float bin	1 bit	1 - 63	Aligned: 1 ≤ n ≤ 27: one word; 36 ≤ n ≤ 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:

(END)

merge

float binlbin float

(END)

merge

EXAMPLES OF KEY DESCRIPTIONS

`char(10), 0(18)` Character string, Multics ASCII code, length ten characters; starts at bit 18 of word 0.

`char(8), 0, descending` Character string, Multics ASCII code, length eight characters; starts at bit 0 of word 0; ranking is descending.

`character(4), 0, dsc` Character string, Multics ASCII code, length four characters; starts at bit 0 of word 0; 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

(END)

merge

digits including exponent and sign (that is, through the end of word 2).

(END)

EXITS STATEMENT

An Exits statement specifies the exit procedures to be used during execution of the 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:

   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 is already known 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

(END)

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.)

WRITING EXIT PROCEDURES

The exit points to be used during an execution of the Merge and the names of the corresponding user supplied exit procedures are specified in the Exits statement 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 the description of the merge_ subroutine in Section II of MPM Subroutines.

(END)

Examples

```
merge  -if merge.in_1 -if merge.in_2  -output_file =.out  -ci
Input.
key:  char(10), 0;
.
```

In this example, the arguments of the command state that there are two input files, whose pathnames are merge.in_1 and merge.in_2; the output file pathname is merge.out; and the Merge Description is input via the user's terminal.

The Merge 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.

```
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:

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

The Merge 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 is one exit, an output_record procedure exit; the output_record exit procedure entry point is named user$output.

```
merge -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 "c -extend". For the I/O module

(END)

vfile_, this means that the pathname is c and the file is to be extended; that is, output records from the Merge 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 Merge Description (not shown) will be read from the user's terminal.

```
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.

Name: sort_

The sort_ subroutine provides a generalized file sorting capability, which is specialized for execution by user supplied parameters. The basic function of 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 sort_ subroutine 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;

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 process.

The arguments to the sort_ subroutine include one or more pointers to additional information necessary to specialize sort_ for execution. This additional information is called the Sort Description.

(END)

INPUT AND OUTPUT

The user can specify the input and output files. In this environment, the Sort 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, the user can supply either an input_file procedure or an output_file procedure (or both). An input_file procedure is responsible for reading input and releasing records to the Sort. An output_file procedure is responsible for retrieving records (ranked by the Sort) from the Sort and writing output.

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

KEY FIELDS

The user can specify the key fields to be used in ranking records. Key fields are described in the Keys statement - or in the keys structure - of the Sort Description. Up to 20 key fields may be specified. Any PL/I string or numeric data type - except complex or pictured - may be specified for a given key field. Ranking may be ascending, descending, or mixed. For a character string key field, the collating sequence is that of the Multics standard character set.

Alternatively, the user can supply a compare procedure, which is then used to rank records.

The original input 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 list of input files (in the input_specs 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.

(END)

EXITS

The Sort provides exits to user supplied procedures at specific points during the sorting process. Exit procedures are named in the Exits statement - or in the exits and io_exits structures - of the Sort Description. The following exit points are provided:

input_file	To obtain input records and release them one by one to the sorting process.
output_file	To retrieve ranked records one by one from the sorting process and output them.
input_record	To perform special processing for each input record, such as deleting, inserting, or altering records to be input to the Sort.
output_record	To perform special processing for each output record, such as deleting, inserting, or altering records to be output from the Sort; or summarizing data by accumulating it into a summary record.
compare	To compare two records; that is, to rank them for the sorting process.

Details of exit procedures are given below under the heading Writing Exit Procedures.

(END)

Usage:

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

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

where:

1. input_specs — is 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 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 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.

   If the user is supplying an input_file exit procedure, then input_specs(1), the first

(END)

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.

2. output_spec — is the 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

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 used, the first input file must be specified by a pathname, not by an attach description.

-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

(END)

attach description must support the sequential_output opening mode and the iox_ entry point write_record.

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.

3. sort_desc — is an array of pointers to the Sort Description. See the heading Sort Description below. (Input)

4. temp_dir — is the pathname of the directory which will contain the Sort's work files. (Input)

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

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

5. user_out_sw — specifies the destination of both the summary report and diagnostic messages for errors detected in the arguments to sort_ or in the Sort 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_ 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

(END)

switch must be attached and open for stream output.

6. file_size — is the total amount of data to be sorted, in millions of bytes. If this argument is zero, the default assumption is approximately one million bytes (file_size = 1.0). (Input)

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

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 — is a standard Multics status code returned by sort_. Possible values are listed below under the heading Status Codes. (Output)

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.

STATUS CODES

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

(END)

0	Normal return (no errors).
bad_arg	One or more arguments specified to sort_, including those in the Sort Description, was invalid or inconsistent. The Sort 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 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.
out_of_sequence	The call to sort_ is not in the sequence required by the Sort; that is, sort_ has been called after initiation of the Sort but before termination of that invocation.

(END)

Sort Description

The Sort Description contains additional information to specialize the Sort for a particular execution. The Sort Description is specified via the sort_desc argument to sort_. The information specified 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 Description is required. As a minimum, the user must specify how records are to be ranked, either by describing key fields in the Keys statement or by naming a compare exit procedure in the Exits statement. Other information in the Sort Description is optional.

The Sort Description may be supplied to sort_ in either of two forms, called source form and internal form.

The source form of the Sort Description is written exactly as specified for the sort command (see the Multics Programmers' Manual, Commands and Active Functions, Section III), and is stored as an ASCII segment; that is, as an unstructured file in the Multics storage system. If source form is used, then the sort_desc argument to sort_ must have an array extent of 1 and the one pointer must be a pointer to the segment. (The segment must contain only the Sort Description.) The source form is useful when the user writes the Sort Description and supplies it to the procedure which calls sort_.

The internal form of the Sort Description is a set of one to three structures. The sort_desc argument must have an array extent of 3, and the three pointers are pointers to the three 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)
```

where the three structures (keys, exits, and io_exits) are defined below. The internal form is useful when the procedure calling sort_ constructs the Sort Description.

(END)

KEYS STRUCTURE

The keys structure is used when the caller describes key fields. The Sort's standard compare routine 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 corresponding pointer in the array sort_desc must be null and the compare exit must be specified in the exits structure. The user supplied compare routine will then be used to rank records.

The keys structure is:

```
dcl 1 keys,
      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.

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 the table 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.

(END)

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 the table 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.

(END)

DATATYPE ENCODING AND SEMANTICS OF SIZE

	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

(END)

EXITS STRUCTURE

The exits structure is:

```
dcl 1 exits,
      1 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 — 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.

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).

(END)

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.

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_. 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 sort_desc must be null. If the structure is included but an exit specified in it is not to be used, then the corresponding entry variable must be set to sort_$noexit, which is declared:

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

An exit point may not be altered after the call to sort_. 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 below.

(END)

Writing Exit Procedures

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

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

The following exit points are provided:

```
input_file
output_file
compare
input_record
output_record
```

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

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.

(END)

INPUT_FILE EXIT PROCEDURE

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.

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). 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)

(END)

OUTPUT_FILE EXIT PROCEDURE

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.

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_$return (described later). If sort_$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)

(END)

COMPARE EXIT PROCEDURE

A compare exit procedure replaces the standard record comparison procedure of the Sort. The Sort calls the compare exit procedure each time the sorting 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 a 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      ptr,
     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 containig 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.

(END)

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

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

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.

(END)

INPUT_RECORD EXIT PROCEDURE

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

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.

(END)

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 (end of input), 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.

(END)

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 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).

(END)

OUTPUT_RECORD EXIT PROCEDURE

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

1. Each time it has determined the next record in ranked order from the merging process;

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 Sort 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 input_record exit procedure returns to the Sort:

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 merge. 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

(END)

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 merge have equal keys (equal_key = 0), then accumulate data into the output record and delete the record leaving the merge (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, 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. This is accomplished by setting close_exit_sw = "1".

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

(END)

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 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), 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 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 merge. The output_record exit procedure may not alter the contents of this record. For all actions except insert, the Sort 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

(END)

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), there is no record leaving the merge and rec_ptr_2 = null().

4. rec_len_2 is the length of the record leaving the merge. The output_record exit procedure may not change the length of this record. For all actions except insert, the Sort 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. (Input/Output)

Possible values of action are:

0 = accept the output record. The output record is written to the output file. The Sort 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 merge becomes the new output record, and a new record leaving the merge has been obtained.

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

1 = delete the output record. No record is written to the output file. The Sort 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 merge becomes the new output record, and a new record leaving the merge has been obtained.

2 = delete the record leaving the merge. (This action should be used for summarization into the output record.)

(END)

sort_

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 merge has been obtained. The Sort 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 uses the returned values of rec_ptr_1 and rec_len_1 to identify the record to be written. The Sort 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 merge remains the same. The Sort 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 merge 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 sets

(END)

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 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 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 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

(END)

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, 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 Sort's key comparison routine 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 entry sort_ 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 entry sort_$return 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 sets close_exit_sw to this value.

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

(END)

RECORD POINTERS

Since the Sort 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 earlier under the heading Key Fields.

(END)

Entry: sort_$release

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

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 the heading Status Codes. (Output)

Notes

The Sort aligns each record on a double word boundary in a work area.

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_.

(END)

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.

(END)

Entry: sort_$return

The sort_$return entry is used each time the caller retrieves a record, in ranked order, from the Sort. Calls to sort_$return are made from a user supplied output_file 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.

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 the heading 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)

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 (as determined by the key fields of the record). The current record is returned to the caller, and the caller may continue to request records from the Sort.

Name: merge_

The merge_ subroutine provides a generalized file merging capability, which is specialized for execution by user supplied parameters. The basic function of 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 records according to the values of those key fields, and write a single output file of ordered (or "ranked") records. The merge_ subroutine has the following general capabilities:

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

(END)

Very large files, such as multisegment files, can be 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 merging process.

The arguments to the merge_ subroutine include one or more pointers to additional information necessary to specialize merge_ for execution. This additional information is called the Merge Description.

INPUT AND OUTPUT

The user specifies the input and output files. The 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.

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

KEY FIELDS

The user can specify the key fields to be used in ranking records. Key fields are described in the Keys statement - or in the keys structure - of the Merge Description. Up to 20 key fields may be specified. Any PL/I string or numeric data type - except complex or pictured - may be specified for a given key field. Ranking may be ascending, descending, or mixed. For a character string key field, the collating sequence is that of the Multics standard character set. The records of each input file must be in order according to those key fields.

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

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

(END)

follows:

1. If two equal records come from different input files, then the record from the file which is specified earlier in the list of input files (in the input_specs 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 Merge provides exits to user supplied procedures at specific points during the merging process. Exit procedures are named in the Exits statement - or in the exits structure - of the Merge Description. The following exit points are provided:

output_record	To perform special processing for each output record, such as deleting, inserting, or altering records to be output from the Merge; or summarizing data by accumulating it into a summary record.
compare	To compare two records; that is, to rank them for the merging process.

Details of exit procedures are given below under the heading Writing Exit Procedures.

(END)

Usage:

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

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

where:

1. input_specs — is 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 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 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.

2. output_spec — is the specification of the output file. Only one output file may be specified.

(END)

(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_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.

3. *merge_desc* is an array of pointers to the Merge Description. See the heading Merge Description below. (Input)

4. user_out_sw specifies the destination of both the summary report and diagnostic messages for errors detected in the arguments to merge_ or in the Merge Description. (Input)

This argument may have the following values:

(END)

"" = 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, 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.

5. code is a standard Multics status code returned by merge_. Possible values are listed below under the heading Status Codes. (Output)

NOTES

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

STATUS CODES

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

0 Normal return (no errors).

bad_arg One or more arguments specified to merge_, including those in the Merge Description, was invalid or inconsistent. The Merge will have previously written diagnostic messages as directed by the user_out_sw argument. The merging process itself has not been started.

fatal_error The Merge has encountered a fatal error during the merging process. The Merge will have previously generated a specific error message and signalled the sub_error_ condition via the sub_err_ subroutine.

(END)

out_of_sequence	The call to merge_ is not in the sequence required by the Merge; that is, merge_ has been called after initiation of the Merge but before termination of that invocation.

(END)

Merge Description

The Merge Description contains additional information to specialize the Merge for a particular execution. The Merge Description is specified via the merge_desc argument to merge_. The information specified 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 Merge Description is required. As a minimum, the user must specify how records are to be ranked, either by describing key fields in the Keys statement or by naming a compare exit procdure in the Exits statement. Other information in the Merge Description is optional.

The Merge Description may be supplied to merge_ in either of two forms, called source form and internal form.

The source form of the Merge Description is written exactly as specified for the merge command (see the Multics Programmers' Manual, Commands and Active Functions, Section III), and is stored as an ASCII segment; that is, as an unstructured file in the Multics storage system. If source form is used, then the merge_desc argument to merge_ must have an array extent of 1 and the one pointer must be a pointer to the segment. (The segment must contain only the Merge Description.) The source form is useful when the user writes the Merge Description and supplies it to the procedure which calls merge_.

The internal form of the Merge Description is a set of one or two structures. The merge_desc argument must have an array extent of 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)

where the two structures (keys and exits) are defined below. The internal form is useful when the procedure calling merge_ constructs the Merge Description.

(END)

KEYS STRUCTURE

The keys structure is used when the caller describes key fields. The Merge's standard compare routine 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 corresponding pointer in the array merge_desc must be null and the compare exit must be specified in the exits structure. The user supplied compare routine will then be used to rank records.

The keys structure is:

```
dcl 1 keys,
      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.

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 the table 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.

(END)

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 the table 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.

(END)

DATATYPE ENCODING AND SEMANTICS OF SIZE

	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	floec	9 bit digit	1 - 59	n + 2 digits

(END)

EXITS STRUCTURE

The exits structure is:

```
dcl 1 exits,
      1 version          fixed bin init(1),
      2 compare          entry,
      2 reserved         entry init(merge_$noexit),
      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. reserved — is reserved for future use.

4. output_record — specifies the entry point of a user supplied output_record exit procedure.

ENTRY VARIABLES

In the exits structure, 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 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 the exits structure is to be used, then that structure can be omitted and the corresponding pointer in the array merge_desc must be null. If the structure is included but an exit specified in it is not to be used, then the corresponding entry variable must be set to merge_$noexit, which is declared:

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

(END)

An exit point may not be altered after the call to 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 below.

(END)

Writing Exit Procedures

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

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

The following exit points are provided:

output_record
compare

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

(END)

COMPARE EXIT PROCEDURE

A compare exit procedure replaces the standard record comparison procedure of the Merge. The Merge calls the compare exit procedure each time the merging process is ready to rank two records; that is, to determine which of the two is first in the merged 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 a corresponding return value to the Merge. 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      ptr,
     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 containig 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.

(END)

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

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

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.

(END)

OUTPUT_RECORD EXIT PROCEDURE

An output_record exit procedure supplements the standard output writing function of the Merge. The Merge calls the output_record exit procedure:

1. Each time it has determined the next record in ranked order from the merging process;

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 Merge gives the output_record exit procedure access to two records:

1. The output record, about to be written to the output file.

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 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 merge. 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.

(END)

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 merge have equal keys (equal_key = 0), then accumulate data into the output record and delete the record leavig the merge (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 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 Merge. This is accomplished by setting close_exit_sw = "1".

The output_record exit procedure must return to the 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);
```

(END)

```
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 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), 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 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 merge. The output_record exit procedure may not alter the contents of this record. For all actions except insert, the 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), there is no record leaving the merge and rec_ptr_2 = null().

(END)

4. rec_len_2 is the length of the record leaving the merge. The output_record exit procedure may not change the length of this record. For all actions except insert, the 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 Merge. (Input/Output)

Possible values of action are:

0 = accept the output record. The output record is written to the output file. The 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 merge becomes the new output record, and a new record leaving the merge has been obtained.

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

1 = delete the output record. No record is written to the output file. The 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 merge becomes the new output record, and a new record leaving the merge has been obtained.

2 = delete the record leaving the merge. (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 merge has been obtained. The Merge uses the returned values of rec_ptr_1 and rec_len_1 to identify the output record

(END)

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 Merge uses the returned values of rec_ptr_1 and rec_len_1 to identify the record to be written. The 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 merge remains the same. The Merge uses the returned values of rec_pt_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 merge 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 output and end of output), only one record is present and the 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

(END)

exit procedure, then the 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 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 Merge sets seq_check_sw to this value.

Sequence checking means comparing the output record to the record previously written to the output file. Sequence checking is performed after the output_record exit procedure returns to the 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 Merge's key comparison routine is used. If the user supplied a

(END)

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 merge_; see the entry merge_ above.

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 Merge sets close_exit_sw to this value.

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

(END)

RECORD POINTERS

Since the 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 earlier under the heading Key Fields.

Name: sort

The sort command is described in the Multics Programmers' Manual, Commands and Active Functions, Section III. This description includes only additional optional control arguments which are not described in MPM Commands.

Usage

sort input_specs output_specs control_args

where:

3. control_args can be chosen from the following (in addition to those control arguments specified in MPM Commands):

-time prints timing information for the Sort:
System load (hmu)
Merge order
String size
and for each phase of the Sort:
Elapsed time
Real cpu time
Virtual cpu time
Page faults
Paging device faults
Comparisons executed

(Times are given in seconds.)

(END)

Sort/Merge PLM

-merge_order m — specifies that the merge order is to be m. The argument m must be a decimal integer. This argument is meaningful only if all input files are in the Storage System, so that the total input file size can be obtained by the Sort.

-string_size s — specifies that the string size (as produced during the presort) is to be s bytes. The argument s must be a decimal integer, and must be less than the system maximum segment size. The actual size of any string may differ somewhat from s, since the length of the last record inserted into the string may not exactly match the space available.

Merge order and string size cannot both be specified.

-debug — specifies that temporary files will be left initiated (but truncated to zero length) after completion of the Sort. This argument is intended for use with performance measurement and analysis tools which print reference names, such as sample_refs.

If this argument is omitted, temporary files will be deleted after completion of the Sort.

If -debug is specified, deletion of temporary files must be done explicitly by the user. Some temporary files are in the process directory; the work files are in the directory specified by the -temp_dir argument. The names of all temporary files are generated uniquely for each invocation of the Sort, and always contain the string "sort_".

(END)

Name: merge

The merge command is described in the Multics Programmers' Manual, Commands and Active Functions, Section III. This description includes only additional optional control arguments which are not described in MPM Commands.

Usage

merge input_specs output_specs control_args

where:

3. control_args can be chosen from the following (in addition to those control arguments specified in MPM Commands):

-time prints timing information for the Merge:
System load (hmu)
and for each phase of the Merge:
Elapsed time
Real cpu time
Virtual cpu time
Page faults
Paging device faults
Comparisons executed

(Times are given in seconds.)

-debug specifies that temporary files will be left initiated (but truncated to zero length) after completion of the Merge. This argument is intended for use with performance measurement and analysis tools which print reference names, such as sample_refs.

If this argument is omitted, temporary files will be deleted after completion of the Merge.

If -debug is specified, deletion of temporary files must be done explicitly by the user. All temporary files are in the process directory. The names of all temporary files are generated uniquely for each invocation of the Merge, and always contain the string "sort_".
(END)

Name: sort_

The sort_ subroutine is described in the Multics Programmers' Manual, Subroutines, Section II. This description includes only additional entry points which are not described in MPM Subroutines.

Entry: sort_$initiate

The sort_$initiate entry point is used when the Sort is "driven" by its caller. The Sort is said to be driven if the caller supplies a procedure which calls (or directly performs) the input file processing and output file processing procedures. Such a driver must have the following general form:

```
call sort_$initiate(arguments);

call input_file_proc(code);

call sort_$commence(code);

call output_file_proc(code);

call sort_$terminate(code);
```

where:

1. sort_$initiate — is the procedure of the Sort which must be called first (it "initiates" the Sort).

2. input_file_proc — is an input_file procedure, as specified in the description of the sort_ subroutine in MPM Subroutines. Instead of calling an input_file procedure, the driver may perform the necessary functions directly.

3. sort_$commence — is the procedure of the Sort which must be called when the input_file procedure has completed releasing records to the sorting process (it "commences" the merging process). See the entry sort_$commence below.

4. output_file_proc — is an output_file procedure, as specified in the description of the sort_ subroutine in MPM Subroutines.

Sort/Merge PLM

Instead of calling an output_file procedure, the driver may perform the necessary functions directly.

5. sort_$terminate — is the procedure of the Sort which must be called when the output_file procedure has completed retrieving records from the Sort (it "terminates" the sorting process). See the entry sort_$terminate below.

The entry points sort_$initiate, sort_$commence, and sort_$terminate are specifically designed to be used by COBOL object programs. They support the ANSI COBOL Sort/Merge Module, Level 2 (the SORT, RELEASE, and RETURN statements).

Normally, when called as a command (sort) or as a subroutine (sort_), the Sort itself contains the driver to perform the five calls listed above.

Usage

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

call sort_$initiate(temp_dir, keys_ptr, exits_ptr,
                    user_out_sw, file_size, code);
```

where:

1. temp_dir — is the pathname of the directory which will contain the Sort's work files. If this argument is "", then work files will be contained in the user's process directory.

   This argument should be used when the process directory will not be large enough to contain the work files. The get_dir_ functior may be used to obtain the name of the user's current working directory. (Input)

2. keys_ptr — is a pointer to the keys structure, which describes the key fields to be used for ranking records. This structure is identical to that specified under the heading Keys Structure in the description of the sort_ subroutine in MPM Subroutines, Section II. If the user is supplying a compare exit

Sort/Merge PLM

procedure, then keys_ptr must be null and the compare procedure must be specified in the exits structure. (Input)

3. exits_ptr — is a pointer to the exits structure, which specifies which exit points are to be used and gives the entry point names of the corresponding user supplied exit procedures. This structure is identical to that specified under the heading Exits Structure in the description of the sort_ subroutine in MPM Subroutines, Section II. If no exits are to be used, then exits_ptr must be null. If the compare exit is specified, then keys must not be described. (Input)

4. user_out_sw — specifies the destination of both the Sort's summary report and diagnostic messages for errors detected in the arguments to sort_$initiate. (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_$initiate 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. This switch must be attached and open for stream output.

5. file_size — is the total amount of data to be sorted, in millions of bytes. If this argument is zero, the default assumption is approximately one million bytes (file_size = 1.0). (Input)

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.

6. code — is a standard Multics status code returned by sort_$initiate. Possible values are listed below under the heading Status Codes. (Output)

Entry Variables

Entry variables in the exits structure should be set (either initialized or assigned) by the procedure which calls the sort_initiate entry point.

Cleanup Handling

In order that the Sort can be terminated properly in case of an abnormal exit, the cleanup procedure of the caller of sort_$initiate must include a call to the entry point sort_$terminate.

Status codes

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

0	Normal return (no errors).
bad_arg	One or more arguments specified to sort_$initiate, including the keys and exits structures, was invalid or inconsistent. The Sort 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 has encountered a fatal error. The Sort will have previously generated a specific error message and signalled the sub_error_ condition via the sub_err_ subroutine.

Sort/Merge PLM

out_of_sequence	The call to sort_$initiate is not in the sequence required by the Sort; e.g., sort_$initiate has been called after initiation of the Sort but before normal termination of that invocation via a call to sort_$terminate.

Entry: sort_$commence

The sort_$commence entry point must be called after the driver of the Sort has completed its input_file procedure. See the entry point sort_$initiate above. The call to sort_$commence informs the Sort that end of input has been reached. Upon return from sort_$commence, the driver can begin its output_file procedure.

Usage

```
dcl sort_$commence  entry(fixed bin(35));

call sort_$commence(code);
```

where code is a standard Multics status code returned by sort_$commence. Possible values are listed below under the heading Status Codes. (Output)

Status Codes

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

0	Normal return (no errors).
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.
out_of_sequence	The call to sort_$commence is not in the sequence required by the Sort; e.g., sort_$commence has been called before sort_$initiate.

Sort/Merge PLM

Entry: sort_$terminate

The sort_$terminate entry point must be called after the driver of the Sort has completed its output_file procedure. See the entry point sort_$initiate above. The call to sort_$terminate informs the Sort that the current execution of the Sort is complete. Upon return from sort_$terminate, the caller can initiate another execution of the Sort.

Usage

```
dcl sort_$terminate entry(fixed bin(35));

call sort_$terminate(code);
```

where code is a standard Multics system status code returned by sort_$terminate. Possible values are listed below under the heading Status Codes. (Output)

Status Codes

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

0	Normal return (no errors).
out_of_sequence	The call to sort_$terminate is not in the sequence required by the Sort; e.g., sort_$terminate has been called before sort_$initiate.

(END)