Burroughs

B 6000/B 7000 Series

SYSTEM SOFTWARE OPERATIONAL GUIDE

VOLUME 1

PRICED ITEM

Printed in U.S.A.

June 1980

5011661

Burroughs believes that the software described in this manual is accurate and reliable, and much care has been taken in its preparation. However, no responsibility, financial or otherwise, can be accepted for any consequences arising out of the use of this material, including loss of profit, indirect, special, or consequential damages. There are no warranties which extend beyond the program specification.

The Customer should exercise care to assure that use of the software will be in full compliance with laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change. Revisions may be issued from time to time to advise of changes and/or additions.

Correspondence regarding this document should be addressed directly to Burroughs Corporation, P. O. Box 4040, El Monte, California 91734, Attn: Publications Department, TIO – West.

PREFACE

The SYSTEM SOFTWARE OPERATIONAL GUIDE (SOG), in two volumes, provides descriptions and operating instructions for the utilities available to users of the BURROUGHS B 7000/B 6000 series of data processors. Additionally, chapters documenting other pertinent features of the system are included.

Volume 1 contains information about those utilities of interest to programmers and systems personnel.

Volume 2 contains information more often required by operations personnel.

A degree of overlap exists between the two volumes of the SOG manual, as certain subjects are pertinent to both the operations and the technical staffs. Each volume of the SOG contains a listing of the information available in both volumes.

SOG VOLUME 1

CHAPTER	SUBJECT
1	BACKUP
2	CARDLINE
3	COMPARE
4	DUMPALL
5	FILEDATA
6	INTERACTIVE XREF
7	ISAM (Index Sequential Access Methods)
8	LOADCONTROL
9	INTRINSICS
10	PATCH
11	SORT
12	GUARDFILE
13	DCSTATUS

SOG VOLUME 2

RAILROAD DIAGRAMS

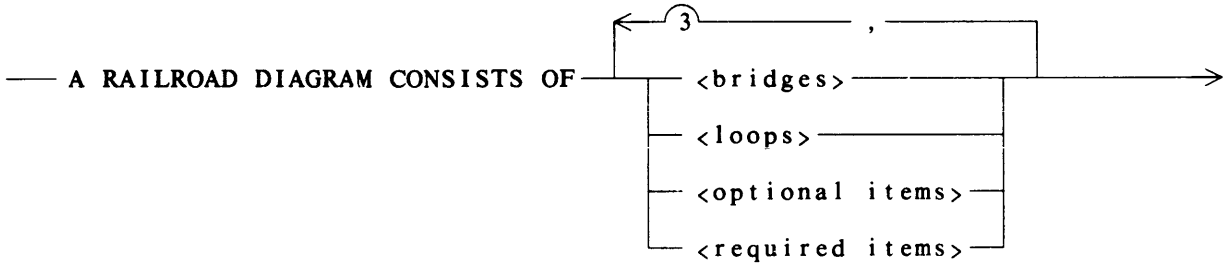
A railroad diagram is a technique used to graphically represent the syntax of utility input statements, language verbs, and Operator Display Terminal commands.

Traversing a railroad diagram from left to right, or in the direction of the arrowheads, and adhering to the limits illustrated by bridges produces a syntactically valid statement. Continuation from one line of a diagram to another is represented by a right arrow ">" appearing at the end of the current line and the beginning of the next line. The complete syntax diagram is terminated by a vertical bar "|" or a diamond "<>".

Items contained in broken brackets "< >" are syntactic variables that are further defined in the manual or require the user to supply the requested information.

Uppercase items must appear literally. Minimum abbreviations are underlined.

Example



The following syntactically valid statements may be constructed from the above diagram:

A RAILROAD DIAGRAM CONSISTS OF <bridges> AND IS TERMINATED BY A VERTICAL BAR OR DIAMOND.

A RAILROAD DIAGRAM CONSISTS OF <optional items> AND IS TERMINATED BY A VERTICAL BAR OR DIAMOND.

A RAILROAD DIAGRAM CONSISTS OF <bridges>, <loops> AND IS TERMINATED BY A VERTICAL BAR OR DIAMOND.

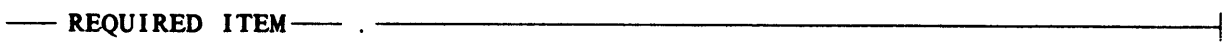
A RAILROAD DIAGRAM CONSISTS OF <optional items>, <required items>, <bridges>, <loops> AND IS TERMINATED BY A VERTICAL BAR OR DIAMOND.

RAILROAD COMPONENTS

<required items>

No alternate path through the railroad diagram exists for required items or required punctuation.

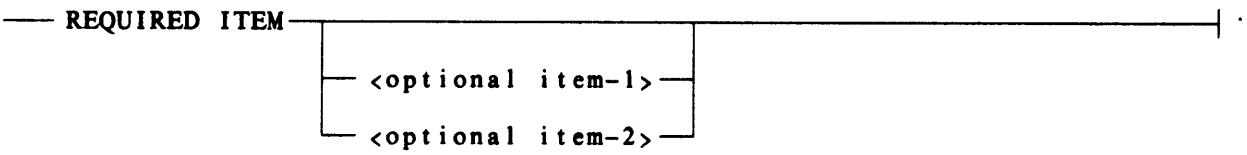
Example



<optional items>

Items shown as a vertical list indicate that the user must make a choice of the items specified. An empty path through the list allows the <optional item> to be absent.

Example



The following valid statements may be constructed from the above diagram:

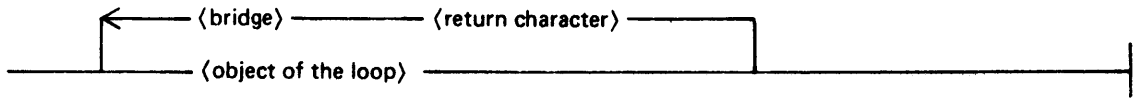
REQUIRED ITEM

REQUIRED ITEM <optional item-1>

REQUIRED ITEM <optional item-2>

<loops>

A <loop> is a recurrent path through a railroad diagram and has the following general format:



Example

The following are some of the statements that can be constructed from the above diagram:

<optional item-1>

<optional item-1>,<optional item-1>

<optional item-2>,<optional item-1>

A <loop> must be traversed in the direction of the arrowheads, and the limits specified by bridges cannot be exceeded.

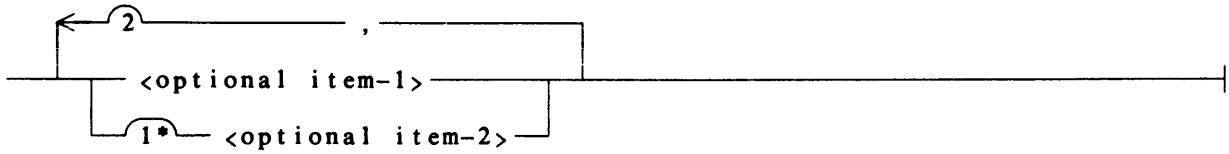
<bridges>

A <bridge> illustrates the minimum or maximum number of times a path may be traversed in a railroad diagram.

The two forms of <bridges> are as follows:

/ n \	n is an integer that specifies the maximum number of times the path may be traversed.
/ n* \	n is an integer that specifies the minimum number of times the path must be traversed.

Example



The loop may be traversed a maximum of two times; however, the path for <optional item-2> must be traversed at least one time.

The following are some of the statements that can be constructed from the above diagram:

<optional item-1>,<optional item-2>

<optional item-2>,<optional item-2>,<optional item-1>

<optional item-2>

BACKUP

TABLE OF CONTENTS

1. RATIONALE FOR BACKUP FACILITY

Slower peripheral devices, such as printers and punches, have typically been bottlenecks on computer systems. One problem is that a printer or punch may not be available for assignment when an executing program requires one; another problem is that the operation of a printer or punch is relatively slow and therefore ties up the controlling program as well as some of the other system resources that the program is using. In still other cases, such as an exception report file or monitoring files, the printer is typically used infrequently; therefore, assigning the printer directly to the program would needlessly tie the printer up and make it unavailable to other programs.

The Burroughs B 7000/B 6000 series of computer systems are dedicated to the effective use of the overall pool of system resources in a multiprogramming environment. To use slower peripherals more effectively, these systems provide the capability of backup files.

When the "backup" technique is used, a program requesting a printer or punch device is assigned a faster peripheral such as tape, disk, or pack. These peripheral devices simulate a printer or punch; the program writes to them logically, but the physical output operations are less frequent (because blocking is possible) and therefore, not as time consuming.

The system can be placed in automatic backup mode by means of two system options; LPBDONLY (for line printer files) and CPBDONLY (for card punch files). These options may be set or reset by use of the OP+ and OP- messages. With these options set, logical files whose KIND is set to PRINTER or PUNCH, are changed to PRINTER BACKUP DISK or PUNCH BACKUP DISK. If a backup device other than head-per-track disk (which is the default device for backup files) is desired, the alternate device can be specified by use of the Operator Display Terminal (ODT) SB message.

Statements in Work Flow Language can be used to set the KIND attribute to the kind or kinds of backup device(s) desired for a job. Such a statement can be overridden by the SB message.

Another method of creating backup files is by use of the OU message. If a program requires a line printer or card punch and none is available, the system operator can specify, in an OU message, the device to be used for backup. Consequently, a transfer later takes place from the completed backup file to the printer or punch. This operation can be performed automatically by a system utility program (SYSTEM/BACKUP) which makes minimum demands on system resources yet operates the peripheral devices at efficient speeds.

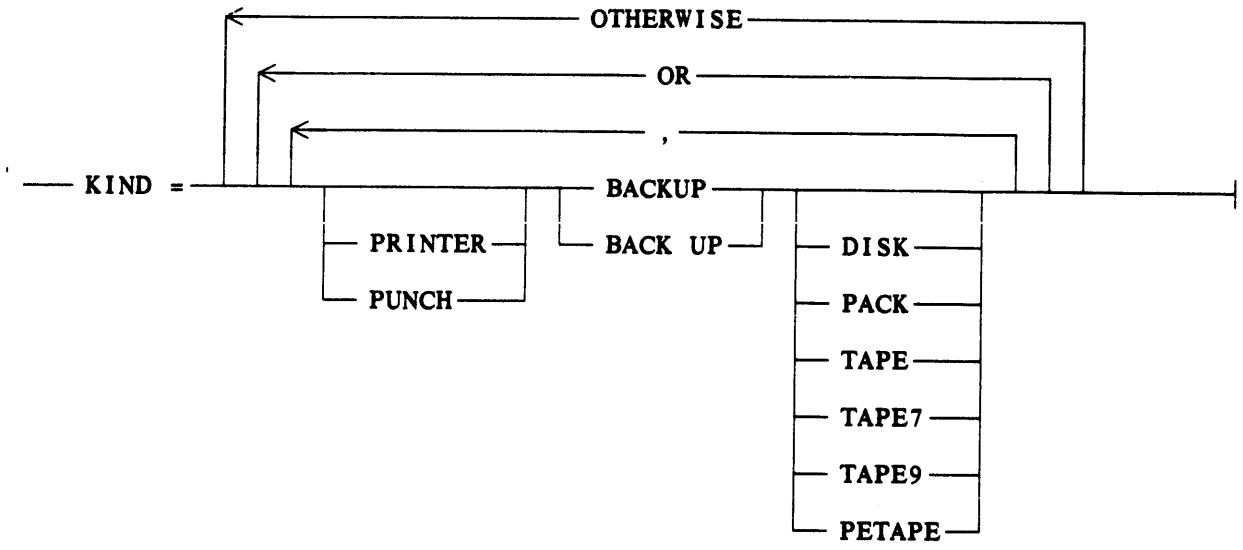
BACKUP

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

2. PROGRAMMER CONSIDERATIONS

A programmer is able to specify (via WFL syntax) a variety of backup media; DISKPACK as a backup medium is provided along with DISK, TAPE, TAPE7, TAPE9, and PETAPE. The following syntax diagrams illustrate both "old" WFL and "new" WFL usage:

OLD WFL SYNTAX



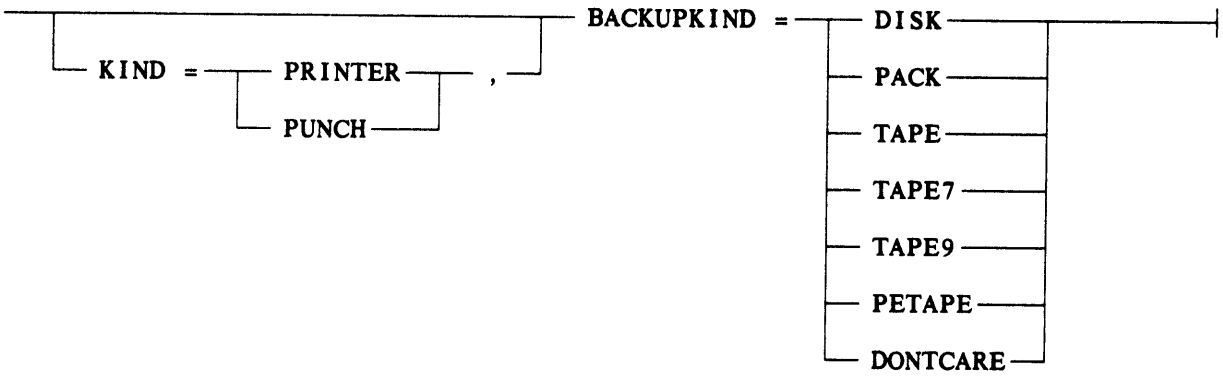
Semantics

Omitting PRINTER or PUNCH specifies that PRINTER is to be used.

Omitting a backup medium specifies that the MCP is first to try DISK, then is to try TAPE.

The OR, OTHERWISE, and "," allow concentration of various requests in a specific order of preference.

NEW WFL SYNTAX



Semantics

Omitting KIND = PRINTER or KIND = PUNCH specifies that PRINTER is to be used.

Using the mnemonic DONTCARE specifies that the MCP is first to try DISK, then is to try TAPE.

If the programmer specifies the following in an "old" WFL deck:

```
FILE CF(KIND=PRINTER BACKUP TAPE9 OR PRINTER BACKUP TAPE7);
```

a printer backup file is opened on a nine-track NRZ tape unit, if one is available. If no nine-track unit is available, a search for a seven-track NRZ unit is initiated. If neither is available, the operator receives a message asking for a nine or seven-track tape that can be used for printer backup of the CF file. At this point, the operator has the option of forcing the file to a backup device.

The programmer may alternately specify:

```
FILE OF(KIND=PRINTER OR PRINTER BACKUP PACK OR PRINTER
        BACKUP DISK);
```

in which case, the preference is first for a printer, then for pack, otherwise disk.

If the programmer specifies the following in a "new" WFL deck:

```
FILE LP(KIND=PRINTER,BACKUPKIND=TAPE);
```

a printer file is opened on a tape (any tape) unit if one is available.

The programmer has the option of providing a PACKNAME with a printer file. If a PACKNAME is specified, the MCP looks for the specified pack. If PACKNAME is not specified, a system resource pack is selected, if available.

If a programmer specifies the PACKNAME file attribute with a PRINTER or PUNCH file, the MCP guarantees that if the backup medium is PACK, then it is the specified pack. The MCP does not accept an OUPKnnn to any other pack. This restriction reflects the necessity for directing backup diskpack to a pack that is to be removed and used on another system. Furthermore, the programmer can prevent AUTOBACKUP from printing and removing that file destined to be transported by using the BDNAME task attribute and the BDBASE task attribute.

If a system does not have any backup substitutions and if the programmer specifies that a file is to be spooled on one of the following backup media devices - DISKPACK, TAPE7, TAPE9, PETAPE and/or TAPE - the options LPBDONLY and CPBDONLY have no effect because the LPBDONLY and CPBDONLY control only non-direct files requesting a PUNCH or PRINTER.

If the programmer uses a direct file, the file cannot be spooled on any backup media. If the programmer specifies a backup medium for a direct file, the task is DSed because backup and direct files are incompatible.

NOTE

The programmer is able to specify on which medium the backup is to be spooled.

3. OPERATOR CONSIDERATIONS

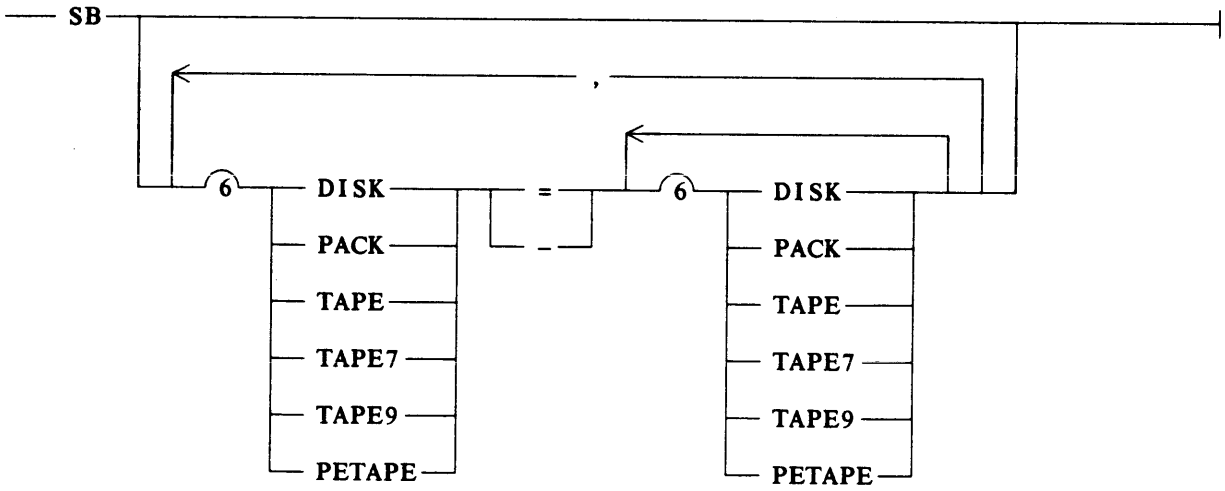
If a request for a specific peripheral or backup peripherals cannot be satisfied (because the peripheral is either not ready or under the exclusive use of another task), the system operator has the choice of

1. Physically making the requested peripheral available.
2. Waiting until the other task releases the requested peripheral.
3. Entering the SB (Substitute Backup) message to respecify which backup media to use.
4. Entering the OU message to direct the file to an available peripheral.
5. Entering the DS message to terminate the program.

SB MESSAGE

The operator can modify, and in some instances override, the programmer's choice of the six previously mentioned types of backup media. This option is invoked by use of the SB message. The following syntax illustrates the SB message:

Syntax



Semantics

Diverting backup disk to backup diskpack may result in a better balance between disk/diskpack channel utilization, in which case, SB DISK=PACK (ETX) yields increased system throughput.

The following examples illustrate various SB messages and their actions.

Example 1

```
SB DISK=PACK
```

Action: Directs all disk backup to pack (assuming pack backup is already going to pack).

Example 2

```
SB DISK=PACK, TAPE=PACK, TAPE7=PACK, TAPE9=PACK, PETAPE=PACK
```

Action: Directs all backup to pack.

Example 3

```
SB   PACK=DISK,   TAPE=TAPE7   TAPE9,   TAPE7=TAPE7   TAPE9,
TAPE9=TAPE7 TAPE9, PETAPE=TAPE7 TAPE9
```

Action: Keeps backup off of PETAPE; wants backup tape files to first try seven track, then try nine track (all disk and pack backup should be on disk).

If backup media selection is left to the discretion of a programmer or a number of programmers, leaving the SB message set to what it is after a cold start may be desirable. Thus, if the operator desires to return to the setting that existed at cold start, the following SB message can be used:

```
SB DISK=DISK, PACK=PACK, TAPE=TAPE, TAPE7=TAPE7, TAPE9=TAPE9,
PETAPE=PETAPE
```

However, SB is not recursive. Thus,

```
SB DISK=PACK, PACK=DISK
```

serves to divert backup disk to backup diskpack and to divert backup diskpack to backup disk in a simple crossover. Files declared backup disk go to diskpack, and files declared backup diskpack, go to disk.

An error condition occurs if duplicate entries appear in the SB message. For example,

```
SB DISK=PACK TAPE PACK
```

causes the error response REDUNDANT SUBSTITUTION.

The current setting of Substitute Backup may be interrogated by inputting

```
SB
```

which causes the status of Substitute Backup to be displayed.

The operator could have specified LPBDONLY for the system. In such a case, any request for a printer by a non-direct file is transformed into a request for printer backup disk. Any request for a card punch file when CPBDONLY is set, is transformed into a request for punch backup disk. By setting appropriate bits in the task attribute OPTION, the programmer can direct that task to operate as if LPBDONLY and CPBDONLY are set. An MCS, such as CANDE or RJE, can

also set these bits in the OPTION task attribute of the user tasks it processes, thereby causing the same action.

When LPBDONLY or CPBDONLY is set and a task opens a non-direct printer or punch file, the system selects the backup medium specified in the SB message as opposed to DISK. For example, if LPBDONLY is set and if SB DISK=PACK has been previously entered, any task opening a non-direct printer file results in a printer backup pack file being opened.

REQUIRES RSVP MESSAGE

If a backup medium for a file is not available, and nothing has been substituted by the SB message, the MCP generates an RSVP message in the form

> <task no.> <filename> REQUIRES <output medium> BACKUP
> (<backup medium>)

with the <output medium> being LP or CP.

If a PACKNAME file attribute has been specified, the message appears as:

> <task no.> <filename> REQUIRES <output medium> BACKUP
> (<packname>)

If "old" WFL is used, one or more <backup medium>s may be listed by the RSVP message, depending on the programmer's specifications, as defined under PROGRAMMER CONSIDERATIONS. Only one <backup medium> can be listed for "new" WFL.

The REQUIRES RSVP message allows one of the following operator responses:

1. Use OU message, which is a reply to the REQUIRES RSVP message.

2. Make the specified peripheral type ready. The MCP notes the status change and wakes up the program.

3. Use SB to equate a backup medium that is present and available. Initiating an SB causes the waiting process to search again for an available backup medium.

4. DS the program.

OU MESSAGE

An OU is not subject to SB equating. If the operator inputs <mix no.> OUDK and the site has head-per-track disk, the file goes to head-per-track disk regardless of how SB is set.

The use of the OU message is essentially a reply to an RSVP message indicating an output medium is required.

Example

<mix no.> OUPK	Places the file on the appropriate diskpack. If the file has PACKNAME set, the chosen diskpack is the pack with that name. If no PACKNAME is given, the system resource pack is used.
<mix no.> OUPKnnn	Places the file on the specified pack. If no PACKNAME has been specified by the programmer, the programmer can OUPKnnn to any native mode write-enabled base pack. If a PACKNAME has been specified by the programmer, the MCP insists that if the file goes out on diskpack, it goes to a native mode write-enabled base pack with the specified name; PACKNAME has no effect with backup media types other than pack. An OU to a different backup medium is allowed.
<mix no.> OUMT	Places the file on magnetic tape.
<mix no.> OUMTnnn	Places the file on the specified magnetic tape unit of the number nnn.
<mix no.> OUDK	Places the file on head-per-track disk.

Responses to inappropriate OU messages are as follows:

IS DIRECT FILE: CANNOT BACKUP

A direct file asking for a line printer or card punch cannot go to backup under any circumstances. Direct implies that the program can look specifically at result descriptors and set error maskout. Direct files must therefore deal with the actual target peripheral.

THAT PK IS NOT PRESENT

An OUPK failed because the PACKNAME specified in the attribute list of the file could not be found on the system. This message is also generated when a pack of that name is present but is inappropriate because it is interchange, write-locked out, or is a continuation pack.

REQUIRES PK WITH CORRECT NAME

An attempt was made to OUPKnnn for a file with the PACKNAME attribute set, and the PKnnn did not have the correct name.

PK PACK IS NOT PRESENT

An attempt was made to OUPK when no PACKNAME was provided by the programmer and no designated system resource pack was present.

REQD PKUNIT NOT A MOUNTED-BP

The operator directed OUPKnnn to a pack that was not a mounted native mode base pack. Backup to diskpack must be to a mounted, write-enabled, ready, native mode base pack.

NEED AN OUTPUT TAPE FOR OUMT

OUMT failed because no tape was in proper state.

NOTE

An OU for a diskpack backup file requires a write-enabled native mode base pack. If the OUed pack is interchange, not write-enabled, or is a continuation pack, the OU is not honored.

ERROR HANDLING

The operator has the option of deciding whether or not to continue after an irrecoverable parity error. On encountering such an error, an ACCEPT (AX) message is sent to the operator giving the operator the option of continuing or stopping. If the printout is resumed, all lines read with a parity error are flagged on the output.

Allowable responses are DS and OK. If DS is entered, processing of the current backup file is discontinued. If OK is entered, processing continues. If anything else is entered, the ACCEPT message is repeated.

BACKUP

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

4. BACKUP FILES

NAMING CONVENTION

When a BACKUP DISKPACK is selected, the PACKNAME file attribute can be used to select a particular native mode pack family. An interchange disk pack family cannot be used.

The format of a printer backup disk file name is as follows:

BD/<job no.>/<task no.>/<modified filename>

The format of a punch backup disk file name is as follows:

BP/<job no.>/<task no.>/<modified filename>

The <modified filename> refers to three digits, 000 to 999, appended to the front of the declared printer or punch file name to prevent duplicate file names. This number is incremented each time a backup file is opened by the task.

If more than one backup disk file is created by a program, the system creates a "tree" of files in the directory in the following manner:

BD/<job index>

Specific examples are:

BD/0000365/0000366/000TASKFILE

BD/0000365/0000366/001LINE

BD/0000365/0000366/002PRNT

A backup file on tape is labeled BACKUP/<filename>, where <filename> is the name of the file as specified by the program creating it. Backup tapes may be written as multi-reel files and as multi-file reels. The system intermixes printer and punch backup files on a backup tape.

FILE FORMAT

Backup DISK, PACK, and TAPE files are variable length record, fixed-length block files. Each block is 300 words long, with word 298 containing the number of records in the block and word 299 containing the record number of the first record. Within a block, each logical record is composed of one control word followed by zero or more words of data. A terminal control word of all zeros indicates that no more records appear in the present block.

Control Word

Each control word is divided into three specific fields. These fields are:

FIELD	CONTENTS
[47:28]	Identical to the corresponding portion of an I/O control word.
[19:3]	Character count residue for the data record (if the record to be printed consists of complete words, the value of the field will be zero).
[16:17]	Word count for the following data in the record in full words, not counting the control word.

Control Record

The first record of the file is a control record containing information that is not printed or punched. This first record is minimally 12 words long excluding the control word. The following information describes the words in the first record:

WORD 0:

Is the control word.

WORD 1:

Is the block character control word which consists of the following:

FIELD	CONTENTS
[47:8]	The index to FORMMESSAGE.
[39:8]	The index to JOBNAME.
[31:8]	The index to CHARGECODE.
[23:8]	The index to USERCODE.
[3:1]	A 1 if the file is a backup disk file, the label type of the file is STANDARD, and the label entries are not present.

[2:1]	A 1 if JOBNUMBER is wanted or a 0 if JOBNUMBER is not wanted.
[1:1]	A 1 if LSN = origin is remote or a 0 if origin is not remote.
[0:1]	A 1 if control word is valid or a 0 if control word is not valid.

WORD 2:

Is the logical kind of control word which consists of the following:

FIELD	CONTENTS
[47:1]	A 1 if forms are required or a 0 if forms are not required. The above field contained in WORD 2 and the dependency of the FORMMESSAGE beginning at WORD 12 are being eliminated. The information is contained in WORD 1.
[5:6]	The unit type of the backup file. For example, 11 (decimal) if card punch or 7 if line printer.

WORD 3:

Is a path control word consisting of the following:

FIELD	CONTENTS
[15:1]	A 1 if the originating unit is a remote unit or a 0 if the originating unit is not a remote unit.
[14:15]	The number of the unit in the system which introduced the file. If the origin was remote, then the number is an LSN; otherwise, it is the physical unit where introduced.

WORD 4:

TRAINID (file attribute). If non-zero, the user-specified train-printer character set is used for printing. If 0, the file is generated as if for a drum printer.

WORD 5:

EXTMODE (file attribute).

WORD 6:

LABELTYPE (file attribute).

WORD 7:

I/O mask for page specifications. Refer to the Input/Output Subsystem Reference Manual, form number 5001779, for a complete description of the PAGESIZE, LINENUM and PAGE file attributes.

WORD 8:

The job number of the job being printed.

WORD 9:

Contains the level number of the backup file.

WORDS 10 THROUGH 11

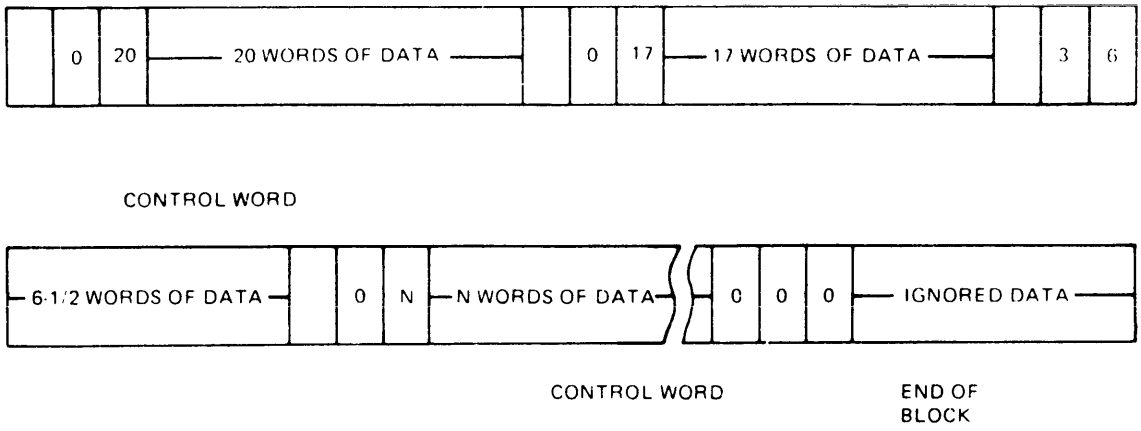
Not presently defined.

WORD 12:

FORMMESSAGE (temporary). The FORMMESSAGE begins at this point; however, this word may change in the future. The correct value of the FORMMESSAGE can be found in WORD 1.

Blocking

Records in a backup file are not split across blocks. That is, if the last record in a given block ended in word 290 and the next record is 12 words long, then word 291 is a control word containing all zeros. This control word indicates the end of the present block and indicates that the next record begins at the start of the next block. The following illustration shows a typical block of **BACKUP** records contained within **BACKUP** files:



FILE LAYOUT FOR BACKUP FILES ON TAPE

BACKUP

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

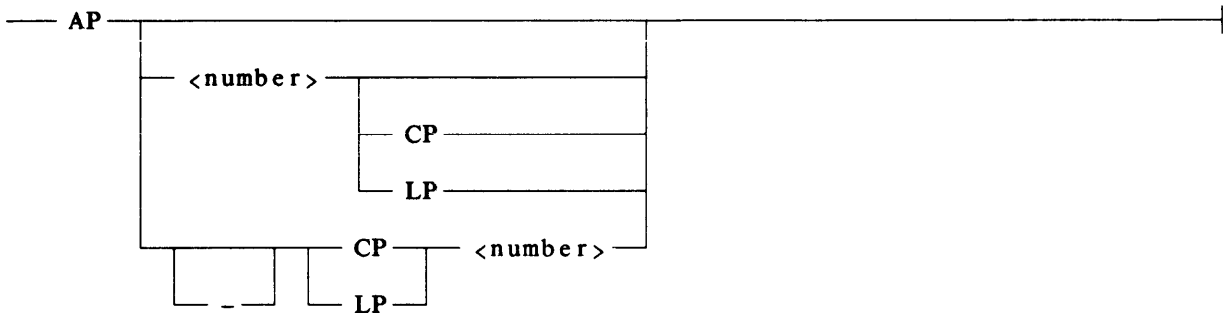
5. AUTOBACKUP

AP MESSAGE

The AP (AUTOPRINT/AUTOPUNCH) message sets the number of line printers and/or card punches to be made available for the automatic output of backup files. This message can also be used to select certain line printers and/or card punches to be used as preferred AUTOBACKUP units. This number (output devices) is set to zero at Halt/Load time if the MCP option AUTORECOVERY is reset.

AUTOBACKUP is invoked using the AP message.

Syntax



Semantics

The AP message can appear simply as AP, in which case the system responds by displaying the current number of line printers and card punches available to AUTOBACKUP. For example:

```
AP
AP MAX=3;AP-ED LPS=2;AP-ED CPS=1
```

When a number immediately follows AP, that number is used as the maximum number of printer AUTOBACKUPs that may run at any one time. Also, when a number and a device type immediately follow AP, that number is used as the maximum count of AUTOBACKUPs of that type that may run at one time. For example:

```
AP3
AP MAX=3;AP-ED LPS=2;AP-ED CPS=1
```

If the number following the AP is 0, no automatic printing occurs.

When a device and unit number immediately follow the AP, such as

```
AP LP 12
```

then the indicated output device is marked as a preferred AUTOBACKUP unit. AUTOBACKUP always attempts to use such a unit before trying to use another unit.

Only if no APed units are available, is output started to a non-APed unit. Whenever a unit is APed, the count of AUTOBACKUPs allowed for that unit type is automatically increased by one. The number of devices assigned, as described above, may exceed the number of allowed AUTOBACKUPs. Thus, this sequence of AP messages

```
AP 0
AP LP 11
AP LP 11
AP 1
```

are all equivalent.

When the device is preceded by a hyphen, as in

```
AP-CP 13
```

the indicated unit is marked as a "non-APed" or "unpreferred" unit, and the count of allowed AUTOBACKUPs for that unit type is automatically decreased by one (unless it is already 0). Also, when such a message references a device on which output is currently being generated, the activity is allowed to proceed to a normal termination before the device is disabled.

QUEUEING AND UNIT PREFERENCE

When use of AP results in all line printers being freed of the APed status and AP is set to 0, the queue of disk/diskpack backup yet to be printed is forgotten. This process does not affect the actual backup files, except that they are not printed automatically by AUTOBACKUP. If subsequently, any LP is given AP status or AP is set to a non-zero number, all disk and native mode diskpacks are searched for backup files queued for printing. The larger the system, the more time is consumed in the search. A similar situation is true for card punch.

Backup files introduced by Library Maintenance (that is, COPY, COPY&COMPARE, and ADD) from tape are picked up by this rebuilding of the queues and are printed in turn.

The taskname that shows in the mix picture for AUTOBACKUP includes the mix number of the job to be output, even if no output unit has yet been selected.

The number of AUTOPRINT and AUTOPUNCH tasks started (exclusive of PB requests) is controlled by AP MAX and CP MAX, respectively, and not by the number of APed units. APed units are treated as preferred units and do not influence the total number of AUTOBACKUPs started.

Setting AP to 0 and then to 1 does not erroneously start outputting the print files for active CANDE jobs.

BACKUPBYJOBNR SYSTEM OPTION

The operator has the option of deciding whether output should be printed in order of job number or by the old method of smallest job first. This system option is invoked by the BACKUPBYJOBNR run-time option.

When this option is set, jobs are printed by order of the job number. When reset, jobs are printed in reverse order of print quantity.

If BACKUPBYJOBNR is set and output from one card reader is directed to a single printer (by APing only that printer or by using the PA message), the output is approximately in the order in which the card decks were entered in the reader. Discrepancies in this order can arise because of one of the following three conditions:

1. AUTOBACKUP does not begin printing a job unless it has gone to EOJ. When other jobs with a higher job number are to be printed, AUTOBACKUP prints the higher jobs rather than waiting for the job still in the mix.

2. When wraparound occurs (that is, the job numbers go back to 0000 from 9999), the low job numbers are printed first.

3. Programmatic setting of various attributes select remote printers or printers with special forms.

Efficiency considerations with BACKUPBYJOBNR come into play if the site turns AUTOPRINT and AUTOPUNCH on and off. When the AP message is manipulated in such a way as to reduce the AP max to 0 and the APed LPs to 0 (or the equivalent for AUTOPUNCH), then the appropriate AUTOBACKUP queue is deallocated. When the AP is again enabled, the MCP must go out to the disk and diskpack subsystem to find everything necessary to print or punch.

This rebuilding of the backup queue is much more efficient if BACKUPBYJOBNR is set. When set, the MCP does not need to access many extra directories to determine the size of the disk and diskpack files in the build queue routine.

Changing BACKUPBYJOBNR when AUTOBACKUP is running and print and punch backup files are queued affects only jobs that subsequently come to EOJ since the old queue is not destroyed. The order of output is somewhat scrambled during this transition because part of the queue is by jobnumber and the other part is by printer backup size. If a site cannot tolerate this interval where part of the files are being printed via a different priority criterion, the situation can be avoided by setting (1) AP to 0 and APed LP to 0; (2) changing BACKUPBYJOBNR; (3) restoring the AP and APed LP to the original value. This process results in the old queue being destroyed and a new queue being built.

When BACKUPBYJOBNR is reset, and two jobs have the same amount of output, the jobs are printed in the order they EOJed.

PRINTERLABELS OPTION

If the system option PRINTERLABELS is reset, then the USASI TAPE LABELS output for all printer files is suppressed. Furthermore, AUTOPRINT attempts to guarantee that every (labeled) print file is separated from the preceding file by one, and only one, page eject. Also, within a file, multiple page ejects are suppressed unless the I/O transfers some data.

FILE PROCESSING

AUTOPRINT attempts to group formed/nonformed files together by making several passes over the BD directory. During each pass, AUTOPRINT outputs all files for the job which have some particular PRINT TRAIN combination. At the start of each pass, if the first file output is either unformed or formed and labeled, then beginning and ending banners are output around the files in the group.

In certain cases (for example, not ready printers), if AUTOPRINT releases the printer and selects another one, it may end up with the same printer. In these cases, the pass described above may be broken up and a new pass started with a different set of forms. (This does not happen if AUTOPRINT is DSed.)

The multiple passes described occur for each job and not for the entire print queue.

Normally, a job has a joblog to be output; thus the first pass prints all unformed files.

A trailing banner similar to the QT banner is printed if the operator DSes AUTOBACKUP.

AUTOBACKUP is QTed if a disk I/O error occurs.

AUTOBACKUP can be QTed while waiting for a disk or pack and while waiting for an output unit (a line printer or card punch).

NOTE

If a backup file has been QTed, then it has been terminated. However, the file does remain in the directory and may be printed later by a PB ODT message. Refer to the Operator Display Terminal Manual, form number 5001704, for further information on the QT command. The PB message is discussed later in this manual.

PACK SPECIFICATION

The system tries to combine all output from one job unless directed to do otherwise by use of task and file attributes (for example, BDBASE in the task attribute OPTION or the FORMMESSAGE file attribute). The backup is printed together provided the pack containing one of the files is not dismounted or powered off.

For more efficiency, the use of named pack should be avoided unless sufficient reason exists. In retrieving files to print from named packs, AUTOBACKUP incurs extra overhead because it must examine every named pack on the system in addition to the system resource packs.

A backup file cannot be directed to an INTERCHANGE mode pack. No printer or punch backup file is allowed to have file attribute INTERCHANGE set to true.

DISK AND DISKPACK SPOOLING

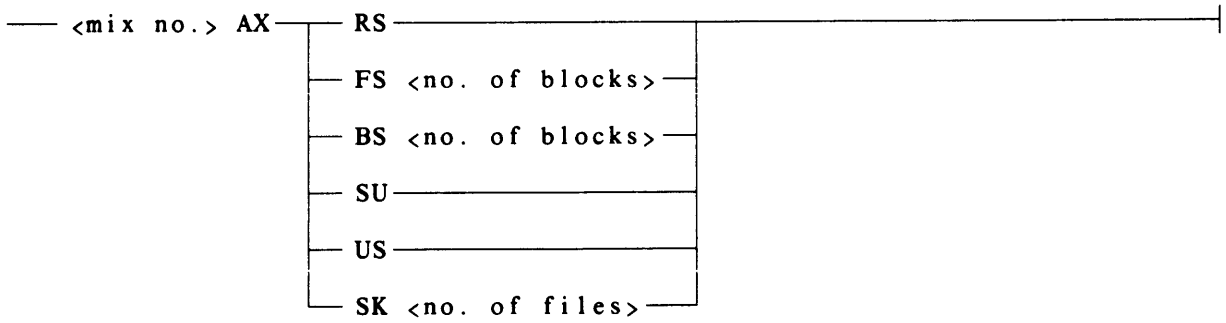
With regards to backup spooled to DISK and DISKPACK, unless the site has AUTOBACKUP turned off or the BDNAME or DESTNAME string task attributes set, backup spooled to DISK and DISKPACK will be automatically queued for printing when the job has finished. After automatic printing, the file is purged.

If it is necessary to prepare a pack backup file for transport to a Burroughs 4700 series computer system, it is necessary to rewrite the backup file from disk or a native mode pack to an interchange pack by writing a program that reads the 300 word blocks of a backup disk/diskpack file and writes an interchange disk file of 300 word blocks.

FILE REPOSITIONING

AUTOBACKUP can be interrupted from its normal operation to reposition the file that is being printed or punched. This repositioning is effected by the use of the ODT AX (accept) message. The following syntax represents the input format.

Syntax



Semantics

RS - Restart file from the beginning

FS - Forward space n blocks from current position

BS - Back space n blocks from current position

SU - Suppress all carriage control (skips and spaces)

US - Unsuppress (turn off carriage control suppression)

SK - Skip n number of files - valid only for backup tape files.

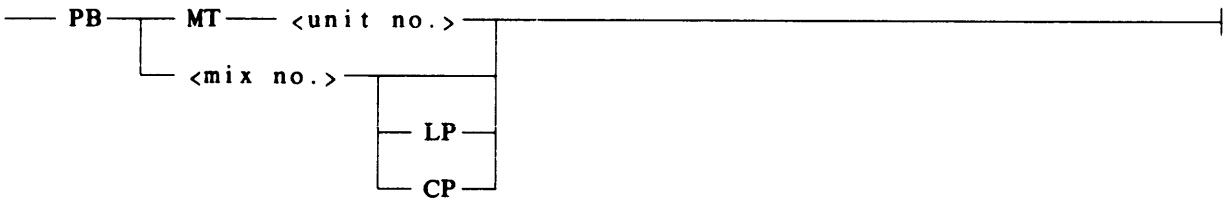
The number of records in a backup file block is variable because the size of the records is variable and can range between 1 and 100. A common average is 13.

PB ODT MESSAGE

The PB message is used to print or punch backup tape or disk files. The entry of the PB message forces at least one copy of AUTOBACKUP into the mix regardless of the AP setting or the number of APed units. It is extremely useful in the event AUTOBACKUP has been QTed, either by the system or by the operator.

BACKUP

Syntax



Semantics

When a PB message is entered specifying a tape unit, that tape is rewound and the backup files on the tape are either printed or punched depending on whether they are printer backup or punch backup files.

When a mix number is specified, that message causes all backup disk files generated by that job and its subtasks to be printed and/or punched. If LP is specified in the message, only printer backup files are output. If CP is used, only punch backup files are output. All other files are left on disk.

Example

PB 697 LP

Action: Prints all printer backup files generated by job number 697 and its subtasks.

Example:

PB MT 17

Action: Either prints or punches all files from the tape on unit 17, depending on whether they are printer backup or punch backup files.

6. RJE BACKUP

Backup files generated by RJE initiated jobs are placed in directories separate from those employed at the main system. Specifically, all printer and punch backup files are placed in their respective REMLP and REMCP directories. Furthermore, when the RJE terminal option AUTOBACKUP is set, the autobackup routine of RJE is processed to output these particular REMLP and REMCP backup files.

A number of similarities do exist between SYSTEM/BACKUP and RJE backup. Consequently, in order to be aware of these similarities, as well as the existing differences, the user should refer to the Remote Job Entry (RJE) System Reference Manual, form number 5001548. Reference should be made specifically to the *DS, *FM, *QT, *RO, *SO, and *TO RJE input messages.

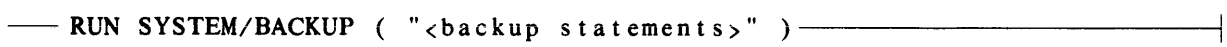
BACKUP

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

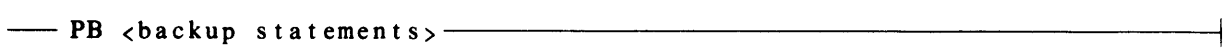
7. SYSTEM/BACKUP STRUCTURE

SYSTEM/BACKUP is an output utility program used to print or punch backup tape or disk files. SYSTEM/BACKUP is invoked by using either the RUN SYSTEM/BACKUP statement or the PB statement of the Work Flow Language. The SYSTEM/BACKUP syntax is given below.

WORK FLOW SYNTAX



OR

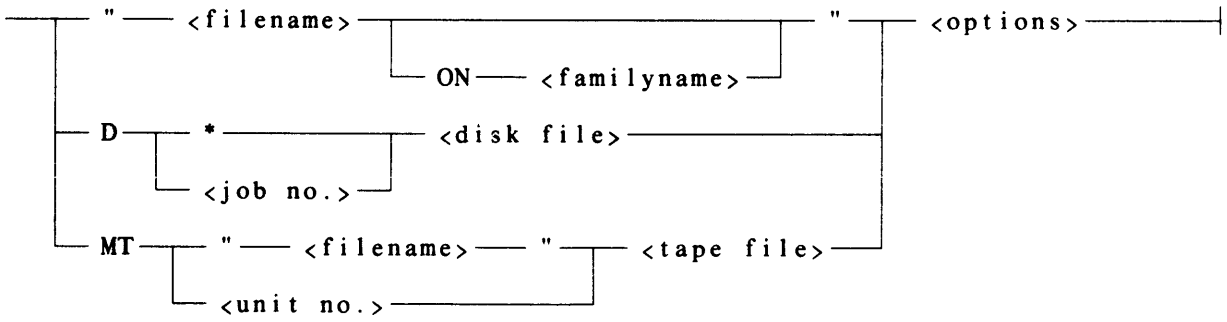


The two syntax diagrams shown above are equivalent; however, a question mark must preceed the second statement when entered through the SPO.

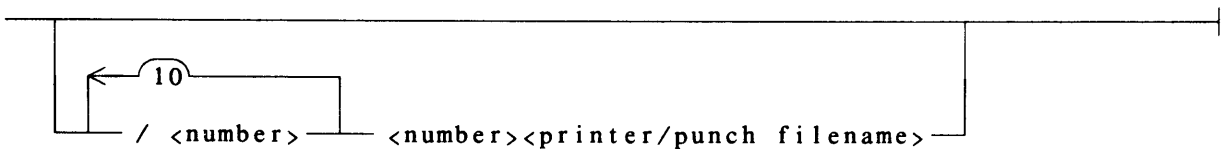
NOTE

Distinct differences between the SYSTEM/BACKUP PB statement and the PB ODT message exist. These differences should be understood before either of these two items of software are used.

<backup statements>

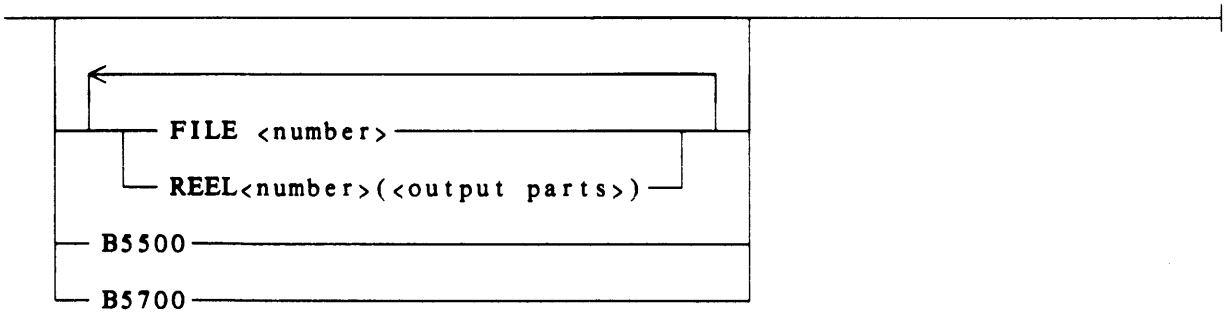


<disk file>

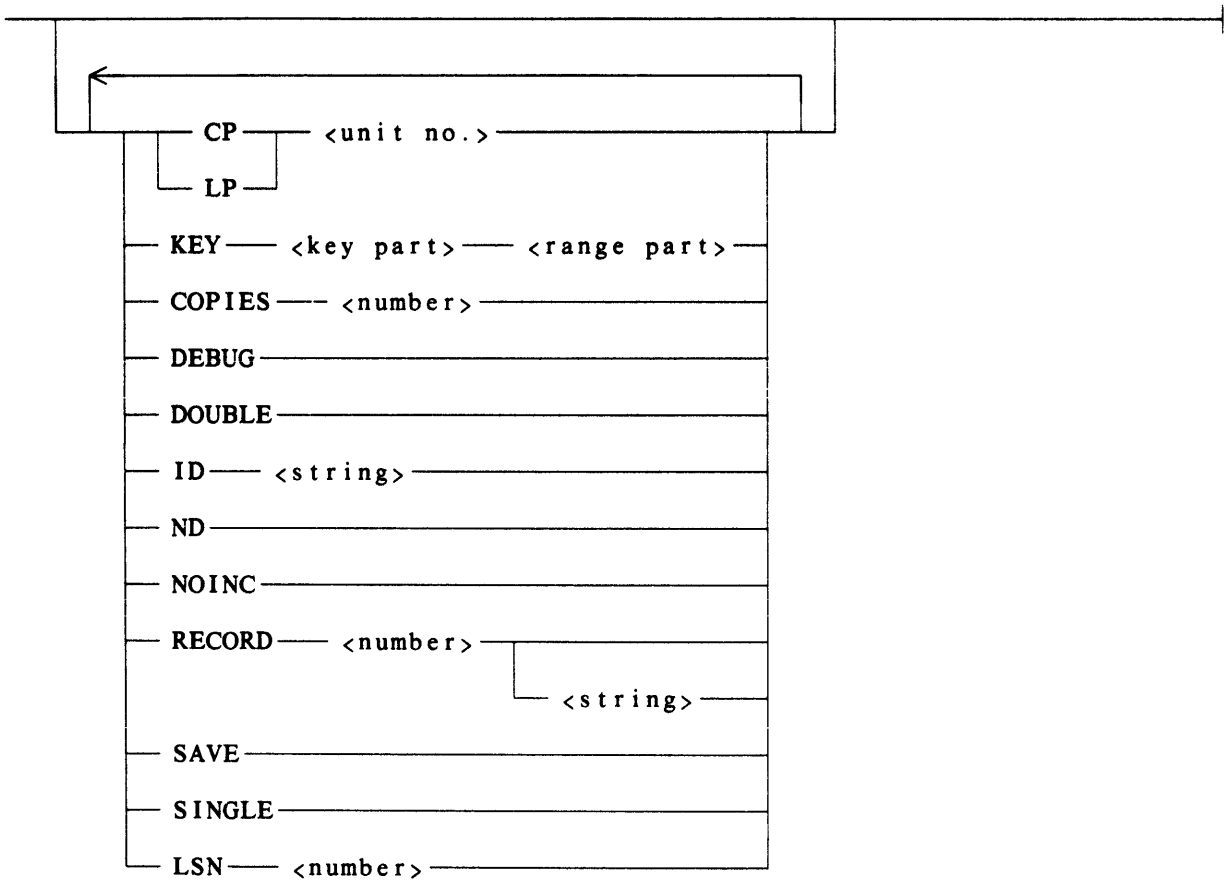


BACKUP

<tape file>

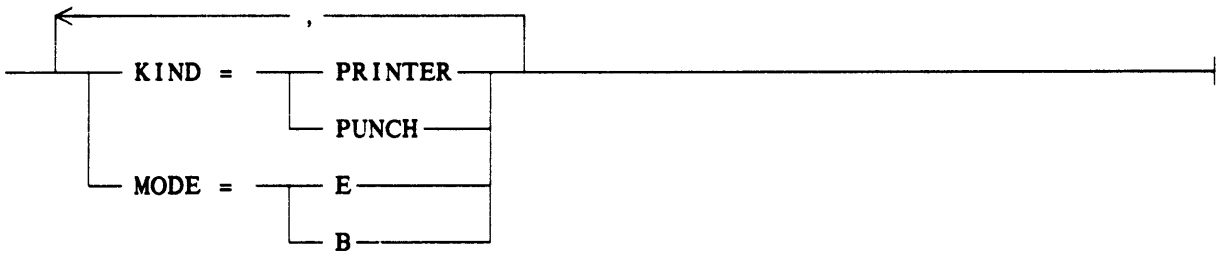


<options>

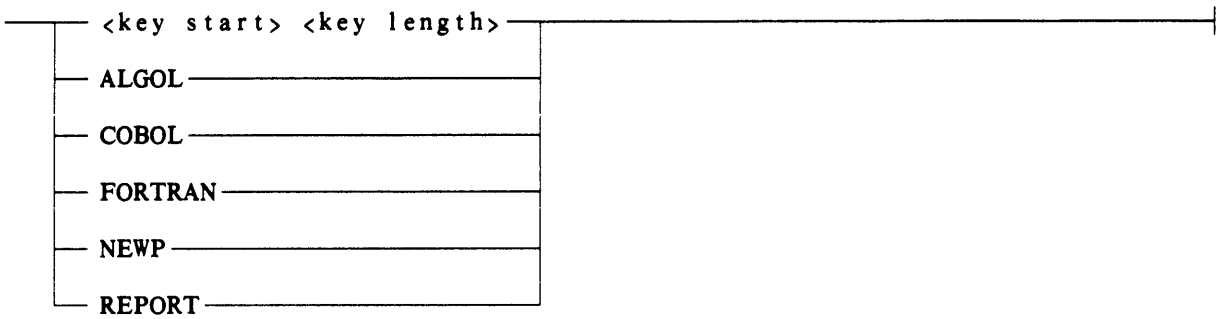


BACKUP

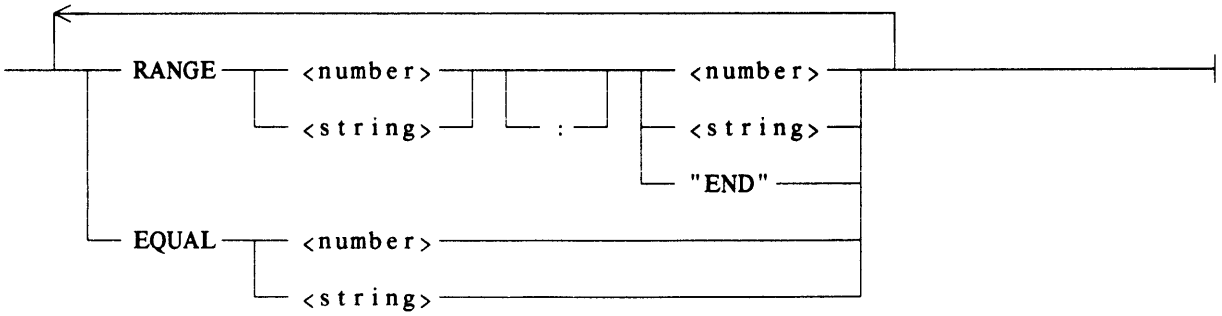
<output parts>



<key part>



<range part>



Semantics

When PBing a printer backup tape (PBT) file, the following SYSTEM/BACKUP PB inputs are available:

* (special character)	Is used in order to print disk files for the mix number of the SYSTEM/BACKUP and is resricted to direct output only. This feature is useful when including a PB card in a WFL deck because it is not possible to know the job number ahead of time.
FILE <number>	Is the number of PBT FILE identified previously by the MT <unit no.> or by <filename>.
REEL <number>	Is the number of the PBT reel serving as SYSTEM/BACKUP input.

KIND	Is the specification that tells whether the PBT is a printer or punch backup.
MODE	Specifies whether the PBT is in EBCDIC or BCL mode.
B5500/B5700	Specifies B 5500 or B 5700, in which case, the PBT on the specified unit is printed. The output goes to a normal (that is, non-direct) printer file called BFILE. Any other options in the input string (RANGE, LP NO., and so forth) are ignored.
<filename>	Is the name of the backup file that is to be printed or punched.
ON <familyname>	If ON <familyname> is not used, SYSTEM/BACKUP searches disk and pack for each file that is to be printed or punched. If the requested file is present on both peripheral families, it is be printed twice. Also, the system family pack DISK is searched, rather than the user's family pack. ON <familyname> causes SYSTEM/BACKUP to print or punch only the file from that <familyname>.

The following semantic discussion deals with the KEY specification and its options.

KEY	Specifies the sequence fields to be used in checking range limits.
<key start><key length>	Are integers that specify the starting column and number of characters in the sequence field to be used in checking range limits.
ALGOL	Is a key specifier indicating the appropriate columns for the ALGOL, DCALGOL, DMALGOL, and ESPOL sequence numbers on compilation listings generated by these compilers.
COBOL	Is a key specifier indicating the appropriate columns for the COBOL sequence numbers on compilation listings generated by this compiler.
FORTRAN	Is a key specifier indicating the appropriate columns for the FORTRAN sequence numbers on compilation listings generated by this compiler.
NEWP	Is a key specifier indicating the appropriate columns for the NEWP sequence numbers on compilation listings generated by this compiler.

The ALGOL, COBOL, and FORTRAN key specifiers are significant only to printer (symbolic) files and therefore, should not be used for punch files. Key lengths are allowed up to 120 characters. If the RANGE specifies a numeric range (for example, RANGE 100 53800000) the numbers are limited to 12 digits regardless of the key length.

REPORT	Uses the columns used by outputs generated by the COBOL Report Writer feature.
RANGE	Denotes the start and stop values of specified keys. For example, if a printer backup file contains the following records

RECORD NO.	CONTENTS
1	AAAA
2	BBBB
3	CCCC
4	AAAA
5	HHHH
6	DDDD
7	ZZZZ
8	DDDD

and the following PB statement is used to print it

PB <filename> KEY 1 4 RANGE "AAAA" "DDDD"

SYSTEM/BACKUP prints every record from the file, beginning with the start value through the stop value. If, however, a key is encountered that is greater than the stop value, that key is considered as the stop value. In this example, SYSTEM/BACKUP prints lines 1, 2, 3, 4, and 5.

The $INCLUDE feature allows the user to specify to SYSTEM/BACKUP that cards which have been included in the listing by such a $INCLUDE card are not to be considered when making the sequence range check but are to be printed if they fall within the desired sequence range. This feature is for use only on listings generated by the ALGOL compiler. It is more useful in cases where included cards cause the symbolic to be out of sequence. This feature is invoked by placing a colon between the sequence range limits as shown in the following example.

PB MT 83 KEY ALGOL RANGE 12340000 : 23450000;

If blanks are found between the numbers, processing is as before. To show how this feature could be useful, consider the following listing:

```
%CARD 1                          00001000
%CARD 2                          10000000
$INCLUDE X 90000000-91000000     10001000
%INCL CARD 1                     90000000
%INCL CARD 2                     90500000
%INCL CARD 3                     91000000
%CARD 3                          10002000
%CARD 4                          10003000
%CARD 5                          20000000
```

If it were desired to print the cards in the sequence range 1000000 - 19999999, the following statement could be used:

<I>PB <device specifier> KEY ALGOL RANGE 10000000 : 19999999;

If backup were run without using this feature, it would stop printing as soon as it hit the card at sequence 90000000.

"END" — Is a range stop indicator for an EBCDIC string (besides either <number> or <string>) which is equivalent to setting the stop integer to 99999999.

EQUAL — Is used to determine if the <number> or <string> options correspond with the character string length or the sequence numbers specified in <key start> and <key length>.

COPIES <number> — Denotes the number of printer backup copies to be printed from a directory.

DEBUG — Is a developmental (subject to change at any time) diagnostic aid intended for debugging purposes.

DOUBLE — Specifies the double-spacing of printer backup files.

ID "string" — Generates the printing of user specified block character headings. This option is valid only when $IDOPTION is set.

ND — Specifies that backup is to use a non-direct file for output. This usage is accomplished by including the parameter ND in the input string. The file used is called BFILE and may be label equated to any allowable medium, such as DISK, PBT, PRINTER, PUNCH, and so forth. If it is equated to a disk file, the file is locked when backup is done. THE * option cannot be used with the ND option.

NOINCL — Specifies that any cards included in a program by an $INCLUDE card are not to be printed. This option is useful only when printing program listings. A KEY must be specified when using this option since backup looks for a digit two characters in front of the sequence numbers.

RECORD <number> — Specifies that the record count will include the three header records plus the one blank record of the file label, if it exists. Thus, printing record 15 of a labeled file that has gone to Backup actually prints record 11 of the file. For example, to obtain the first 20 records after the header and file label records, the bias of four is used (record 5 24).

SAVE	Prevents the purging of all backup files.
SINGLE	Specifies the single-spacing of printer backup files.
LSN <number>	Enables Backup to allow printing RJE files selectively by destination LSN.

Examples of various PB statements are given below.

Example 1

```
PB MT 81 B5500
```

Action: Specifies the printing of B 5500 backup tapes from a specified tape unit onto a normal output (that is, non-direct) printer file called BFILE. Any other options in the input string are ignored.

Example 2

```
PB D   0385 000385/000387/000LINE
```

Action: Specifies that LINE is the title of the printer or punch output file from disk. A prefix of BD or BP is assumed; consequently, SYSTEM/BACKUP constructs the file name by putting BD/ or BP/ first followed by the job number followed by /<file name> exactly as specified.

Example 3

```
PB "TARGET"
```

Action: Specifies that TARGET is the title of a file from disk. The entire file title, including the backup prefix (BD, BP, or whatever was in the BDNAME statement used when creating the file) is enclosed in quotation characters. If this file is a directory, then everything under it is printed.

Example 4

```
PB MT "TEST"
```

Action: Specifies that a tape file called TEST is the input to printer or punch backup. The title of a tape file is simply the name of the printer or punch file used to write it. Therefore, when a PB message is entered which specifies a magnetic tape unit, that particular tape is rewound and the backup files on the tape are printed or punched depending on whether they are printer backup or punch backup files.

Example 5

```
PB D *
```

Action: Prints disk files using the job number under which BACKUP is running.

Example 6

```
PB MT 83 KEY ALGOL RANGE 12340000 : 23450000
```

Action: Indicates that cards included in the printout by a $INCLUDE card are not to be considered when making the sequence range check.

Example 7

PB MT "REAL" CP

Action: Specifies that a tape file called REAL is to be output to punch backup. When LP is specified in the PB message, only printer backup files are output. In contrast to PBing by unit number, SYSTEM/BACKUP prints only the requested file from the tape and does not print the other files on the tape.

SYSTEM/BACKUP allows for the printing or punching of second and subsequent backup reels without having to read through the first reel.

When tape backup is required, the MCP looks for an available backup tape. If one is available, the file is written on that tape. If an uptape backup tape is required and is not available, the MCP looks for a scratch tape. If a scratch tape is available, the system designates it as a backup tape and writes the printer or punch file on it.

Example 8

PB MT83 FILE 2

Action: Causes SYSTEM/BACKUP to begin printing at the second file on the tape on unit 83. Uptape files on multi-file reels can be printed without having to print all of the preceding files. The number of the first file to print is specified to SYSTEM/BACKUP by including FILE <number> in the input string.

BACKUP

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTNG PURPOSES.

8. SYSTEM/BACKUP COMPILE-TIME OPTIONS

$IDOPTION

The $IDOPTION is a compile time option used to determine whether block character headings are printed on SYSTEM/BACKUP printer output. When $IDOPTION is set and the backup files to be printed are located on DISK and PACK, the titles of the backup files contained on these media are printed in block characters. Also, when $IDOPTION is set and the backup files to be printed are located on TAPE, the following message is printed on the listing:

"BACKUP TAPE - UNIT 011"

The number of the tape unit containing that particular backup file is 011.

When the $IDOPTION is set and the ID "string" option is specified in SYSTEM/BACKUP, the user-provided ID "string" of characters (with a maximum length of 60 characters) is printed in block characters.

The ID "string" option does not work with the ND option. For example,

PB "B" ND ID "BBB"

generates a syntax error and, as such, is invalid in SYSTEM/BACKUP. The implementation was not designed to allow non-direct (ND) I/O with the block character headings.

When the $IDOPTION is reset, block characters are not printed. The $IDOPTION must be set if any block character headings are to be printed.

$INFOPTION

$INFOPTION is a compile time option that allows the user to obtain information (for example, the version number and the file printed) displayed onto the console; subsequently, that same information is printed on the WFL output when set at compile time. This option prevents SYSTEM/BACKUP from requiring two printers, except if DEBUG is set or an error condition occurs. The default does not have $INFOPTION set; therefore, it still uses two printers.

BACKUP

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

9. TAPE REPOSITIONING

Tape Repositioning is a feature used to handle cases of printer jams or failures during long printer runs, in which several lines or pages of output from a Printer Backup tape have been lost. The operator may inform SYSTEM/BACKUP to reposition the tape by entering:

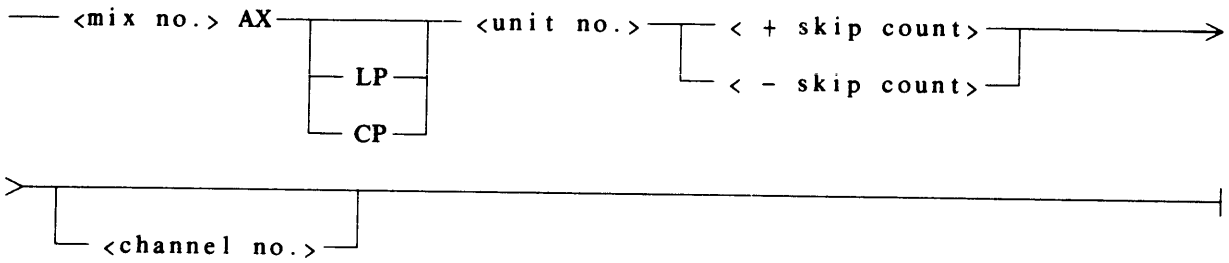
<mix no.> HI

at the console. Backup responds

ACCEPT: ENTER SKIP OR UNIT NO.

The syntax for the reply is given below.

Syntax



NOTE

Tape repositioning cannot be used with non-direct(ND) files.

Semantics

If the LP or CP followed by a unit number is present, backup begins using this unit instead of the one it was previously using.

The first number specified is a skip count. The number may be either positive or negative, indicating skipping forward or backward, respectively. If a negative skip count is entered, the tape is positioned past BOT; SYSTEM/BACKUP starts printing from the beginning of the tape.

If nothing follows the skip count, then the count is assumed to refer to a number of lines.

Example

<mix no.> AX -3

Action: Back up three lines and resume printing.

If the skip count is followed by a number between 1 and 11, the number is interpreted as a channel number, and the skip count refers to skips to that particular channel.

Example

<mix no.> AX -4 3

Action: Back up four skips to channel three.

<mix no.> AX 1 1

Action: Move forward to the next skip to top of page (CHANNEL 1).

NOTE

A skip count must always be present; however, if another unit number is specified, the skip count is not required.

CARDLINE

TABLE OF CONTENTS

1. INTRODUCTION

The SYSTEM/CARDLINE utility is used to printout or punch a BCL, EBCDIC, or BINARY data deck. In addition to a printout of the card images, a card count and sequence check are included. Columns 73-80 are checked for sequence errors.

In addition to the card-to-print function, other utility functions can be accomplished by label equating the input and/or output files of the program. The input file is named CARD, and the output file is named LINE.

CARDLINE

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

2. PRINTING

Examples

1. To list a card deck with EBCDIC data, use the following:

```
<I> BEGIN JOB CARDLINE;
    RUN SYSTEM/CARDLINE; VALUE = <integer>;

    EBCDIC

    .

    <data deck>

    .

    .

<I>  END JOB
```

2. To list a card deck with BINARY data, use the following:

```
<I> BEGIN JOB CARDLINE;
    RUN SYSTEM/CARDLINE; VALUE = <integer>;

    BINARY

    .

    <binary data>

    .

    .

BEND card

<I>  END JOB
```

Pragmatics

The VALUE = <integer> clause appearing in the examples above is used to specify spacing between output lines.

In example 1, <integer> must be a numeric value of 0 through 9, inclusive. This range of values is used with EBCDIC files and serves to specify the spacing between output lines. The values 0 and 1 cause single spacing. An <integer> greater than 1 and less than 10 causes the specified <integer> number of lines to be spaced.

In example 2, <integer> must be a numeric value of 10 to 19, inclusive. This range of values is used with binary files and serves to specify the spacing between output lines. The values 10 and 11 cause single spacing. An <integer> greater than 11 and less than 20 causes the specified <integer> number of lines (MOD 10) to be spaced.

CARDLINE

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

3. PUNCHING

Examples

1. To punch a BCL, EBCDIC, or BINARY card deck, the LINE file is equated to a card punch as follows:

```
<I>  BEGIN JOB CARDLINE;
     RUN SYSTEM/CARDLINE;
     FILE LINE (KIND=PUNCH);
     BCL

     .

     <data deck>

     .

     .

<I>  END JOB
```

2. To list a card deck with binary data, use the following:

```
<I> BEGIN JOB CARDLINE;
    RUN SYSTEM/CARDLINE;
    FILE LINE (KIND = PRINTER);
    BINARY CARD

    .

    <data deck>

    .

BEND card

<I> END JOB
```

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

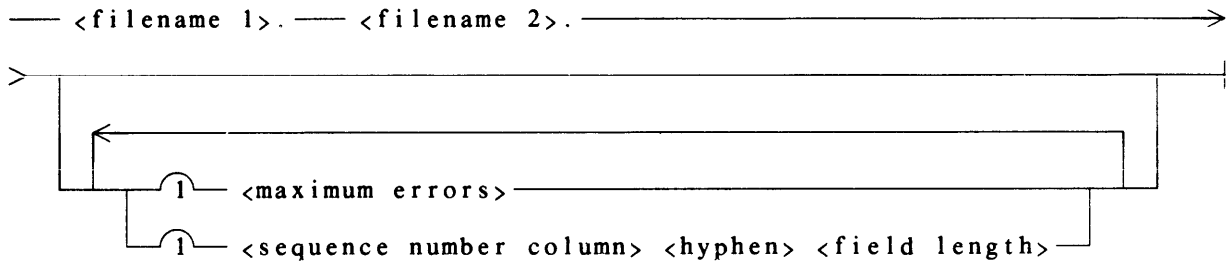
COMPARE

TABLE OF CONTENTS

1. INTRODUCTION

SYSTEM/COMPARE compares one or more pairs of files. The utility performs a bit-by-bit comparison on each record or sequence number and record of each pair of files. If the sequence numbers and/or records are not identical or if one of the specified files is not present, an appropriate error message is printed. The comparison of a pair of files is terminated after a specified number of unsuccessful comparisons have been made, and the utility proceeds to the next pair of files.

Syntax



Semantics

EBCDIC, BCL, ASCII, and HEX disk files can be compared. Input specifications are in free-field format. The <filename> must be followed by a period. The <maximum errors> default is five. The <sequence number column> is the column in which the sequence numbers of the files begin. The <sequence number column> is followed by a hyphen (-), which is followed by the field length of the sequence numbers.

If no sequence information is specified, the files are compared record by record; a new record is read from each file for each comparison. In sequenced files, if a difference occurs, both records are printed. If the files are sequenced, and the sequence numbers agree, but the records do not, the contents of both records are printed. If the record sequence number of the first file is greater than the record number of the second file, the record from the second file is printed, and the next record from the second file is compared against the first, and vice versa. If a difference occurs when comparing unsequenced files, the record number at which the difference occurred is printed.

Example

```
<I> BEGIN JOB COMPARE;
    RUN SYSTEM COMPARE;
    DATA
    PROGRAM/ONE.  PROGRAM TWO.
    PROGRAM/THREE.  PROGRAM/FOUR.  73-8
    PROGRAM/FIVE.  PROGRAM/SIX.  25
<I> END JOB
```

2. OUTPUT FROM COMPARE

The output listing contains the following information:

1. A description of the two files being compared, which includes the MAXRECORDSIZE, BLOCKSIZE, UNITS, INTMODE, and CREATIONDATE. If the file is not in the directory, an error message is printed.

2. If the files differ in blocking specifications (UNITS, BLOCKSIZE, MAXRECORDSIZE), a message is printed, and no comparison is made.

3. The maximum error default is listed.

4. If sequence information is specified, it is printed; otherwise, a message is printed stating that unsequenced files are assumed.

5. If any differences occur, a list of the differences is printed.

6. If the comparison of the files is terminated because the maximum error default has been reached, a message is printed along with the current record number of each file being compared.

7. If an end-of-file condition occurs in one file before the other, a message is printed. The message also indicates in which file the end-of-file condition occurred.

8. The number of differences is listed.

9. The number of records in each file is given.

10. A "total page" is printed providing the information described above, except the list of differences.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

TABLE OF CONTENTS

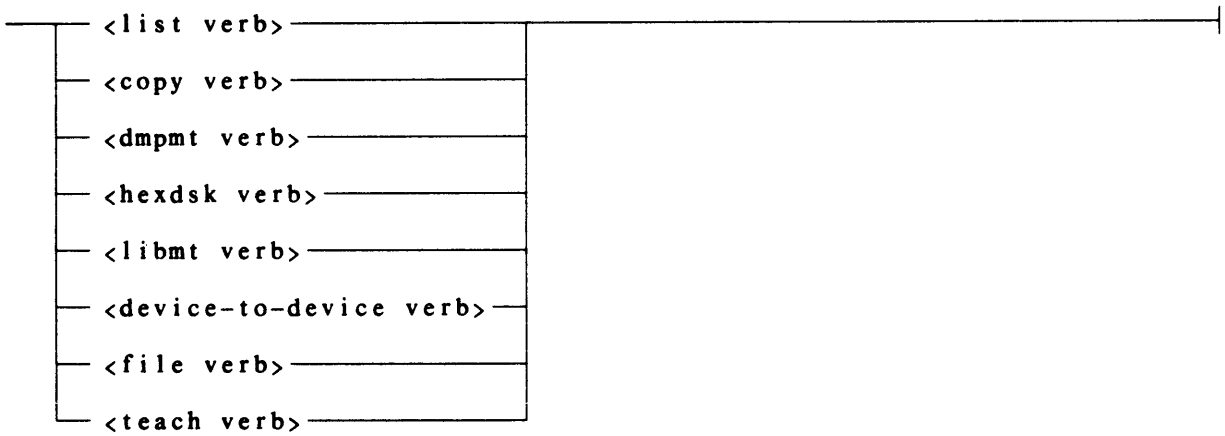
1. INTRODUCTION

SYSTEM/DUMPALL is a utility program that generates printouts of files, controls the dumping of tapes, and provides for the copying of files from one media to another.

Syntax

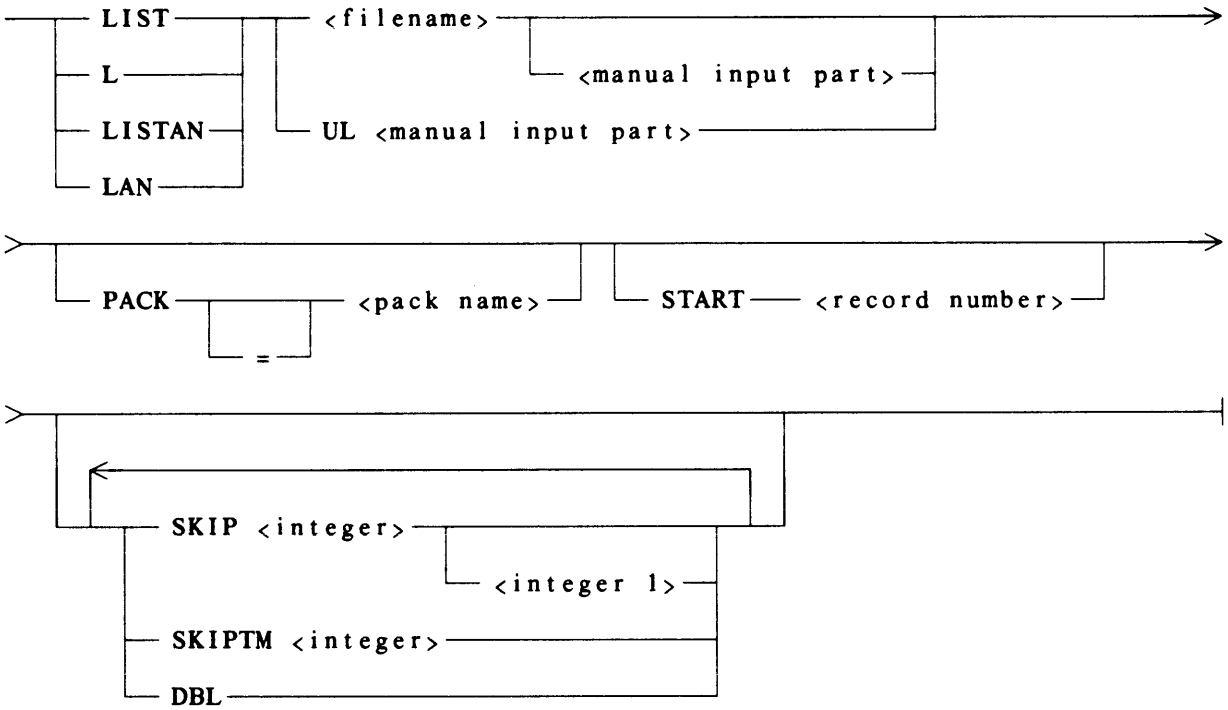
DUMPALL

<control option>



<list verb>

Syntax



Semantics for LIST or L

The LIST option provides a graphic printout of the contents of a labeled or unlabeled file. If the file is titled, the required file parameters may be specified; otherwise, the titled file obtains its parameters from the file itself. If the file is unlabeled (UL), the record/character mode (S, N, E, B), the <maxrecsize>, and the <blocksize> attributes are required. Record/character mode S (STANDARD) refers to XALGOL, BCL, variable-length records (FILETYPE=5); N (NONSTANDARD) refers to alpha or even-parity seven-track (PARITY=1) or paper tape records; E refers to EBCDIC and B to BCL fixed-length records. CHAR defines <maxrecsize> and <blocksize> to be a specified number of characters. The default is words. SKIP <integer> specifies the record number +1 at which printing begins, and <integer 1> specifies the number of records to print. Only the last SKIP specification is used. SKIPTM <integer> specifies the number of tapemarks to be skipped.

Examples

```
. . . ("LIST IN/FILE")
. . . ("L IN/FILE SKIP 306")
. . . ("L IN/FILE SKIP 190 25")
. . . ("L IN/FILE PACK= INPACKNAME")
. . . ("L IN/FILE DBL")
. . . ("LIST UL E 80 80 CHAR SKIPTM 6")
. . . ("L UL S 10 56")
. . . ("L UL B 80 2400 CHAR")
. . . ("L UL N 80 2400 CHAR")
. . . ("L UL E 14 420 SKIP 185 25")
```

In the last example above, the file to be listed is unlabeled, EBCDIC, 14 words per record, 420 words per block, printout starts at the 186th record, and 25 records are to be printed.

Semantics for LISTAN or LAN

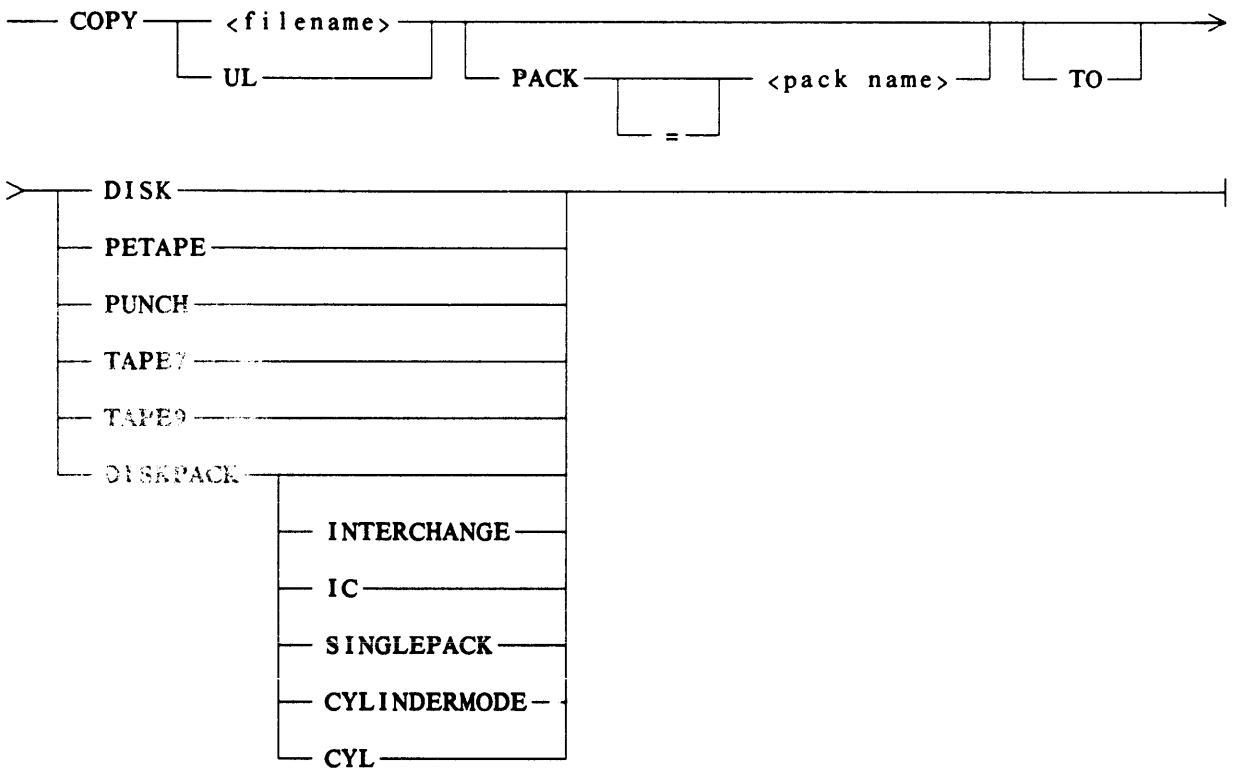
The LISTAN option provides a graphic printout of an unlabeled or labeled file in EBCDIC or hexadecimal characters. LISTAN has the same attributes as the LIST option.

Examples

```
. . . ("LISTAN IN/FILE")
. . . ("LAN IN/FILE")
. . . ("LAN UL E 80 80 CHAR SKIPTM 3 DBL")
```

<copy verb>

Syntax



Semantics

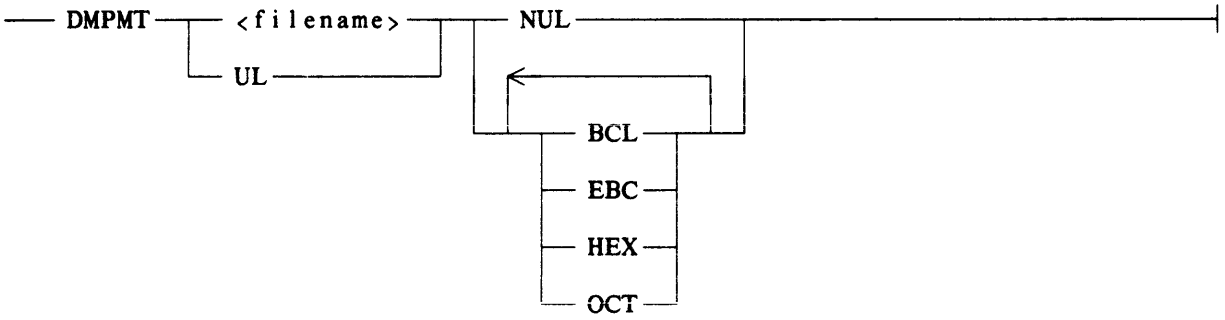
The COPY option allows files to be copied to a specified <device>. For an unlabeled file, the attribute requirements are identical to the LIST option. When using DISKPACK, native mode is assumed CYLINDERMODE, SINGLEPACK, and INTERCHANGE default to false.

Examples

```
. . . ("COPY IN/FILE TO DISK")
. . . ("COPY IN/FILE TO DISKPACK")
. . . ("COPY IN/FILE TO DISKPACK SINGLEPACK CYLINDERMODE")
. . . ("COPY UL E 14 70 TO PETAPE")
. . . ("COPY UL E 84 5880 CHAR TO PETAPE")
```

<dmpmt verb>

Syntax



Semantics

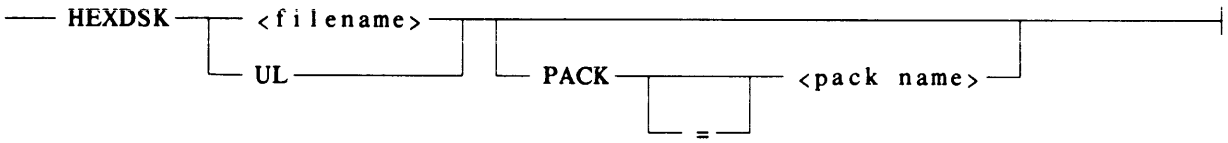
The DMPMT option causes a tape dump in EBCDIC, BCL, HEX, and/or OCTal. The NUL attribute returns only the tapemarks, record size in words and characters, and resulting descriptors.

Examples

```
. . . ("DMPMT UL")
. . . ("DMPMT INFILE")
. . . ("DMPMT UL HEX")
. . . ("DMPMT UL EBC HEX BCL OCT")
```

<hexdsk verb>

Syntax



Semantics

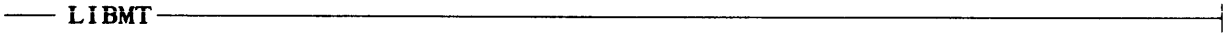
The HEXDSK option lists a file in hexadecimal and EBCDIC. The file is read with a <maxrecsize> and <blocksize> of 30 words.

Example

```
. . . ("HEXDSK INFILE")
```

<libmt verb>

Syntax



Semantics

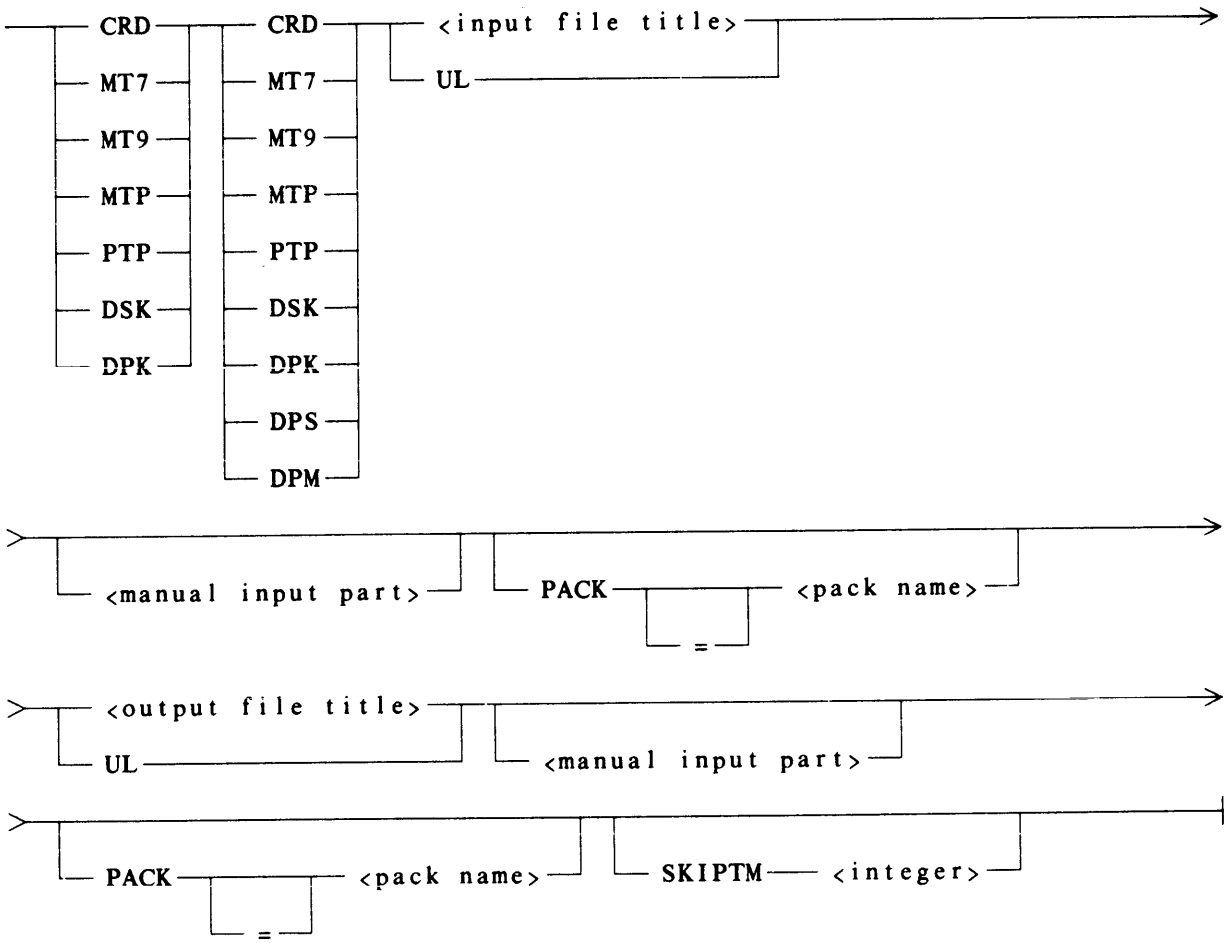
The LIBMT option lists a library tape in hexadecimal. The tape must be ULed in (the UL message).

Example

```
. . . ("LIBMT")
```

<device-to-device verb>

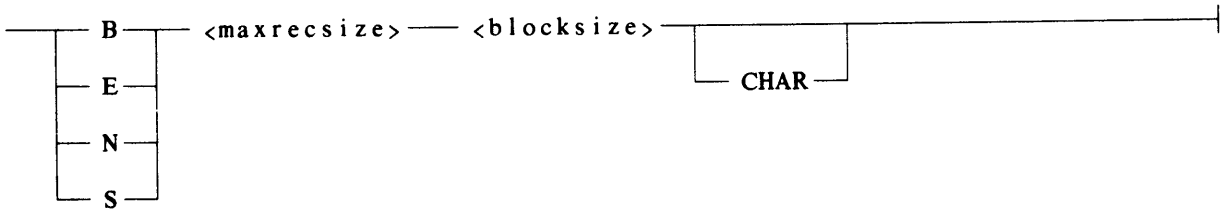
Syntax



CRD = card reader/punch
MT7 = 7-track magnetic tape
MT9 = 9-track magnetic tape
MTP = PE magnetic tape
PTP = paper tape punch

DSK = disk
DPK = disk pack
DPS = disk pack single pack
DPM = disk pack multipack

<manual input part>



Semantics

The <device-to-device> option allows the movement of files from hardware device to hardware device. The input and/or output file can be labeled or unlabeled. If an input tape is unlabeled, the SKIPTM attribute can be used prior to the move. Input and output file parameters are used if supplied; otherwise, input file parameter values are obtained via file attributes, and output file parameters are those used for the input file. If the input file is empty, the output file is not created.

Examples

```
. . . ("DSKDSK IN/FILE OUT/FILE")
. . . ("MT9MT9 IN/FILE E 14 420 OUT/FILE")
. . . ("CRDDSK IN/FILE OUT/FILE E 14 420")
. . . ("DSKDPK IN/FILE OUT/FILE PACK = OUTPACKNAME")
. . . ("CRDDPS IN/FILE OUT/FILE")
```

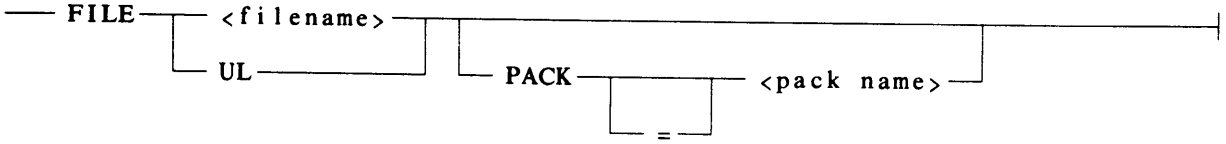
In the example above, if pack is an IC pack(interchange), the "OUT" of OUT/FILE corresponds to the diskpack name.

```
. . . ("DSKDSK IN/FILE OUT/FILE B 10 150")
. . . ("MT9DSK IN/FILE E 80 1200 CHAR OUT/FILE E 14 420")
. . . ("MTPDSK UL E 14 420 OUT/FILE B 80 2400 CHAR")
. . . ("DPKDSK IN/FILE PACK = INPACKNAME OUT/FILE")
. . . ("MTPDPM UL E 80 80 CHAR PACKNAME/FILE E 80 1200 CHAR
      CYL SKIPTM 1")
. . . ("MT7MT9 UL B 10 150 NEW/TITLE E 14 420 SKIPTM 1")
```

In the last example, the file is to be copied from seven-track tape to nine-track tape, the input file is defined as unlabeled, the data written in BCL, <maxresize> is 10 words, and <blocksize> is 150 words. The output file, NEW/TITLE, is to be written in EBCDIC with a <maxrecsize> of 14 words and a <blocksize> of 420 words. Before copying of the file begins, one input tapemark is to be skipped.

<file verb>

Syntax



Semantics

The FILE option lists the values of the initial file attributes, the file attributes after a RESIDENT test, and the file attributes after a PRESENT test. The attributes listed are as follows:

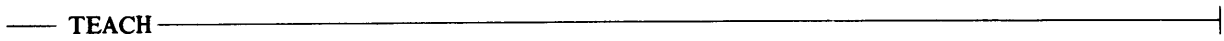
KIND, EXTMODE, INTMODE, MAXRECSIZE, MINRECSIZE, BLOCKSIZE,
UNITS, PARITY, FILETYPE, AREAS,
AREASIZE, ROWSINUSE, LASTRECORD, FILEKIND, and CREATIONDATE.

Example

. . . ("FILE IN/FILE")

<teach verb>

Syntax



Semantics

The TEACH option causes a printout of the information presented above.

Example

. . . ("TEACH")

DUMPALL

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

TABLE OF CONTENTS

1. INTRODUCTION

SYSTEM/FILEDATA, a parameter-driven utility program, produces selected reports regarding files. The reports provide:

1. A hierarchical list of files.
2. A map of files showing their storage layout.
3. A disk checkerboard displaying permanent files and the space around them.
4. The status of all head-per-track (HPT) disk.
5. Specified attributes of a file or a group of files.
6. A list of file names contained in a tape directory.
7. A file suitable for use in a library maintenance copy deck.
8. A raw (HEX) dump of disk file headers.
9. A list of the catalogue information about a file.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

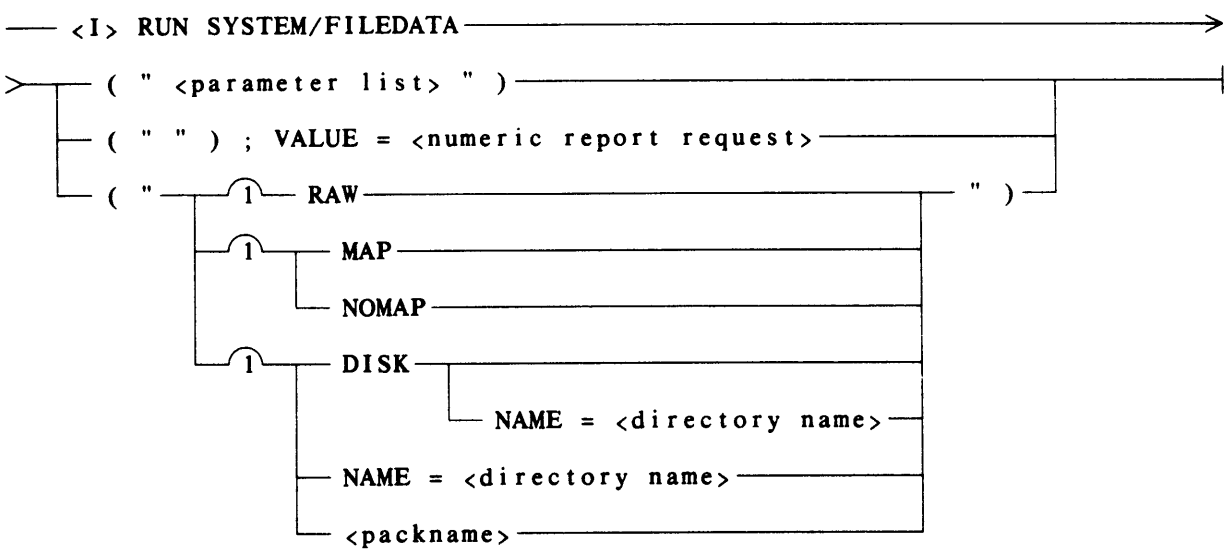
2. FILEDATA EXECUTION METHODS

The FILEDATA utility program may be executed in three ways: (1) by ODT commands, (2) by a WFL card deck, or (3) through a CANDE terminal by the use of WFL commands or the CANDE LFILES command. (Refer to the B 7000/B 6000 Series CANDE Reference Manual, form 5010259, for a description of the LFILES command.)

The following syntax diagrams illustrate how SYSTEM/FILEDATA is executed using WFL statements and ODT commands. Additional syntax diagrams describing frequently used syntactic elements follow the WFL and ODT diagrams.

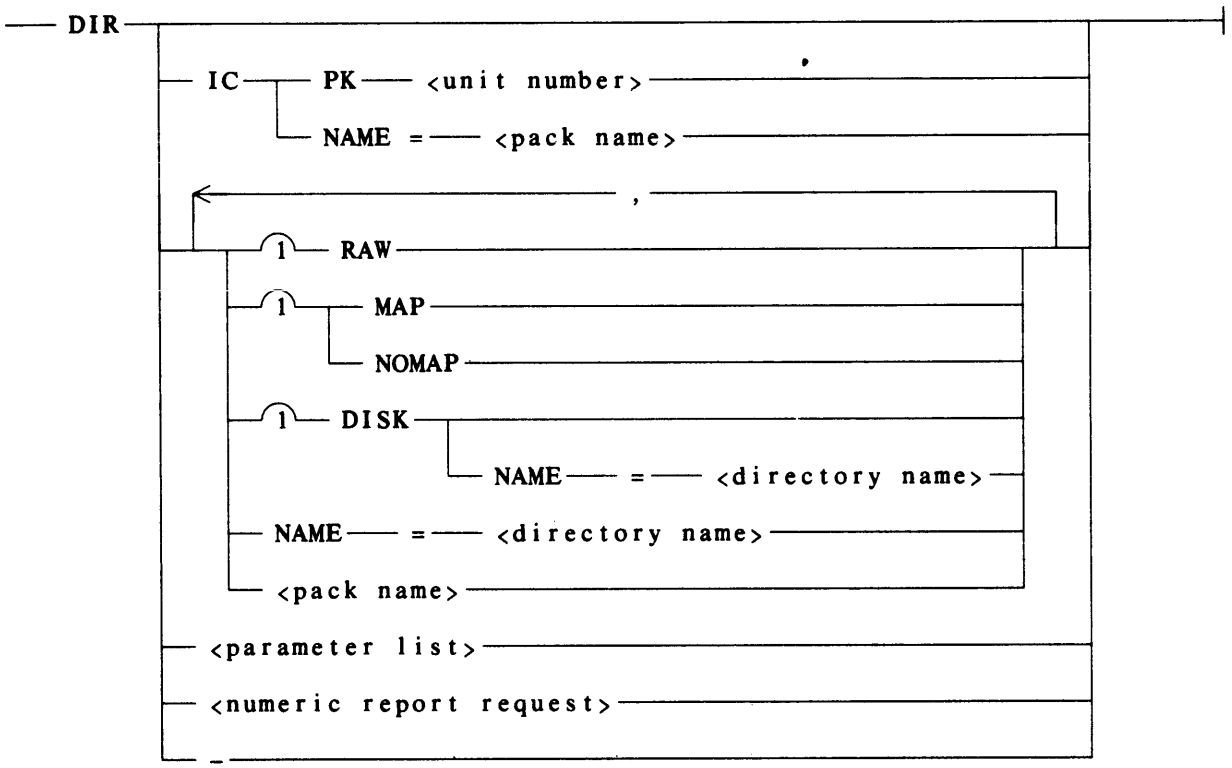
FILEDATA

WFL Syntax



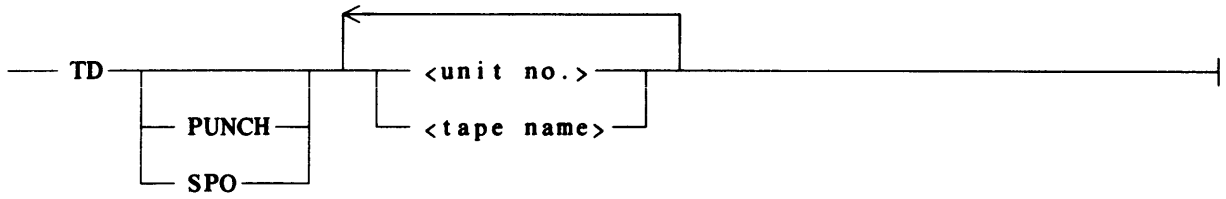
ODT Syntax

DIR

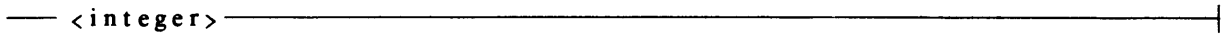


FILEDATA

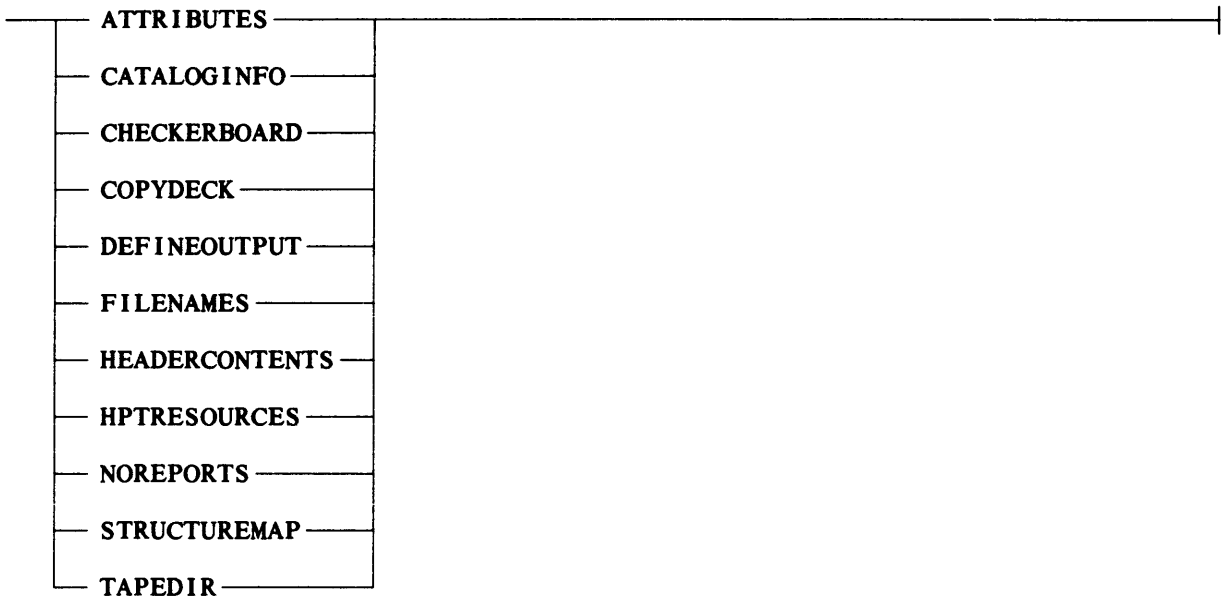
TD



<unit no.>



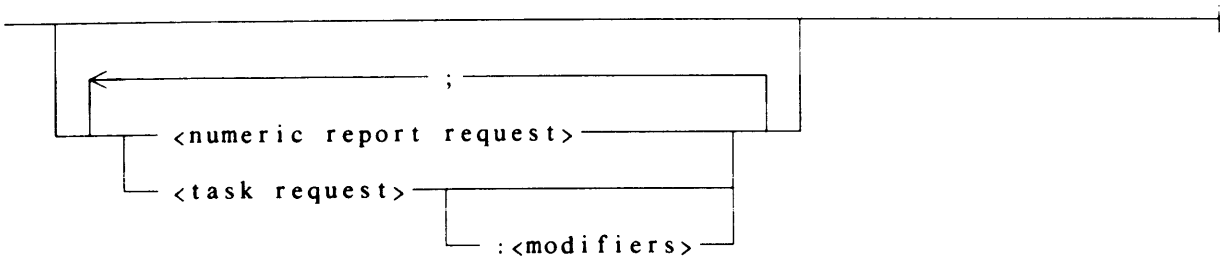
<task request>



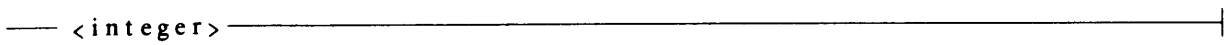
<tape name>

FILEDATA

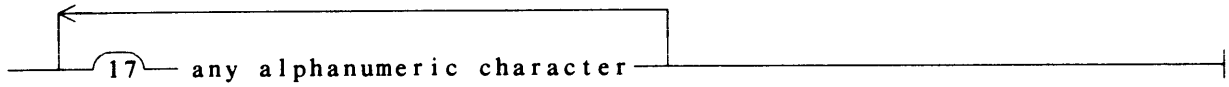
<parameter list>



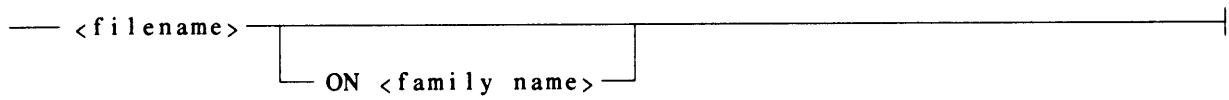
<numeric report request>



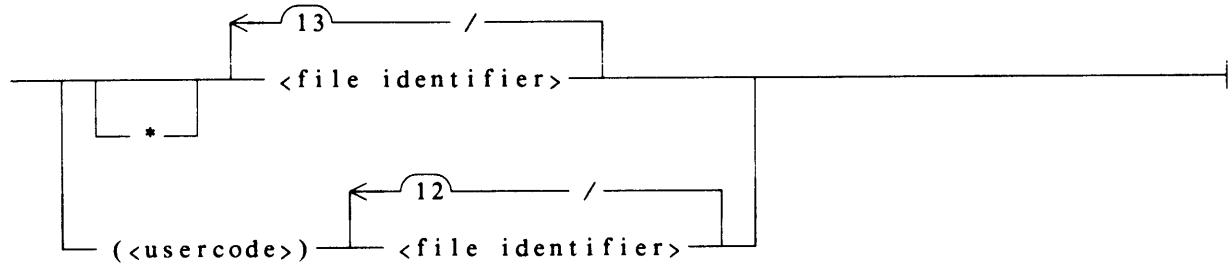
<identifier>



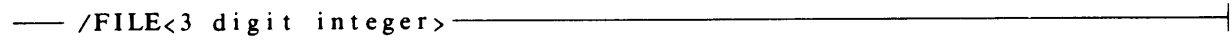
<file title>



<file name>

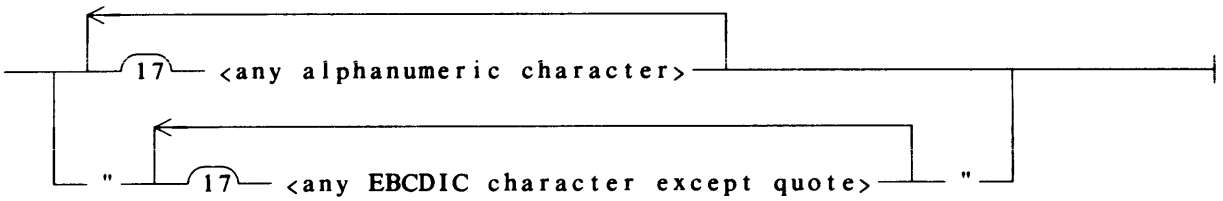


<file no.>

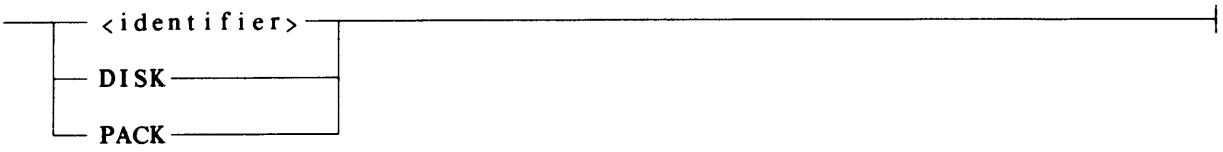


FILEDATA

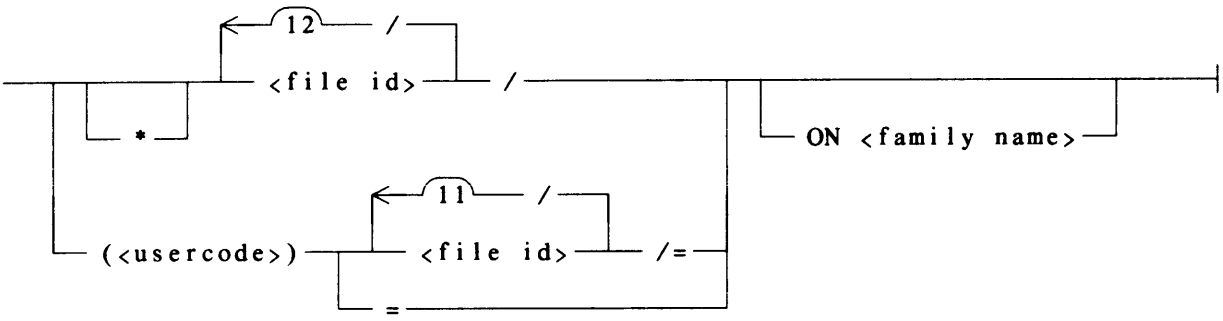
<file identifier>



<family name>



<directory name>



<file id> is a <file identifier>.

<directory name> is a subset of a <filename>. For example, given the <filename> A/B/C/D:

A/=, A/B/=, A/B/C/=

are the only valid <directory name>s.

Examples:

```
<I>RUN SYSTEM/FILEDATA ("FILENAMES:LEVEL=2 TITLE =SYMBOL")
<I>END

<I>RUN SYSTEM/FILEDATA ("1;ATTRIBUTES:DIR=MYSELF ALL;0")
<I>END

<I>RUN SYSTEM/FILEDATA(" ");VALUE=1
<I>END

DIR 1
DIR COPYDECK: CATALOGUE; CHECKERBOARD; 0
DIR-

TD 115
TD SPO TIO
TD PUNCH TIO/FILE001
```

Semantics

DIR and TD are ODT commands that run SYSTEM/FILEDATA.

DIR is used to obtain disk directory information and TD is used for library tape directory information.

The default destination of a FILEDATA generated report is the line printer. The default destination may be overridden by including the key word SPO or PUNCH. SPO causes the output to be directed to the requesting terminal; PUNCH causes the punching of a COPY&COMPARE card deck (with no FROM or TO part) which can subsequently be used by library maintenance.

FILEDATA can report on the status of a tape using the TD ODT command or the TPDIR <task request>. The desired tape may be specified by tape name or by drive number. The latter method must be used if the nth reel of a multireel library dump is required.

Using the SPO option, each tape is reported, in turn, in the order the requests were entered. Entering "△QUIT" in place of "△NEXT" (each occurs at the pause for a new page) causes the program to be terminated, possibly with some still unreported tapes.

If the head of the input string is not a number or a <task request>, the input is processed as PACKDIR. PACKDIR accepts only those keywords shown below. Anything else is treated as a pack name. The reports include the FILENAMES and STRUCTUREMAP reports and, optionally, the CHECKERBOARD and HEADERCONTENTS reports.

DISK: Specifies that the directory to be listed resides on HPT disk. Default is native mode disk packs.

MAP: Specifies that the report is to include a sorted listing of allocated disk segments (CHECKERBOARD). The default is NOMAP.

NOMAP: Suppresses listing of allocated disk segments (default value).

RAW: Specifies that each header is to be printed in HEX.

NAME: Specifies the (qualified) name of the directory to be listed. For disk packs, the first level of the name must be the pack name. No blanks, quotes, or parens may occur anywhere in the name.

Examples:

```
RUN SYSTEM/FILEDATA("DISK MAP RAW")
```

is equivalent to

```
RUN SYSTEM/FILEDATA("FILENAMES;STRUCTUREMAP;CHECKERBOARD;
    HEADERCONTENTS")
```

```
RUN SYSTEM/FILEDATA("PACK1 MAP")
```

is equivalent to

```
RUN SYSTEM/FILEDATA("FILENAMES:PACKNAME=PACK1;STRUCTUREMAP;
    CHECKERBOARD")
```

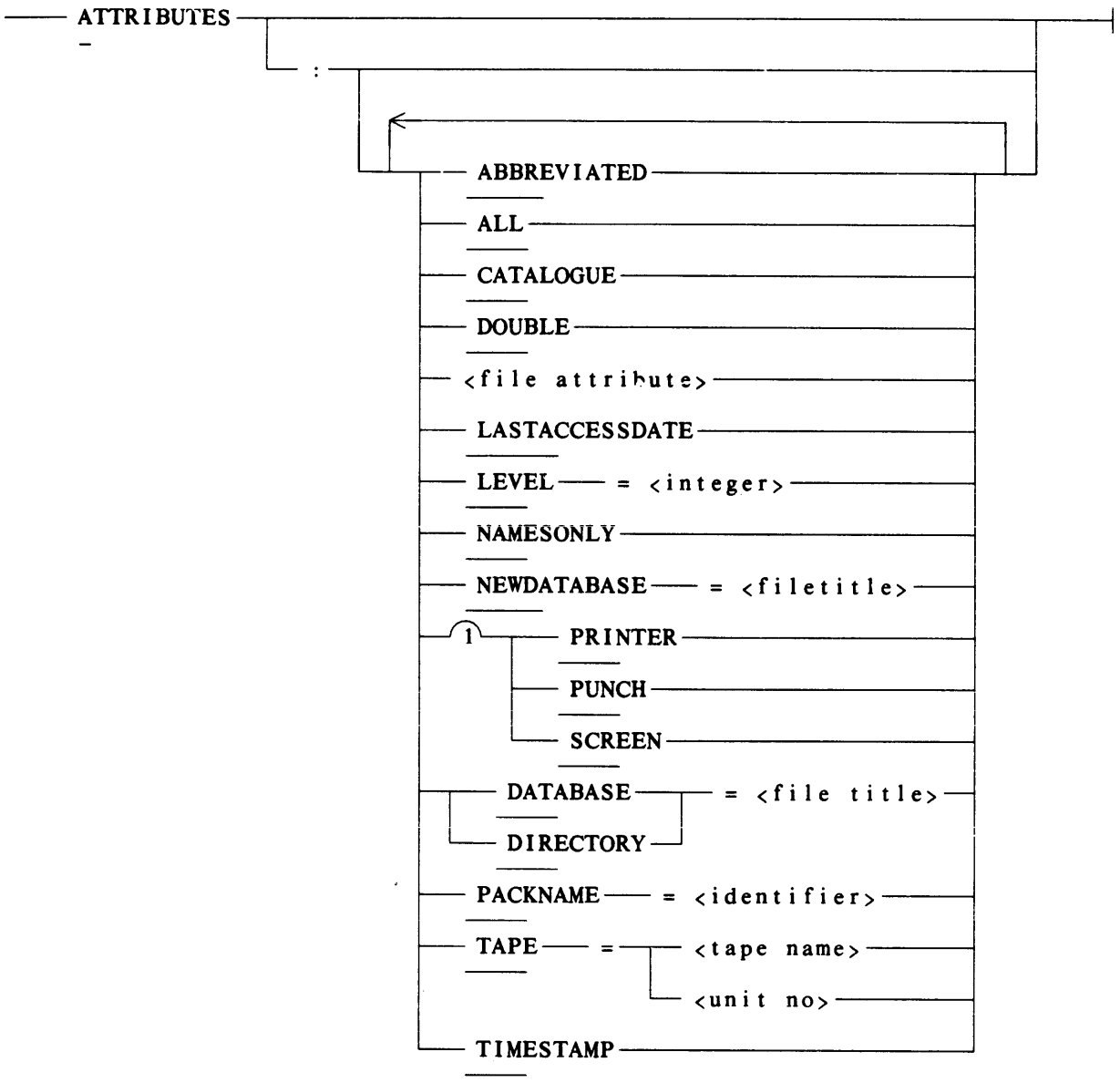
THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

3. TASK REQUESTS

In the following discussions of <task request>s, certain key words are listed. A minimum abbreviation exists for each key word. Beyond the minimum, additional letters may be used up to and including the entire word. If additional letters are used their spelling must be correct. The minimum keyword abbreviation is indicated in the syntax diagram by the appearance of underscores.

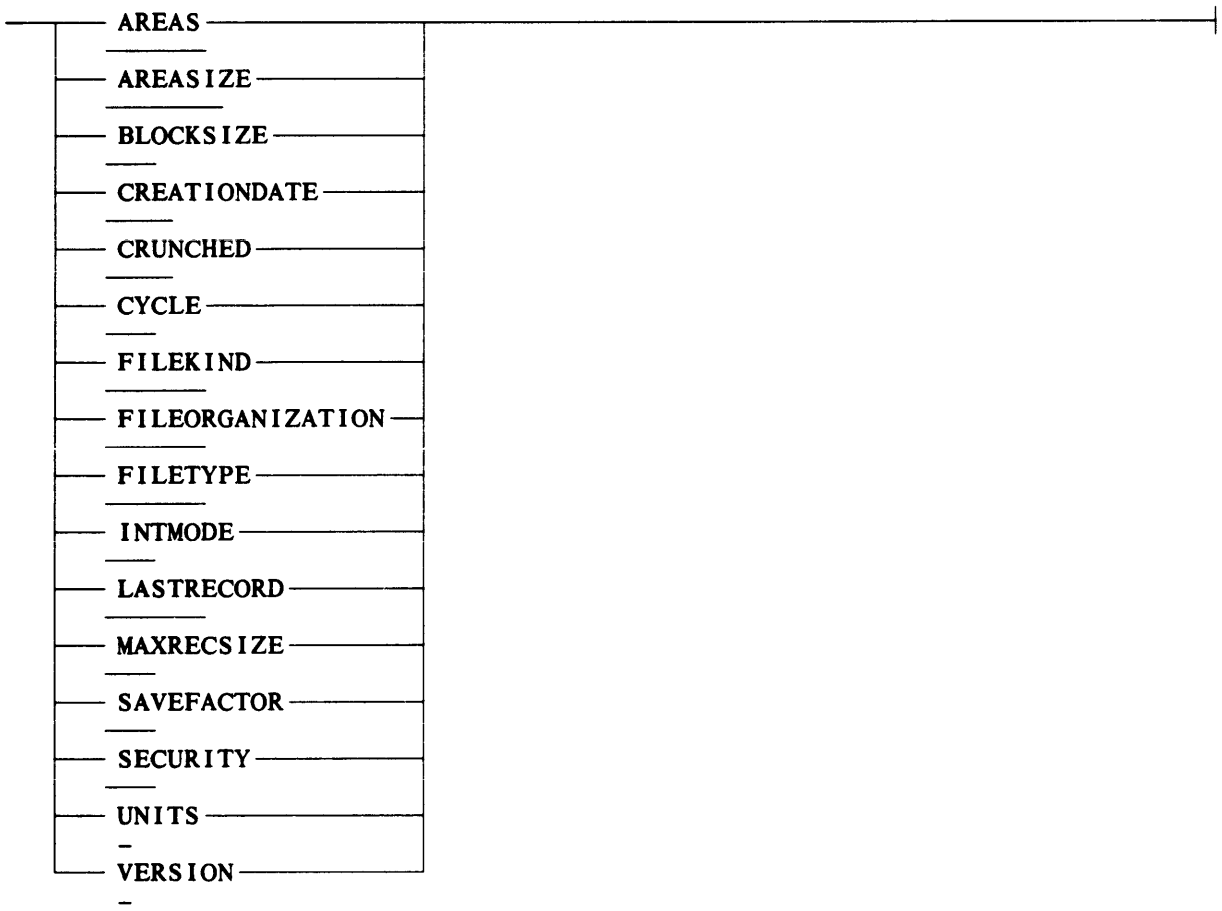
ATTRIBUTES

Syntax



FILEDATA

<file attribute>



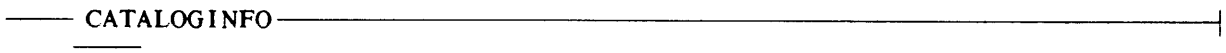
<file attribute> may be any one of the above valid file attribute names. A complete description of all file attributes may be found in the B 7000/B 6000 Series I/O Subsystem Reference Manual, form 5,001779.

Semantics

ATTRIBUTES produces a report showing various requested attributes of a file or group of files.

CATALOGINFO

Syntax

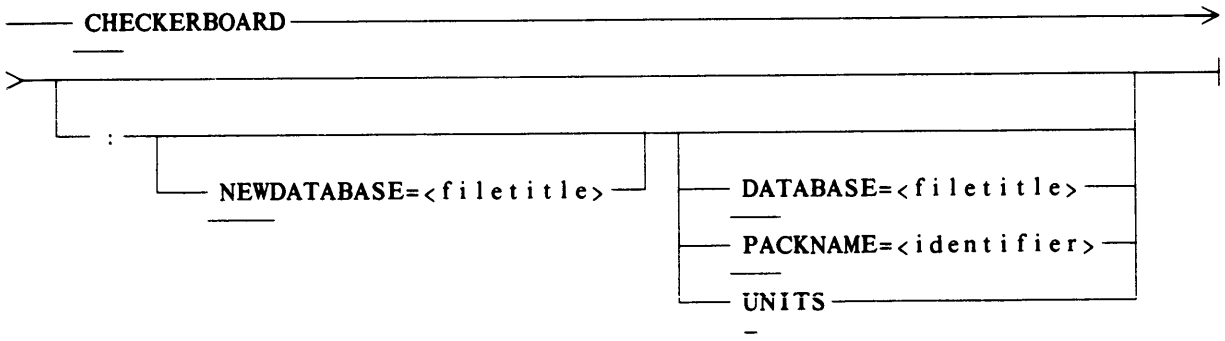


Semantics

CATALOGINFO lists the catalogue information about a file. Included in the output is the creation date, last access date, file kind, status, class, controlled segments, and hierachical names.

CHECKERBOARD

Syntax

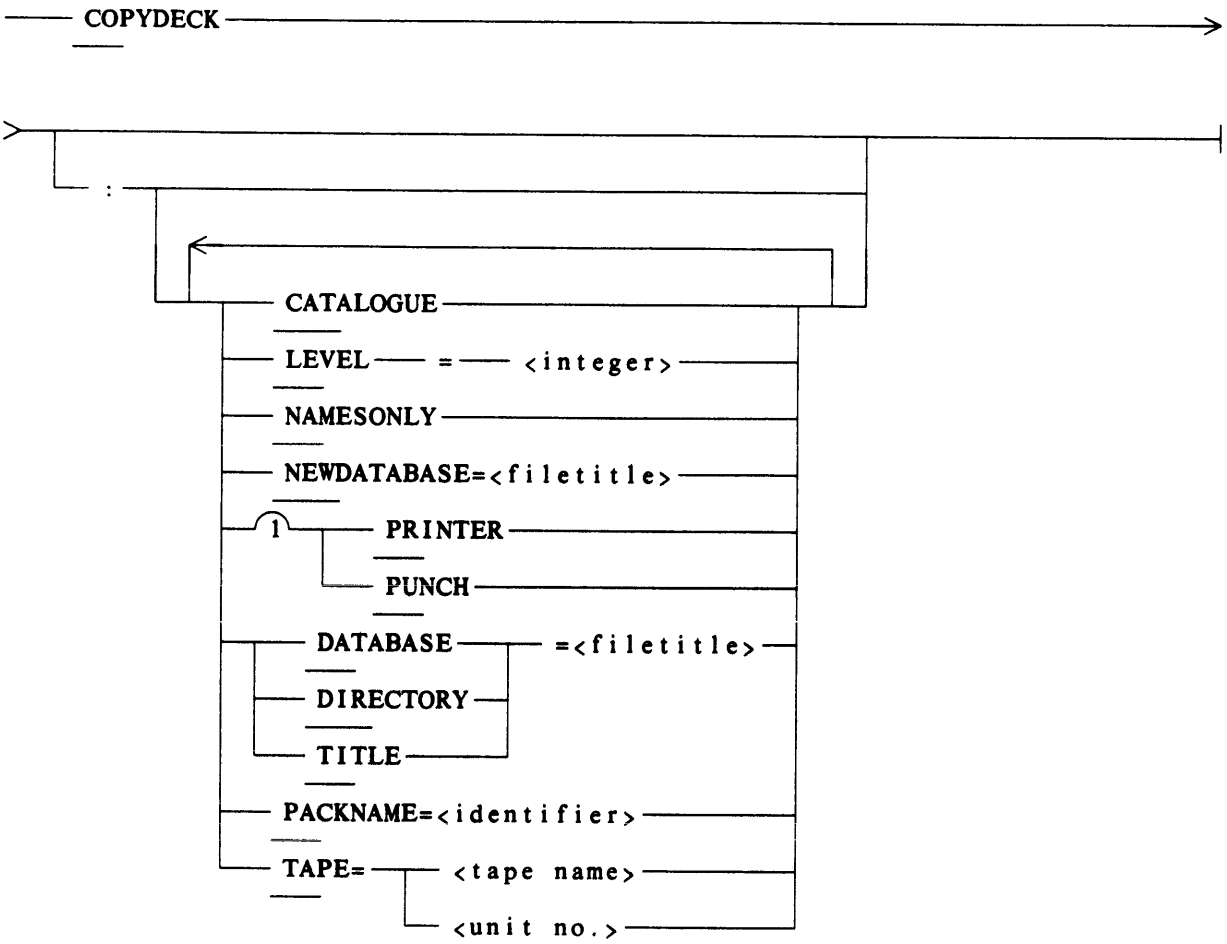


Semantics

CHECKERBOARD produces a disk checkerboard displaying permanent files and the space around them. It includes family index, base address, end addess, length, area, file name, and space between rows.

COPYDECK

Syntax

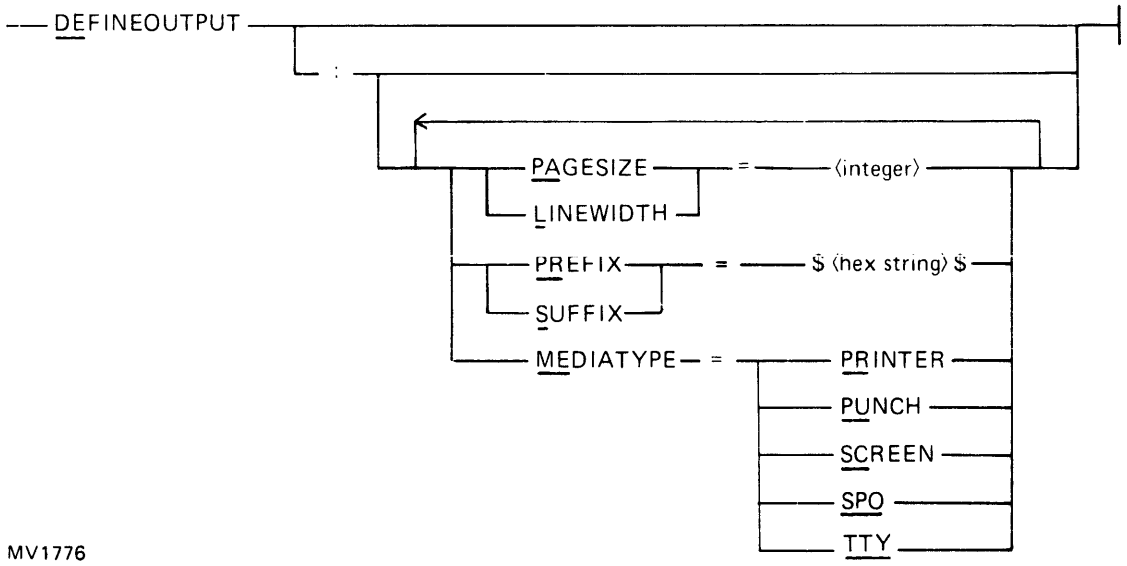


Semantics

COPYDECK produces a file suitable for use in a library maintenance copy deck. The file is sent to the card punch by default.

DEFINEOUTPUT

Syntax

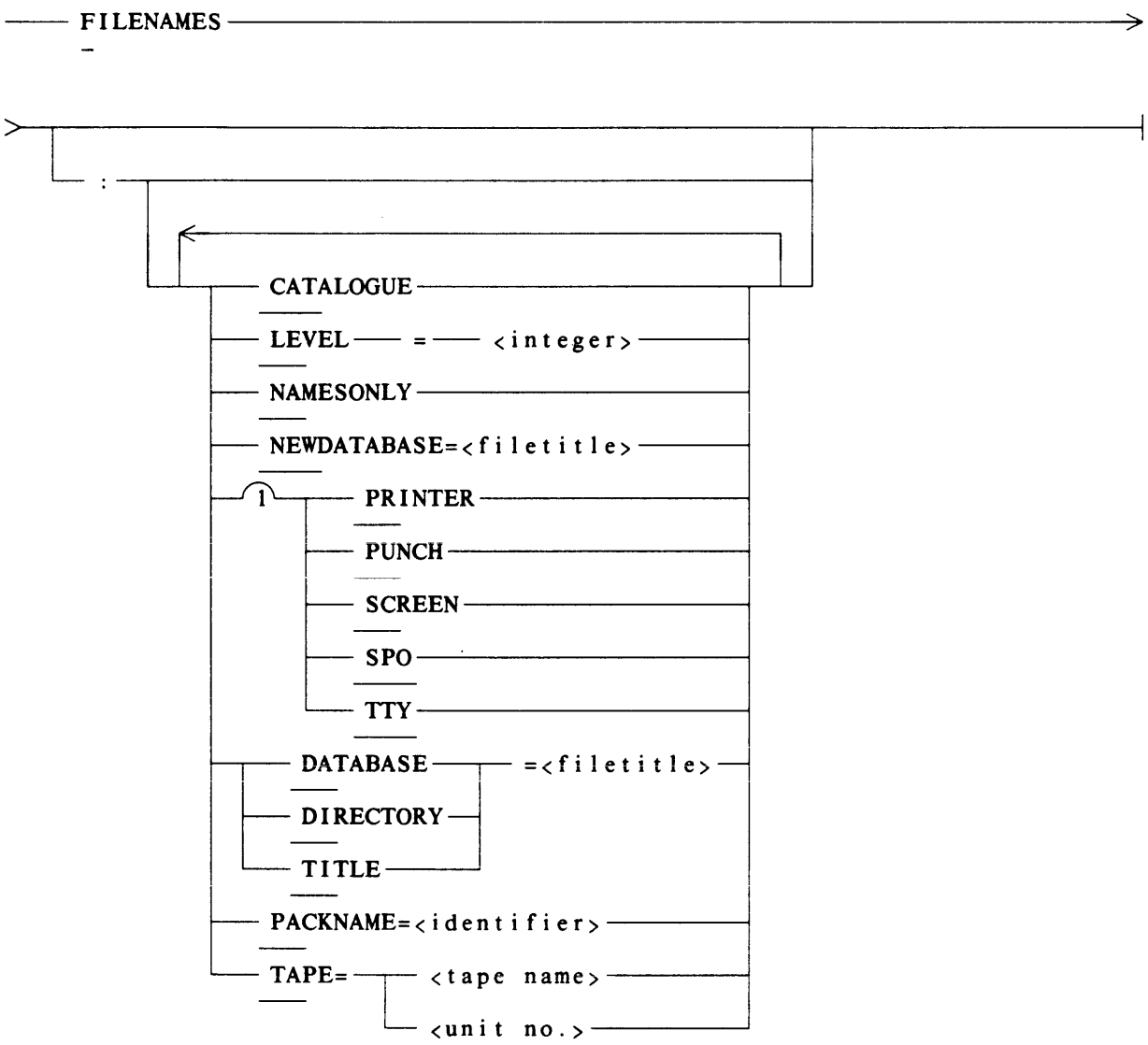


Semantics

DEFINEOUTPUT is used to reformat a report's output. This <task reguest> allows the programmer to explicitly control line width, page size, and output media. Additionally, a prefix and/or suffix may be specified for each output line.

FILENAMES

Syntax

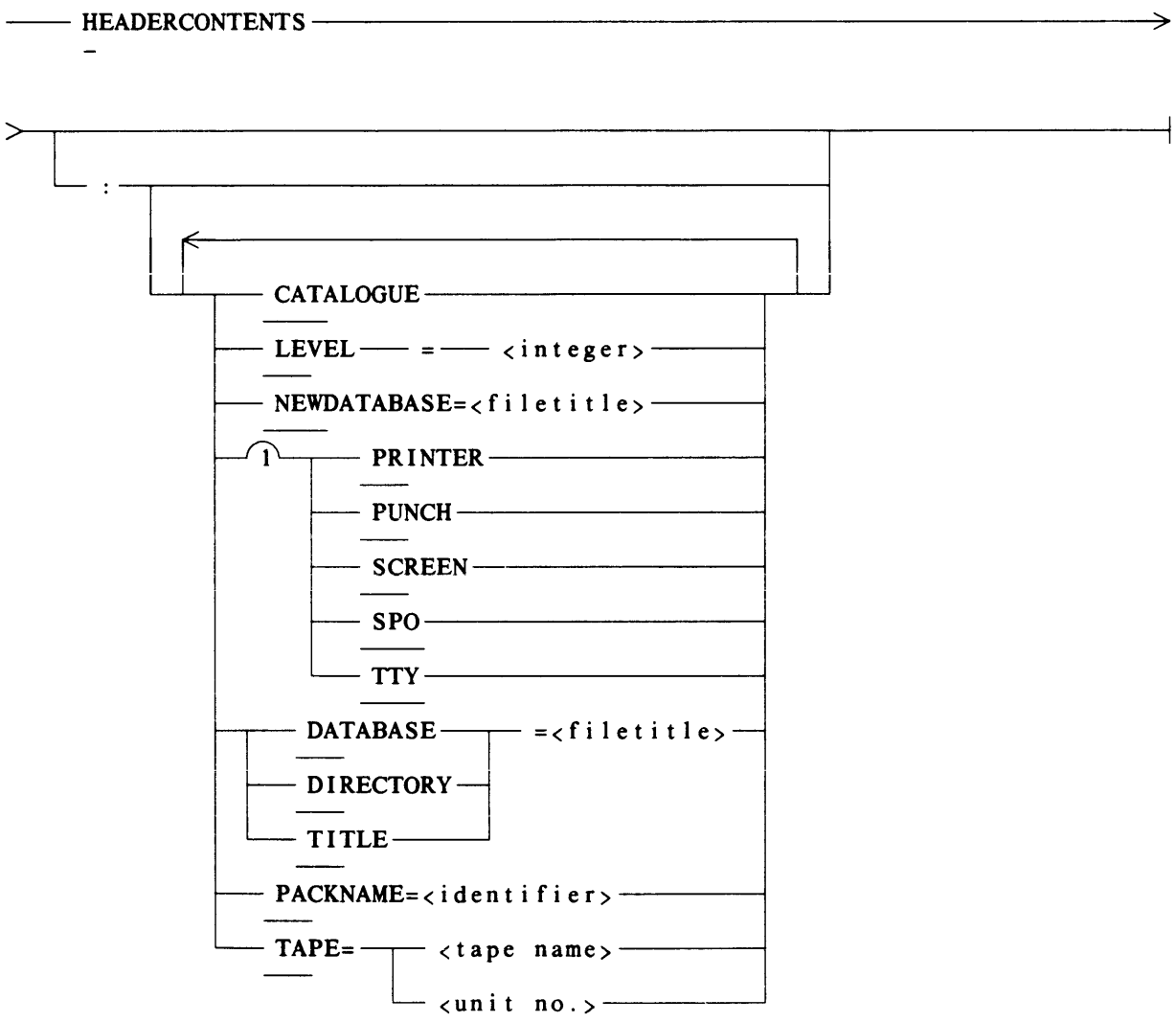


Semantics

FILENAMES produces a hierarchical list of files, including access and creation dates, size in segments, security class, status, and file kind.

HEADERCONTENTS

Syntax

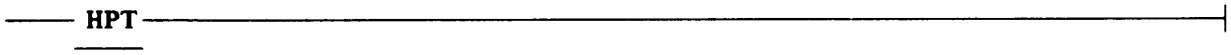


Semantics

HEADERCONTENTS produces a HEX dump of file headers, row address words, and CLASSB security information.

HPTRESOURCES

Syntax

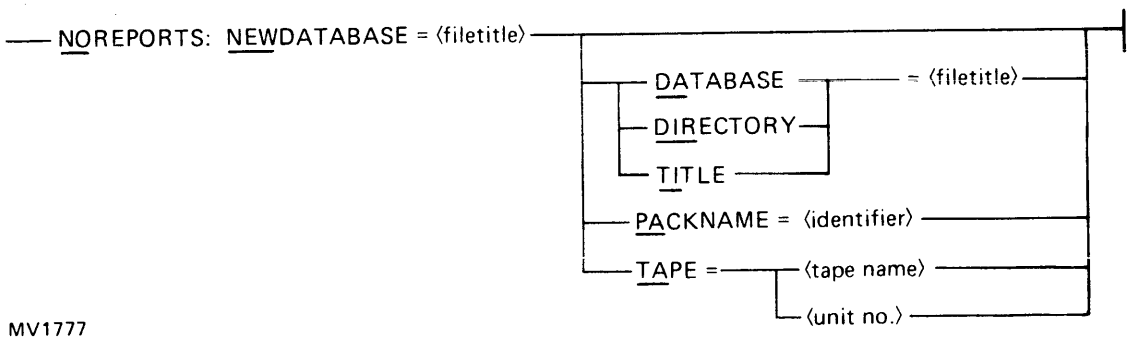


Semantics

HPTRESOURCES produces a report on the status of all HPT disk attached to the system.

NOREPORTS

Syntax

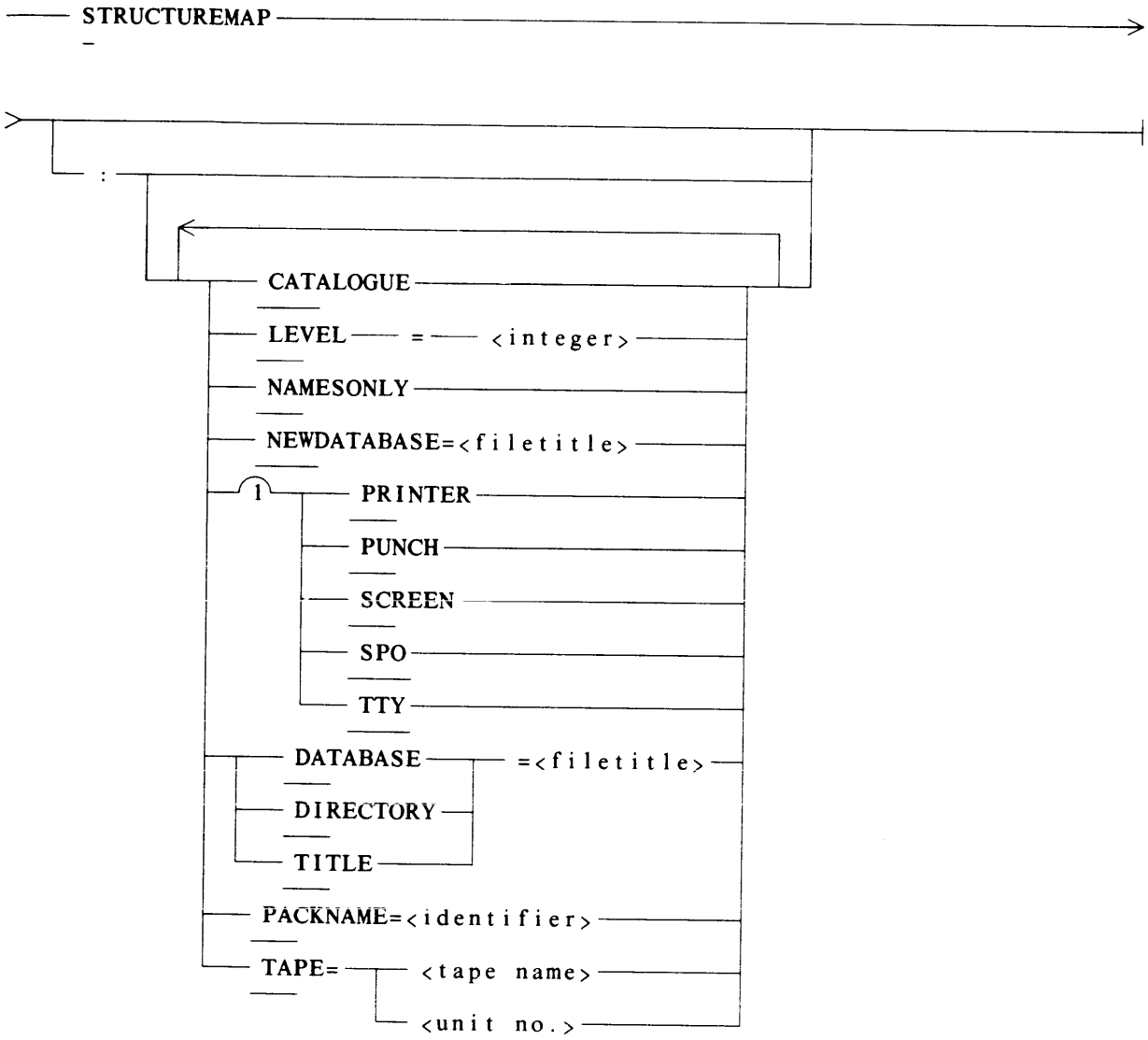


Semantics

NOREPORTS generates a **NEWDATABASE** without generating any reports. This **NEWDATABASE** can then be used in future runs of **FILEDATA**.

STRUCTUREMAP

Syntax

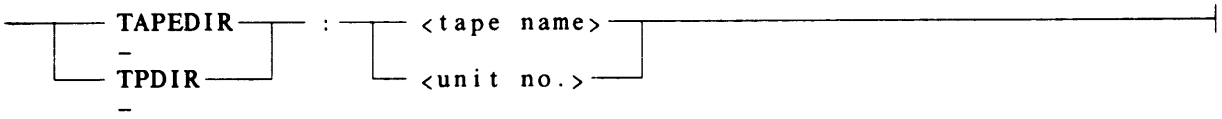


Semantics

STRUCTUREMAP produces a map showing file storage layout by family index and address. The report contains the filename, areas, area class, family index, segment address, size in segments, and number of segments on the family.

TAPEDIR

Syntax



Semantics

TAPEDIR reads library maintenance tape directories and prints the tape name, unit number, serial number, creation date, tape type, and a list of files. TAPEDIR may only be requested by a privileged user as security usercode information is accessible.

FILEDATA

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

4. MODIFIERS

<modifier>s specify options for task requests. Each task request permits a different set of <modifiers>. <modifier>s apply to all reports until overridden by another <modifier>.

The <modifier>s are:

ABBREVIATED

ABBREVIATED causes the titles of the requested attributes to be abbreviated on the output listing. For example, AREASIZE is output as ASIZE.

ALL

ALL specifies that all of the attributes of the specified files are to be listed.

AREAS

AREAS is the number of words in the file header allocated for row addresses.

AREASIZE

AREASIZE is the number of logical records in an area of a disk file. The most current file is the one with the highest CYCLE and the highest VERSION of that CYCLE.

CATALOGUE

Reports the existence of non-resident catalogued files as well as (default) resident files.

CREATIONDATE

CREATIONDATE indicates the date the file was created. The date is given in the form mm/dd/yy.

CRUNCHED

CRUNCHED is listed as an attribute of a permanent disk file which has returned the unused portion of its last area of the file to the system.

CYCLE

CYCLE and VERSION are used to identify generations of a permanent file. The most current file is indicated by the highest CYCLE and the highest VERSION of that CYCLE.

DATABASE

This requests information for the <task> to come from an existing file of raw information which NEWDATABASE created at some time in the past.

DOUBLE

DOUBLE causes the output generated by FILEDATA to be double-spaced.

FILEKIND

FILEKIND describes the internal structure and/or purpose of a record of a disk file or the kind of label on a tape. (Refer to the B 7000/B 6000 Series I/O Subsystem Reference Manual, form 5001779, for a listing of the various FILEKINDS.)

FILEORGANIZATION

FILEORGANIZATION is the organization under which the file was opened.

FILEORGANIZATION is shown only if FILEKIND > OR = VALUE(DATA).

FILETYPE

FILETYPE specifies the format of the records and the structure of the file. (Refer to the B 7000/B 6000 Series I/O Subsystem Reference Manual, form 5001779, for a description of the various FILETYPEs.)

INTMODE

INTMODE lists the internal character size of the file. (For a description of the values and mnemonics, refer to the B 7000/B 6000 Series I/O Subsystem Reference Manual, form 5001779.)

LASTACCESSDATE

LASTACCESSDATE is the date the file was last accessed. The date is printed in the form mm/dd/yy.

LEVEL

LEVEL specifies the number of leading file names to be printed. For example, LEVEL=2 reports on A/B but only shows that the directory X/Y exists although the file X/Y/Z is present.

LINEWIDTH

LINEWIDTH defines the length of an output line.

MAXRECSIZE

MAXRECSIZE is the maximum size of records in the logical file.

MEDIATYPE

MEDIATYPE specifies the output device.

MINRECSIZE

MINRECSIZE is the minimum size of records in the logical file.

NAMESONLY

NAMESONLY indicates that header information is to be neither extracted nor processed in any report.

NEWDATABASE

NEWDATABASE saves the current DATABASE under a specified <filename>.

PACKNAME

PACKNAME changes the source of information from the HPT disk system to the named disk pack. The entire pack is used in the report. This <modifier> overrides DATABASE, DIRECTORY, and TAPE.

PAGESIZE

PAGESIZE specifies the number of output lines per page.

PREFIX

PREFIX allows the user to specify a hexadecimal string that subsequently precedes each line of output.

PRINTER

Output is to go to the line printer, single-spaced, 58 lines per page (six lines per inch), 132 characters per line.

PUNCH

Output is to go to a standard 80-column card punch.

SAVEFACTOR

SAVEFACTOR indicates the expiration date of a file in terms of the number of days past the creation date.

SCREEN

Output is sent to a remote terminal assumed to be a CRT screen device with 24 lines per page and 80 characters per line. A read occurs at the end of each page to allow user action.

SECURITY

SECURITY lists the security type of a file. (Refer to the B 7000/B 6000 Series I/O Subsystem Reference Manual, form 5001779, for a discussion of security.)

SPO

Output goes to the system console, assumed to be 80 columns by 24 lines per page.

SUFFIX

SUFFIX allows the user to specify a hexadecimal string that subsequently is appended to each line of output.

TAPE

TAPE allows information to be extracted from library maintenance tapes. It also assumes the <modifier> NAMESONLY and overrides DATABASE, DIRECTORY, and PACKNAME. Multiple reel tapes function best when the unit number is specified.

TIMESTAMP

The date and time the last alteration was made to the file.

TITLE or DIRECTORY

These <modifier>s allow the user to report on less than the full disk system.

TTY

Output is sent to a remote terminal assumed to be a hard copy device with 80 characters per line.

UNITS

UNITS indicates the UNITS attribute of the file. (Refer to the B 7000/B 6000 Series I/O Subsystem Reference Manual, form 5001779, for a discussion of the UNITS attribute.)

VERSION

VERSION and CYCLE are used to identify generations of a permanent file. The most current file is indicated by the highest CYCLE and the highest VERSION of that CYCLE.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

5. NUMERIC REPORT REQUESTS

In order to cut down the amount of input which must be supplied, especially for standard functions such as LISTDIRECTORY, a <numeric report request> has been included. <numeric report request>s allow reports to be requested by number. Such a number may be entered via a VALUE=<numeric report request> or in the regular <parameter list>. A <task> requested via a VALUE=<numeric report request> is performed before the <parameter list>, if any, is processed. Numeric requests in the <parameter list> are treated like any other <task> request.

Example

```
RUN SYSTEM/FILEDATA ("0;ATTRIBUTES:DIR=MYSELF ALL; 1 ")
```

Semicolons are used to separate <numeric report request>s and <task request>s. <numeric report request>s may not contain <modifier>s. <numeric report request>s are implemented as executable statements within SYMBOL/FILEDATA. New reports are defined by modifying SYMBOL/FILEDATA and recompiling a new SYSTEM/FILEDATA. Reports for 0 and 1 are presently defined (1 is equivalent to the FILENAMES <task> and 0 includes the <task>s FILENAMES, STRUCTUREMAP, and CHECKERBOARD).

Future systems releases utilize odd numbers for reports. Even numbers should be chosen for installation defined reports.

Each request, numeric or standard, goes through an input scanner and the results are reported prior to any line printer listings. These results include the assumed <task> identifier, a listing of input for each <task>, and any error messages. If any errors do occur, this <task> and any subsequent <task>s are checked for input syntax only; no reporting is done.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

INTERACTIVE XREF

TABLE OF CONTENTS

1. GENERAL INFORMATION

INTRODUCTION

A cross reference of a program contains an entry for each identifier declared in the program. This entry is referred to as the header line information. Each entry consists of the following:

1. The alphabetic name.
2. The declared type of the identifier.
3. The environment.
4. The stack location.
5. The sequence number of the declaration.

INTERACTIVEXREF allows interactive access to this information and to detailed information about the identifiers.

FILES

INTERACTIVEXREF obtains information from files generated by SYSTEM/XREFANALYZER. The INTERACTIVEXREF information files are given the titles XREFFILES/<code file name>/DECS and XREFFILES/<code file name>/REFS. <code file name> is the name of the code file that was being generated by the compiler when the INTERACTIVEXREF information files were created. The <code file name> is normally prefixed by OBJECT/ if compiled through CANDE.

The compiler dollar option called XREFFILES causes INTERACTIVEXREF information to be produced. This option exists in the ALGOL, NEWP, ESPOL, and FORTRAN compilers. When XREFFILES is set, and XREF is not set, the XREFANALYZER is run by the compiler to produce INTERACTIVEXREF information files. When both XREFFILES and XREF are set, the XREFANALYZER is run to produce both the information files and a printed output. When XREF only is set, the XREFANALYZER is run to produce printed output. In no case is the normal XREFANALYZER run if any syntax errors are encountered by the compiler.

The XREFFILES may also be produced by running XREFANALYZER with a negative task value. The XREFANALYZER input file (TITLE=XREF/<code file name>) must be specified when initiating XREFANALYZER.

Information from the original symbol file used in the compile is needed by several commands.

NOTE

Because the wrong symbol file can be loaded (see SYMBOL command), caution should be exercised.

VERSION information is included in the XREFFILES. INTERACTIVEXREF checks the VERSION compatibility of the XREFFILES and displays an appropriate error message if the XREFFILES were created with an incompatibile XREFANALYZER.

OPERATION

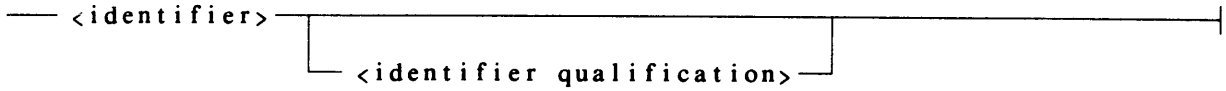
The INTERACTIVEXREF must be run from a terminal. Commands may be entered one at a time, or multiple commands may be placed on the same line, separated by semicolons. A command line may be continued by terminating it with a percent sign (%) and continuing on the next line.

When a command is printing information on the terminal, it may be discontinued by hitting BREAK. When the command is not printing, it may be discontinued by entering the CANDE command ?HI. In either case, the remaining commands on the current input line (if any) are ignored.

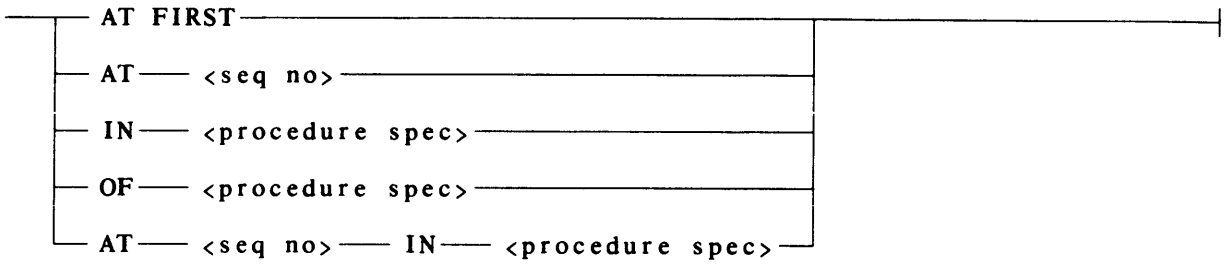
IDENTIFIER SPECIFICATION

Syntax

<identifier spec>



<identifier qualification>



<procedure spec>

Semantics

In many commands, the program must be told exactly which identifier the user has in mind. This declaration, however, becomes complicated because the same identifier may be re-declared in many different procedures or blocks.

<identifier> may be an alphanumeric identifier, which starts with a letter and is composed entirely of letters and digits. In an ALGOL program, an identifier of the form B.0002 is acceptable.

<identifier qualification> is useful when the same identifier is re-declared in many different procedures or blocks. It allows the user to specify which occurrence of the identifier is intended. The possible identifier qualifications are:

AT FIRST

Selects the first occurrence of the identifier which was encountered by the compiler.

AT <seq no>

Selects the occurence of the identifier which is declared or used closest to the specified sequence number and which was not declared beyond the specified sequence number. If all occurrences of the identifier were declared beyond the specified sequence number, the closest occurrence is chosen. If an exact match is not found, a warning will be printed. This identifier qualification gives

undefined results if the source file seen by the compiler has not been sequenced properly (see Appendix 6B).

IN <procedure spec>

Finds a use of the <identifier> in the specified procedure. It looks first for an <identifier> declared by the specified procedure. Failing this, a global identifier referenced by the procedure is sought. Failing this, an <identifier> declared in a procedure nested within the specified procedure is sought. If these searches fail, an error occurs.

OF <procedure spec>

Looks for an occurrence of the identifier which is declared by the specified procedure. Note that an occurrence of the identifier declared by a procedure contained within the specified procedure is not acceptable.

AT <seq no> IN <procedure spec>

From those occurrences of the identifier which are declared or used within the specified procedure, the one which is declared or used closest to the specified sequence number is selected. This option is useful as an alternative to AT <seq no> when the specified procedure, but not the entire source, is properly sequenced (see Appendix 6B).

The <procedure spec> need only be long enough so that its outermost environment is the "best candidate" of the possible environments specified by that identifier. The best candidate is defined as follows:

1. If only one environment exists (for example, module or procedure) with this name, use it.

2. If more than one environment exist with this name, limit the possible candidates to the most global.

3. If equally global environments exist for this name, use the first.

If more than one environment exists for a specified name, a warning of possible ambiguity and the chosen environment are output.

Example

```
JOE
     HARRY
HARRY
FRANK
     TOM
          STEVE
     BOB
          STEVE
               HARRY
```

The environments are the following:

```
JOE
HARRY OF JOE
HARRY
FRANK
TOM OF FRANK
STEVE OF TOM OF FRANK
BOB OF FRANK
STEVE OF BOB OF FRANK
HARRY OF STEVE OF BOB OF FRANK
```

"TOM" locates "TOM OF FRANK." TOM is a unique environment.

"STEVE" locates "STEVE OF TOM OF FRANK." Since both STEVES are at the same level, the first is used and a warning is emitted.

"STEVE OF BOB" locates "STEVE OF BOB OF FRANK." BOB is a unique environment.

"HARRY" locates "HARRY", because it is the most global, and outputs a warning.

"HARRY OF STEVE" is an error. The environment used for STEVE is "STEVE OF TOM OF FRANK" (the first of the STEVES on the same level) and no HARRY exists in that environment.

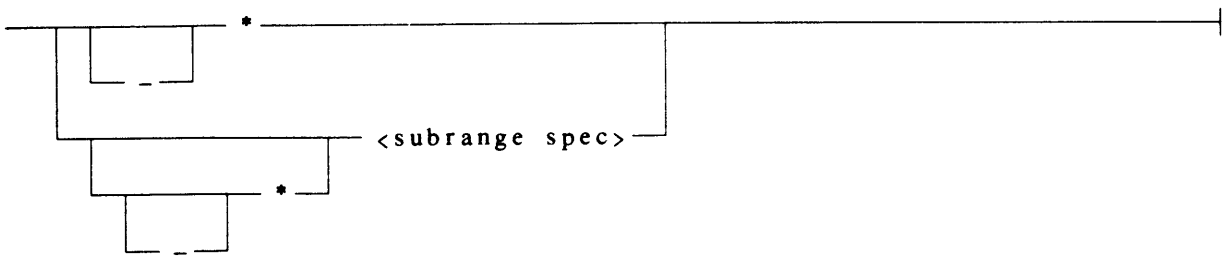
"HARRY OF STEVE OF BOB" locates "HARRY OF STEVE OF BOB OF FRANK." BOB is a unique environment.

"HARRY OF JOE" locates "HARRY OF JOE." JOE is unique.

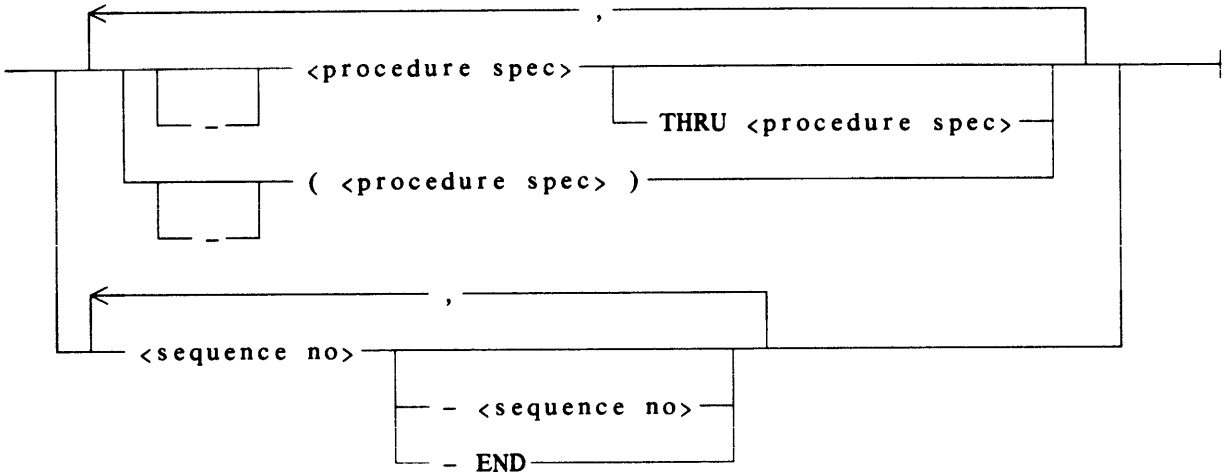
RANGES

Syntax

<range spec>



<subrange spec>



Semantics

Sometimes restricting a request to a certain subset of the source file is useful. Ranges have been implemented for this purpose. Ranges may be specified in terms of either sequence numbers or environments (procedure names). Sequence numbers and environments may not be intermixed in the same range specification.

A sequence number range consists of any number of sequence numbers and/or sequence number pairs. This type of range gives undefined results if the source file seen by the compiler has not been sequenced properly (see Appendix 6B).

An environment range consists of a list of procedure specifications and variations of procedure specifications. The variations are as follows:

<procedure spec>	Include the specified procedure and all procedures nested within it.
<procedure spec> THRU <procedure spec>	Include all procedures (in order of declaration) from the first specified procedure through the second.
(<procedure spec>)	Include only the specified procedure; not the procedures nested within it.

If a minus sign precedes any of the above, it means those procedures should not be included. If the first item of the range begins with a minus sign, the range starts out including all procedures, rather than not including any.

If a range begins with an asterisk (*), it has the effect of inserting the current default reference range at the start of the range. (See RANGE command description.)

If a range begins with a minus asterisk (-*), it has the effect of inserting the complement of the current default reference range at the start of the range.

Examples

500-900, 2300-END

Include all sequence numbers from 500 through 900 and 2300 through 99999999.

JOE, (SAM), B.0004 OF TOM

Include global procedure JOE and all procedures declared within it, global procedure SAM but not the procedures declared within it, and the block within global procedure TOM named B.0004 by the compiler.

-JOE

Include everything but global procedure JOE and all procedures declared within it.

_*

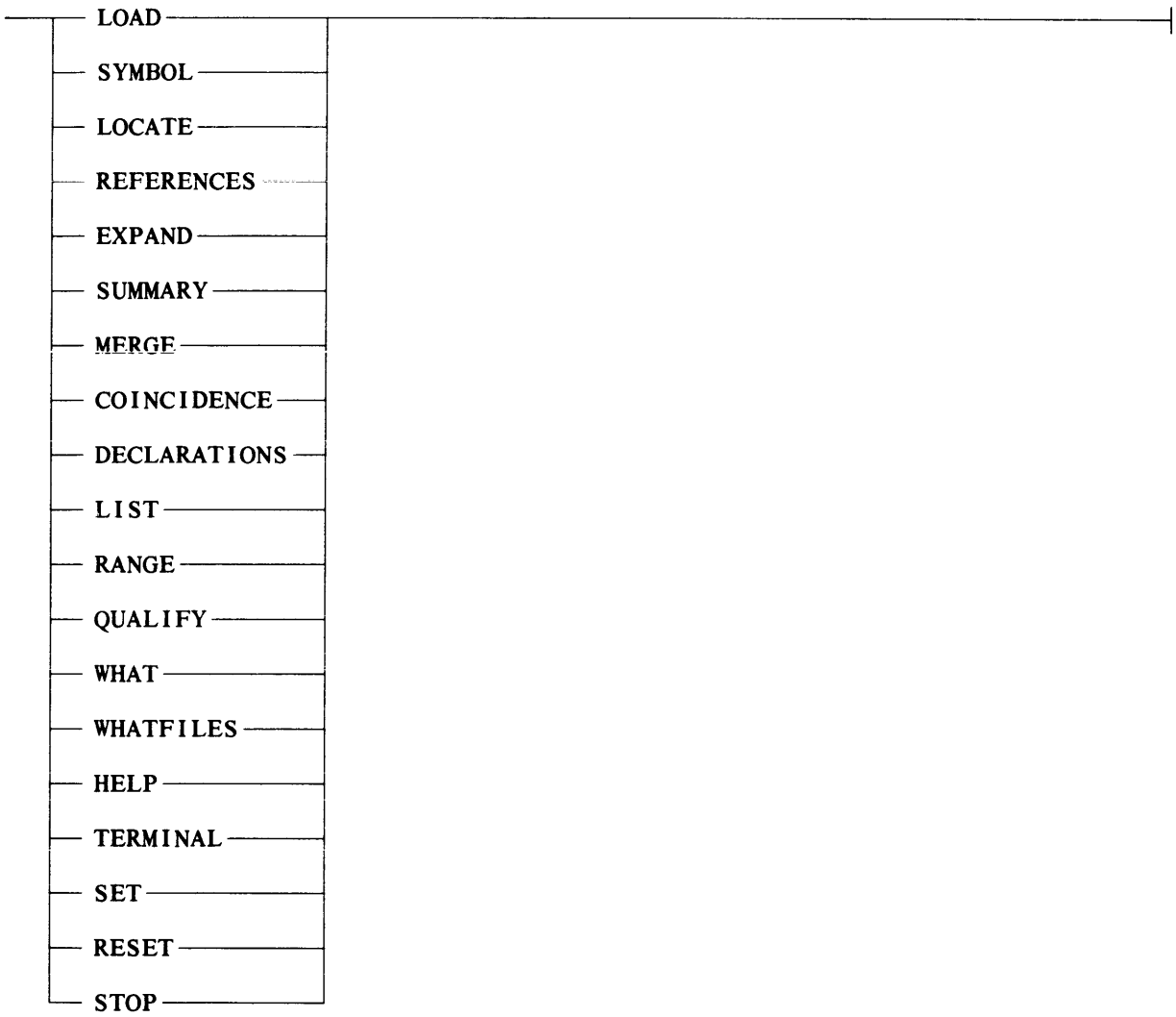
Specifies the complement of the current default reference range.

*, JOE

Specifies the current default reference range as well as global procedure JOE and all procedures declared within it.

2. COMMANDS

Syntax

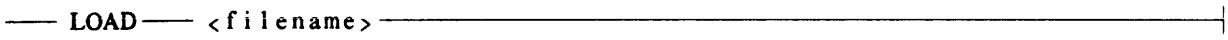


Semantics

The command descriptions appear in an order appropriate for the first time reader. An attempt has been made to write each command description in a way that does not depend on the other descriptions. The order of appearance does not necessarily reflect the usefulness of a particular command (that is, the DECLARATION command will probably be used more than the LOCATE command).

LOAD

Syntax



Semantics

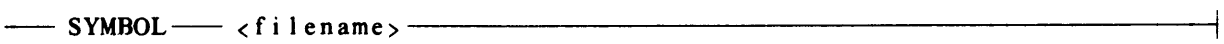
This command loads the INTERACTIVEXREF information files. The file name specified is that of the object file being generated by the compiler when the XREF information files were generated. The titles of the XREF information files are constructed from this file name and are loaded. To prevent confusion, this command nullifies any previous SYMBOL command.

NOTE

When compiling a CANDE work file, the <filename> is of the type "CANDE/CODE"<number>.

SYMBOL

Syntax



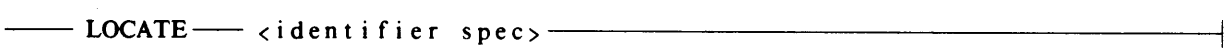
Semantics

This command loads the symbol file from which text for define expansion, text corresponding to a given reference, and text for the LIST command is taken. If none of these items is desired, a symbol file need not be loaded.

The symbol file loaded should be the source that was being compiled when the INTERACTIVEXREF information files were generated. It is desirable, but not necessary, that the symbol file and the XREF information files correspond exactly. If discrepancies are found when processing a command that requires information from both sources, a warning or error is issued.

LOCATE

Syntax



Semantics

The specified identifier is found and described in terms of environment (where declared), compiler class (REAL, INTEGER, and so forth), sequence number (where declared), aliases, and so on. Other commands such as REFERENCES also print out this header line information when a specified identifier is found.

Examples

1. LOCATE I AT 47362000

   Locates the I declared or used closest to sequence number 47362000.

2. LOCATE J IN SAM

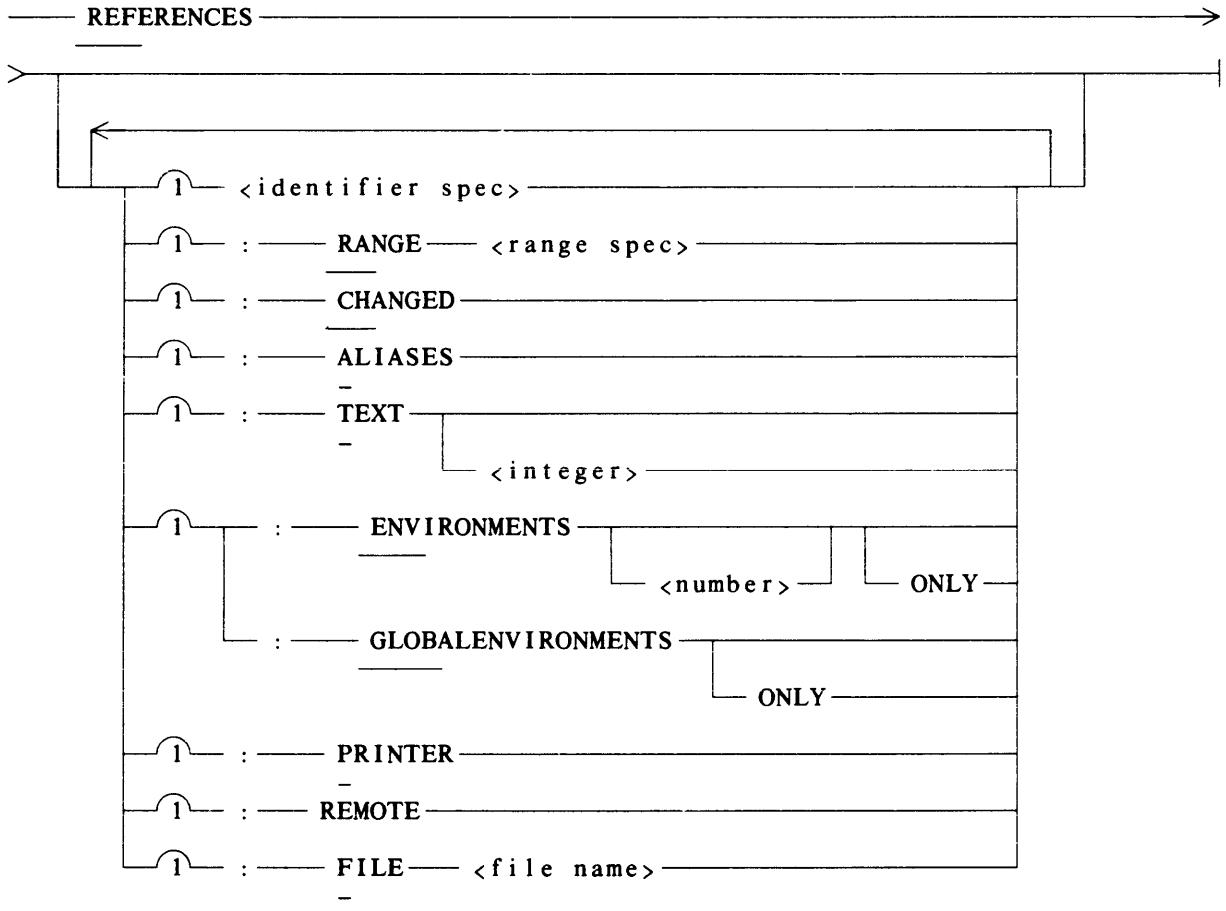
   Looks for an identifier J declared by global procedure SAM. Failing this, looks for a global identifier J referenced by SAM, and failing this, looks for an identifier J declared by a procedure nested within SAM.

3. LOCATE K OF JOE OF HARRY

   If an identifier K declared by procedure JOE of global procedure HARRY is not found, an error results.

REFERENCES

Syntax



Semantics

References to the specified identifier are listed. If no identifier is specified, and the previous command was LOCATE, REFERENCES, EXPAND, or SUMMARY, then the identifier specified in that command is used. This identifier, which is remembered from command to command, is known as the "work identifier".

Unless modified by some of the options described below, the references are printed in the form familiar from printed XREFs. The 8-digit sequence number of each line where the identifier is referenced is printed. It is preceded by an asterisk (*) if the value might be changed by the statement and followed by a pound sign (#) if the reference occurred as part of an expanded define. The available options are as follows:

RANGE — Allows the user to restrict the range over which references are to be printed. If this option is not specified, then the default reference range, as specified by the RANGE command, is used. The default value of the default reference range is the entire program.

CHANGED	Only those references where the value of the identifier might be changed by the statement is listed.
ALIASES	Causes a merged list of references to the identifier and all of its aliases (if any) to be listed. Those sequence numbers where an alias is referenced are marked with a plus sign. Currently, only ESPOL keeps track of aliases.
TEXT	Causes the text from the symbol file to be printed with each reference. If an integer is specified with this option, a sample of that many lines of text, centered at the line containing the reference, is printed with each reference. Note that the symbol file must be loaded to use this option.
ENVIRONMENTS	Causes the names of the environments (procedures and blocks) where the references occur to be printed, appropriately interleaved with the references. If modified by NUMBER, then only that many levels of environments are listed. If modified by ONLY, then only the environments, and not the references, are printed. Note that ENVIRONMENTS ONLY and TEXT are mutually exclusive.
GLOBALENVIRONMENTS	Similar to ENVIRONMENTS, except that references are broken down only by global environment (global procedure). Note that GLOBALENVIRONMENTS ONLY and TEXT are mutually exclusive.
PRINTER	Causes output to go to the line printer, by way of a file internally named LINE.
REMOTE	For use when the output is to go to both the line printer and the terminal.
FILE	Causes all referenced text lines from the symbol file to be output to the user specified disk file; this file cannot already exist. It is made with the same filetype as the file loaded by the symbol command. If no symbol file is loaded, an error message is output to the terminal.

Examples

REFERENCES ABD

A list of references to ABD is printed.

REF K1 IN JOE :CHANGED :TEXT

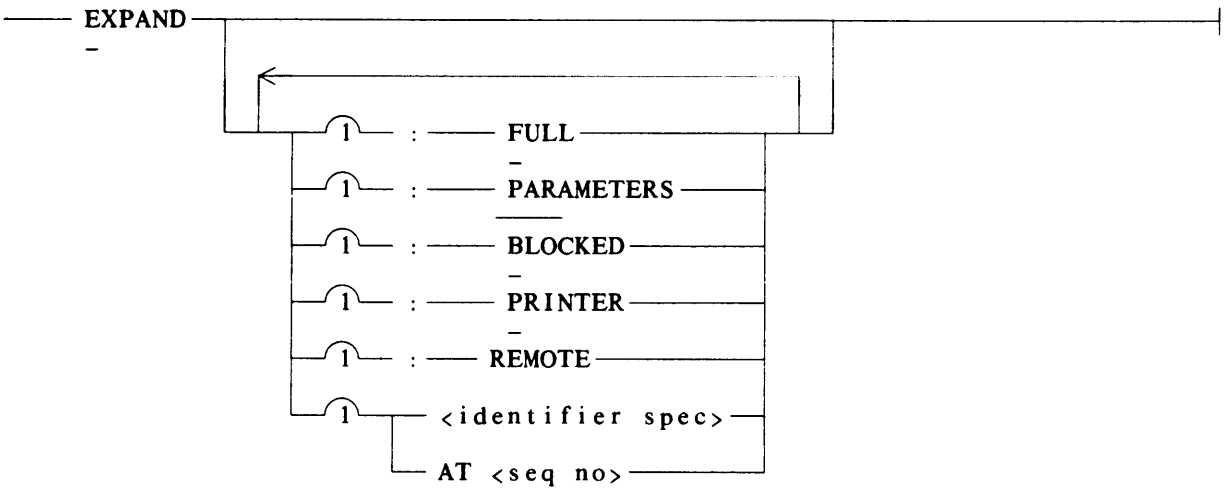
Locates the identifier K1 used in procedure JOE, prints a list of those references where the value of K1 might be changed and the line of text corresponding to each reference.

REF :TEXT 3 :ENVIRONMENTS :PRINTER

Prints a list of references to the work identifier. Three lines of text are printed with each reference. The references are grouped by the procedures within which they occur, and preceding each group is the name of the procedure. References that occur in the main program appear at the start of the list. The output goes to the line printer.

EXPAND

Syntax



Semantics

The text of the specified declaration is written out (expanded) if the identifier is an item such as a define, array, or file, which has text associated with its declaration. If no identifier is specified, and the preceding command was LOCATE, REFERENCES, EXPAND, or SUMMARY, then the identifier specified in that command is used. This identifier, remembered from command to command, is known as the work identifier. If no identifier is specified and the work identifier is empty, an error occurs.

Unless the command is modified by certain options described below, the action taken is as follows. The first time an identifier that is or has just been specified is expanded, the text of the declaration is given. Subsequent requests to expand the work identifier cause this text to be scanned for occurrences of other defines. If such nested defines are found, they are replaced by their text. This process may be repeated, one step at a time, until the text is completely expanded (no more nested defines are found), a new work identifier is specified, or the work identifier is nullified. Once the expansion is complete, a message to that effect is printed. Subsequent requests to expand the work identifier cause the final version to be printed.

The expansion of defines is context sensitive. When a define is used within a procedure or block more local than that in which it was declared, this inner procedure or block may have re-declared some of the identifiers used in the text of the define. Unless the identifier was specified using a qualification such as AT<seq no>, which gives some indication as to what context should be used, a context must be chosen. If the define is referenced, the context of the first reference is used. If the define is never referenced, it is expanded in the context of its declaration. In either case, a warning message is printed.

The following options are available:

AT<seq no> Restarts the expansion of the work identifier in the context of the reference nearest <seq no>.

FULL Causes the text to be completely expanded before it is printed.

PARAMETERS If the identifier being expanded is a parameterized define, and if it is being expanded in the context of a reference, then actual parameters are extracted from the text of that reference and substituted for the formal parameters in the expansion. Even if the expansion was complete before the actual parameters were inserted, it reverts back to the incomplete state because the actual parameters may contain identifiers that are defines.

NOTE

If both FULL and PARAMETERS are specified, the actual parameters are inserted before the full expansion is done.

PARAMETERS works only if the define was referenced directly and not by another define.

BLOCKED Normally, the first expansion is printed out exactly as it appears in the symbol file (including comments). Subsequent levels of expansion are blocked according to a simple scheme that indents at BEGINs and puts statements on separate lines. This option requests that the first expansion also be blocked.

PRINTER Causes output to go to the line printer by way of a file internally named LINE.

REMOTE Causes output to come to the terminal, even if it is also going to the line printer.

Examples

EXPAND UNLOCK

The text of the identifier UNLOCK is printed.

EXPAND :FULL :PRINTER

The text of the current work identifier is completely expanded and written to the line printer.

Limitations

1. The symbol file is needed for this command.

2. The room available for storing expansion text is limited. The first level of expansion is always completely printed out. Higher levels may be truncated if internal storage is insufficient.

3. For any given expansion, expansion of nested defines is carried out in one and only one context. Also, text is not syntaxed completely, and declarations cannot be distinguished from other statements. Thus, higher levels of expansion of defines that contain declarations may be incorrect.

SUMMARY

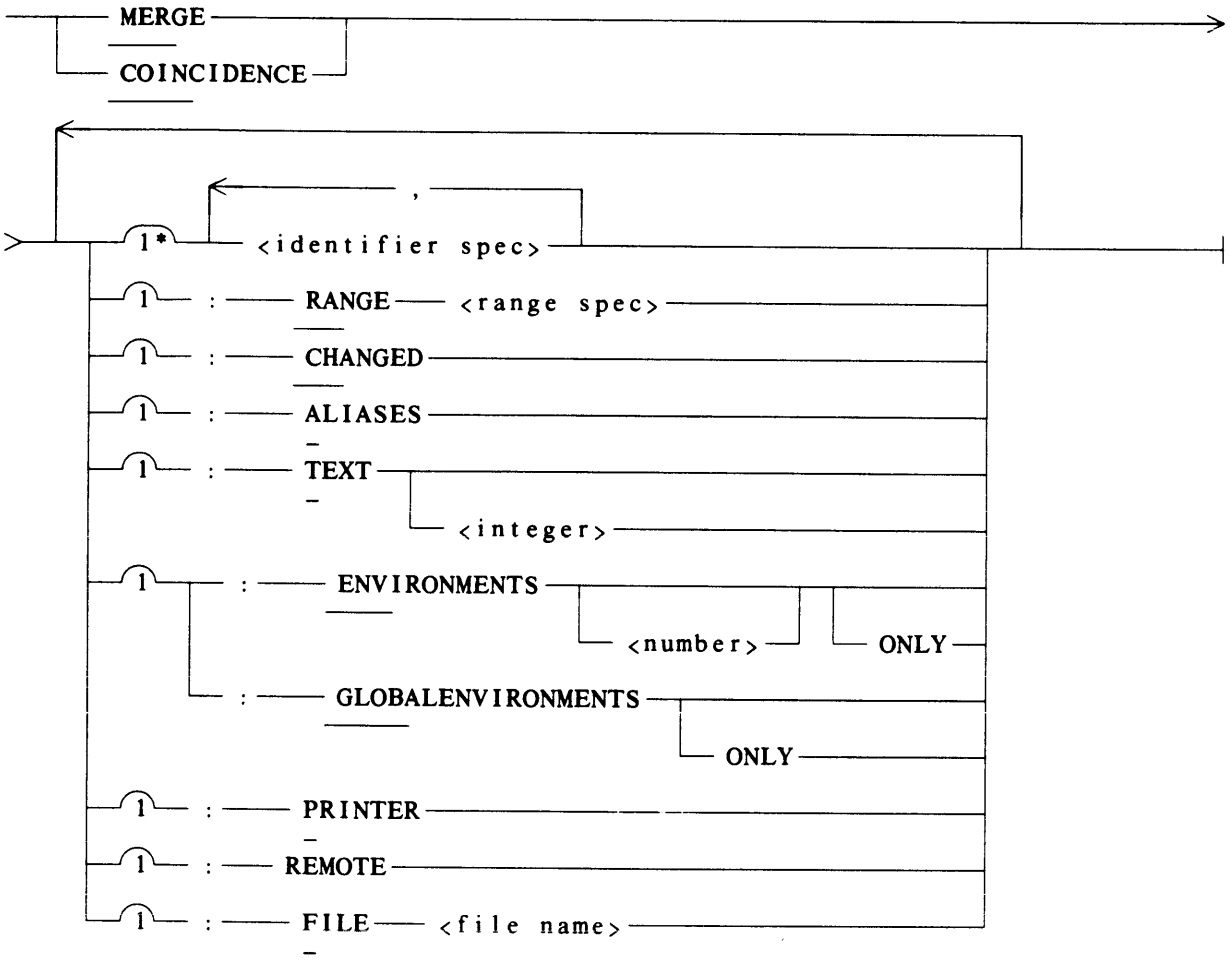
Syntax

Semantics

A summary of the number and kinds of references to a given identifier is printed. If no identifier is specified and the preceding command was LOCATE, REFERENCE, EXPAND, or SUMMARY, then the identifier specified in that command is used. This identifier, remembered from command to command, is known as the work identifier. If no identifier is specified and the work identifier is empty, an error occurs.

MERGE and COINCIDENCE

Syntax



MERGE

Semantics

A merged list of the references to the specified identifiers is produced. To avoid confusion, the work identifier is nullified. All options described under the command **REFERENCES** apply.

Examples

MERGE I, J1, ABC OF SAM :ENVIRONMENTS

Prints a merged list of references to the identifiers I, J1, and ABC OF SAM. The references are grouped by the procedures within which they occur, and the name of the procedure precedes each group.

MERGE A, G :ALIASES

Prints a merged list of references to the identifiers A and G, any aliases of A, and any aliases of G.

COINCIDENCE

Semantics

A list is produced of those places where all specified identifiers appear on the same line. To avoid confusion, the work identifier is nullified. All options described under the command REFERENCES apply, but some may be ambiguous and therefore require further explanation.

CHANGED	Produces a list of those lines where all the specified identifiers appear, and where one or more might be changed by the statement within which it appears.
ALIASES	Produces a list of those lines where all specified identifiers and all of their aliases appear.
ENVIRONMENTS ONLY GLOBALENVIRONMENTS ONLY	Only environments in which all specified identifiers appear on the same line is printed.

Caution must be used when employing the COINCIDENCE command; although the identifiers may be used in the same statement or expression, the statement or expression may be split across a line boundary.

Examples

```
COINCIDENCE K, D1
```

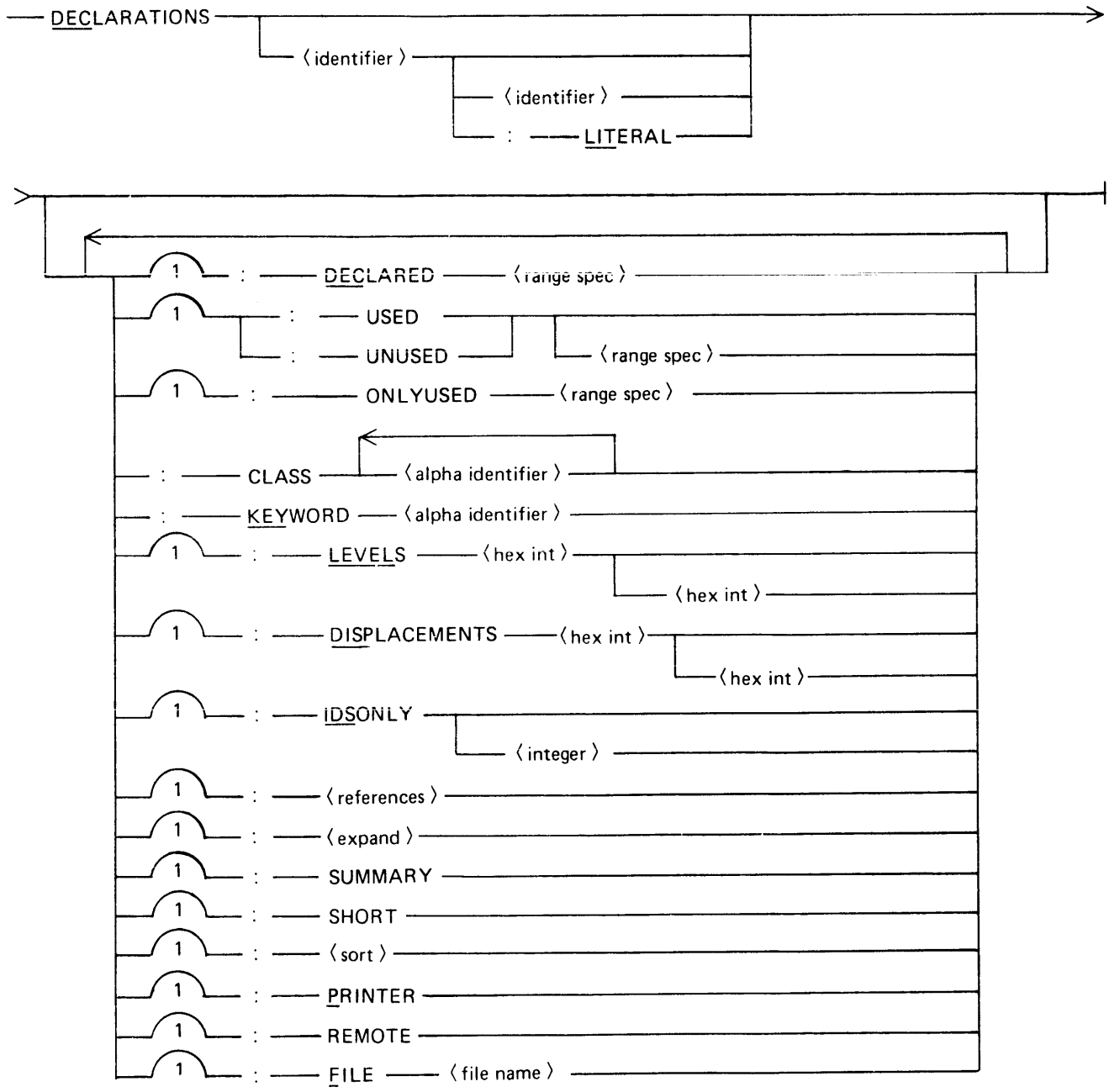
A list of those lines where both K and D1 appear is printed.

```
COINCIDENCE STACK, TASK :CHANGED
```

In this example, STACK is an array and TASK is defined to be STACK[13]. The above command produces a list of those places where the value of TASK might be changed. Note that the command REF TASK :CH produces null output, as defines are never marked as stored-into.

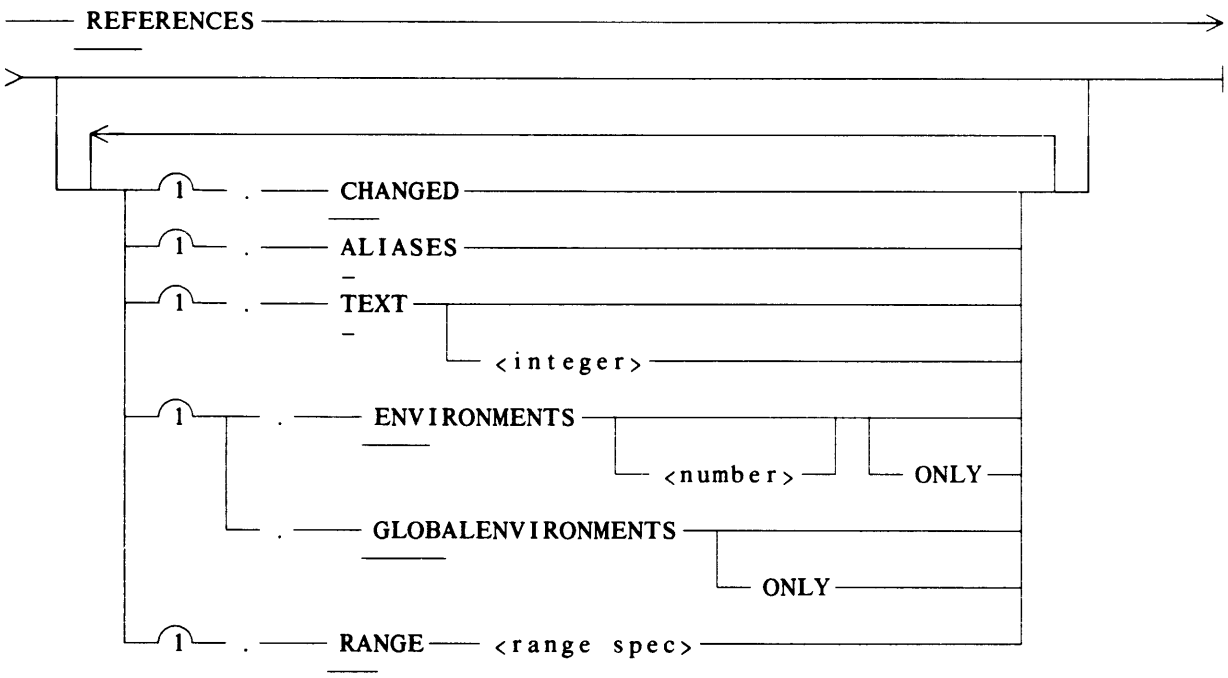
DECLARATIONS

Syntax

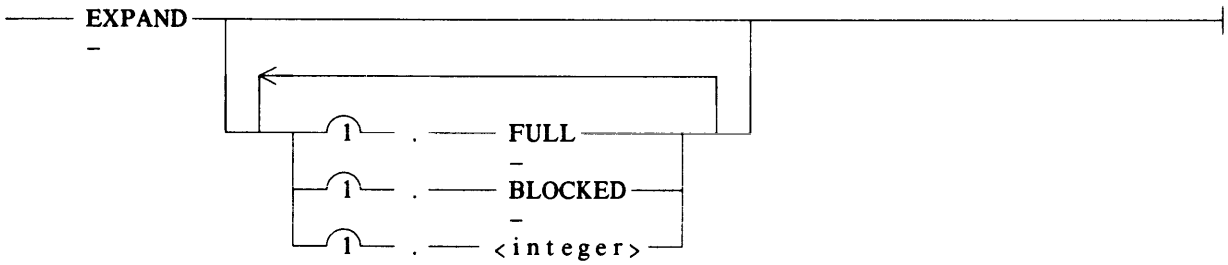


MV1454

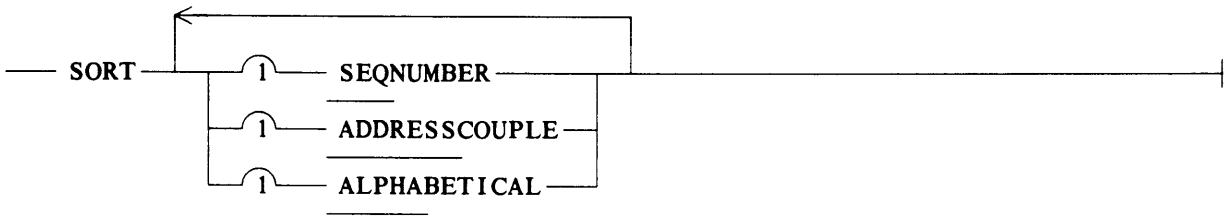
<references>



<expand>



<sort>



Semantics

A specified set of declarations is isolated and listed along with optional information. The options available fall into three categories: (1) those which select the set of declarations; (2) those which specify the output to be produced for each selected declaration; and (3) those which specify the order and destination of the output. To avoid confusion, the work identifier is

nullified.

The defaults, if no options are specified, are as follows:

1. All declarations are included.

2. The header line (name, environment where declared, compiler class, sequence number where declared, aliases, and so forth) is printed for each declaration.

3. The output is ordered alphabetically by identifier name and comes to the terminal.

Options that control the selection of the set of declarations are:

<identifier> Restricts the set to the occurrences of a given identifer.

<identifier>-<identifier>

Restricts the set to a given alphabetic range. The identifier pair must be ordered alphabetically.

<identifier> :LITERAL

Restricts the set to those identifiers that contain this specified identifier as a substring.

DECLARED Restricts the set to those identifiers declared within the specified range.

USED If no range specification is included, restricts the set to those identifiers referenced somewhere in the program. If a range is specified, restricts the set to those referenced within the range.

UNUSED If no range specification is included, restricts the set to those identifiers declared but never referenced; otherwise, restricts the set to those not referenced within the specified range.

ONLYUSED Restricts the set to those identifiers only used within the specific range and not referenced elsewhere.

CLASS Restricts the set to a particular compiler class or group of compiler classes. The compiler class must appear exactly as it does in a header line, for example, BOOLEAN ARRAY, INTEGER, FORMAL NAME REAL, and so forth. Only one compiler class may be specified for each CLASS option; however, the compiler class may contain more than one <alpha identifier> (for example, REAL PROCEDURE). CLASS may be specified as often as desired, thus specifying a group of classes.

CLASS - Restricts the set to all compiler classes except those specified in the alpha identifier

list.

KEYWORD — Restricts the set to a group of compiler classes which contain the specified alpha identifier. For example, KEY BOOLEAN causes BOOLEAN, BOOLEAN ARRAY, BOOLEAN PROCEDURE, and so on, to be included. KEYWORD and CLASS may be specified as often as desired to generate the desired group of classes.

KEYWORD - — Selects a group of compiler classes which do not contain the specified alpha identifier. For example, KEY- BOOLEAN includes exactly the complement of the classes included by KEY BOOLEAN.

LEVELS — Restricts the set to those identifiers that have stack cells with lexicographical levels as specified.

DISPLACEMENTS — Restricts the set to those identifiers that have stack cells with displacements as specified.

Options that specify the output to be produced for each selected declaration are:

IDSONLY — Displays only identifier names and not any of the other header information. The output is displayed in ascending alphanumeric order, with a default field width of 20. The field width may be altered by specifying an optional field width.

REFERENCES — References to the selected declaration are listed. This option may itself be modified by any of the options listed under the REFERENCE command, except PRINTER or REMOTE, with the same effects.

EXPAND — The text of the selected declaration is written out (expanded) if the identifier is an item such as a define, array, or file, which has text associated with its declaration. This option may itself be modified by the options FULL and BLOCKED, as described under the command EXPAND. Note that only the first or the final expansion may be obtained. (If a full expansion of a define is requested, it is done in the context of its first use.) EXPAND may be modified by an integer, which allows specification of an approximate limit on lines of text printed for each declaration. The default is ten.

SUMMARY — A summary of the number and kinds of references to the selected declaration is printed.

SHORT — The aliases of the selected declaration are not printed. Currently, only ESPOL keeps track of aliases; for other languages, SHORT has no effect.

Options that specify the order and destination of output are:

SORT	Controls the order in which the selected declarations are printed. When SORT is followed by SEQNUMBER, the output shows the declarations sorted on sequence number where declared. When SORT is followed by ADDRESSCOUPLE, the output shows the declarations sorted first on address couple (lex level and displacement of stack cell), then on sequence number where declared. When SORT is followed by ALPHABETICAL, the output shows the declarations sorted first alphabetically, then in order of occurrence. (This is the same output that would be produced if SORT were not specified.) When SORT is followed by more than one item, multiple sets of output are produced.
PRINTER	Causes output to go to the line printer by way of a file internally named LINE.
REMOTE	Causes output to come to the terminal, even if it is also going to the line printer.
FILE	The FILE option is only valid when used with the REFERENCE option. Its semantics are explained with the REFERENCE command.

Examples

```
DECLARATIONS :DECLARED JOE
```

Lists all identifiers declared within procedure JOE.

```
DEC :DECLARED -JOE :USED JOE :KEYWORD ARRAY
```

Lists all arrays global to procedure JOE and used within JOE.

```
DEC :DECLARED -JOE :USED JOE :KEYWORD PROCEDURE %
  :REFERENCES. TEXT. RANGE JOE
```

Lists all procedures global to procedure JOE which are called by JOE, along with the sequence numbers and text where they are called within JOE.

```
DEC J
```

Lists all declarations of the identifier J.

```
DEC STBR:LIT
```

Lists all declarations containing substring STBR.

```
DEC :DECLARED (B.0000) :CLASS DEFINE
```

Lists all global defines. The example assumes that the main program is a block; if it is a procedure, the procedure name should appear in the parentheses.

```
DEC :LEVEL 2 :DISPLACEMENTS 12-1A :SORT ADDRESSCOUPLE
```

Lists all lex level 2 identifiers with displacements between 12 and 1A, sorted by address couple.

```
DEC :DECLARED 201000-203000 :EXPAND :PRINTER
```

Lists all declarations declared between 201000 and 203000. The text of each declaration that can be expanded is printed below it, and output goes to the line printer.

```
DEC FAULT - FAULT999
```

Lists all declarations beginning with FAULT but not greater than FAULT999.

```
DEC :KEY PROCEDURE :DECLARED (B.0000)
  :REF. GLOBALENVIRONMENTS ONLY
```

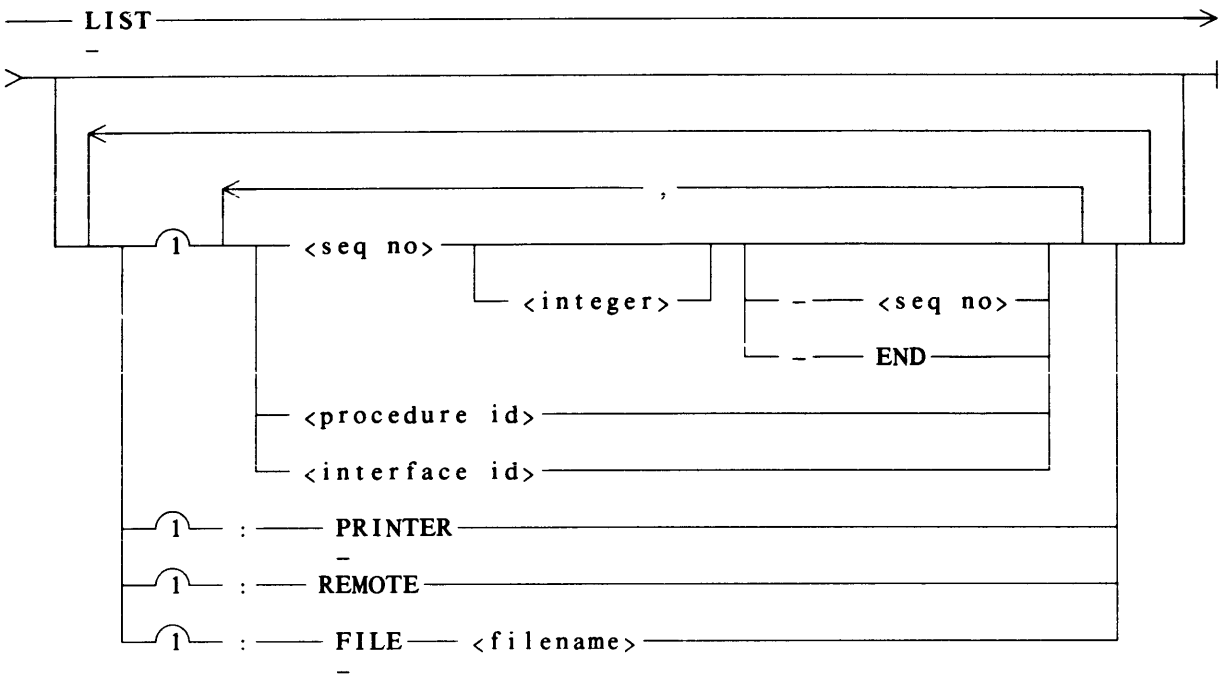
Lists all global procedures, each accompanied by a list of global procedures from which it is called.

```
DEC :UNUSED
```

Lists all identifiers that are declared but not used.

LIST

Syntax



Semantics

Text is listed from the symbol file. If no sequence numbers are specified, the entire file is listed. If sequence number(s) and/or sequence number pair(s) are specified, only those lines are listed. When the sequence number, or first sequence number of a pair, is followed by a left broken bracket (<) and an integer, the command backs up that many records before beginning to list. Entire procedures and interfaces may be listed by using the procedure or interface identifiers in place of the sequence range.

The available options are:

PRINTER Causes output to go to the line printer by way of a file internally named LINE.

REMOTE Causes output to come to the terminal even if it is also going to the line printer.

FILE Causes all lines in the sequence range or all lines of the requested procedures or interfaces to be written to the user-requested disk file; this file cannot already exist. It is made with the same file type as the file loaded by the symbol command.

RANGE

Syntax

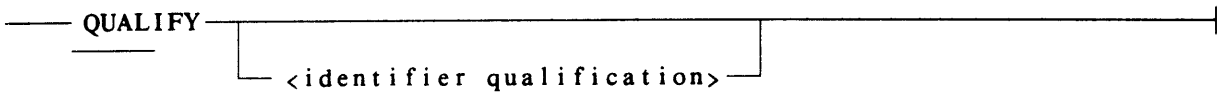
Semantics

Range is used to restrict the range over which references are listed. The default reference range is used when processing a REFERENCE command or a MERGE, COINCIDENCE, or DECLARATIONS command with no specified range.

The RANGE command establishes a new default reference range; the default value of which is the entire program. A RANGE command not containing a range specification only prints the current default reference range.

QUALIFY

Syntax



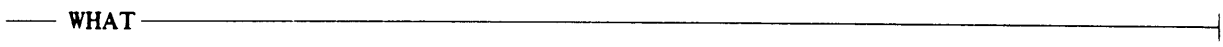
Semantics

This command establishes a new default identifier qualification (the default value is AT FIRST). When an identifier specification is encountered that does not have an explicit identifier qualification, the default identifier qualification is used to locate the identifier. A QUALIFY command not containing an identifier qualification only prints the current default identifier qualification.

Note that when an identifier specification has an explicit identifier qualification, the default qualification is overridden, rather than supplemented.

WHAT

Syntax

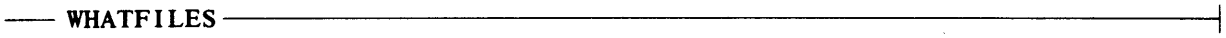


Semantics

This command describes the current work identifier to be used in subsequent REFERENCES, EXPAND, or SUMMARY commands when no identifier is specified.

WHATFILES

Syntax

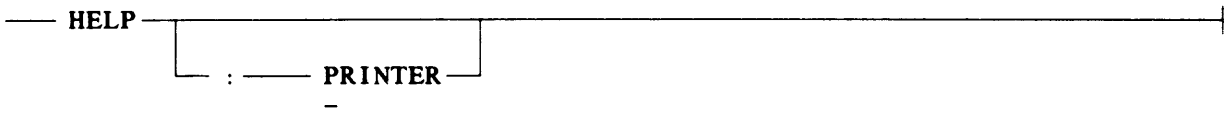


Semantics

Tells which INTERACTIVEXREF information files are loaded as well as which symbol file is loaded.

HELP

Syntax

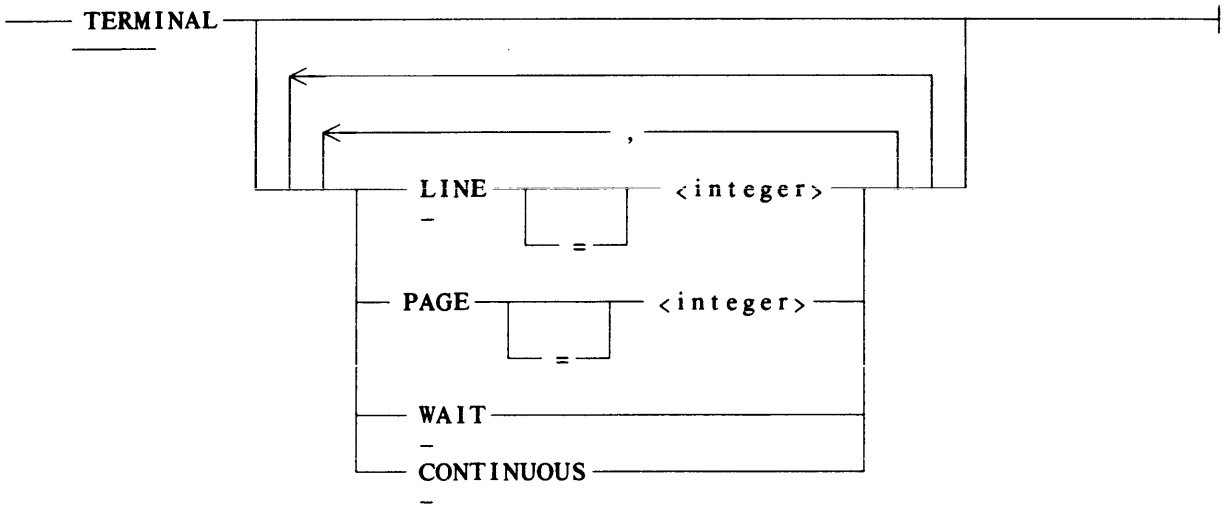


Semantics

A list of all commands, each with a short description, is displayed at the terminal. The listing goes to the printer when the PRINTER option is used.

TERMINAL

Syntax



Semantics

Attributes that control the format of output coming to the terminal may be specified. If no attributes are specified, the current terminal specifications are displayed. The initial values of PAGE and LINE are taken from the file attributes of the remote file when it is opened.

Attributes that may be specified are:

LINE — The maximum width of an output line, expressed in characters; value must be between 72 and 132.

PAGE — On screen terminals, having output grouped into pages of a given number of lines and having the program wait for some response after each page is a convenient way to look at the output. PAGE is the number of lines on each page and is relevant only if WAIT has been specified.

WAIT — The output is grouped into pages as described above. After each page, except the last page, the program stops and waits for input from the terminal. If the input is blank, the next page is printed; otherwise, the command is aborted, as if a break on output had occurred.

CONTINUOUS — Turns WAIT off.

Examples

TERM LINE 80

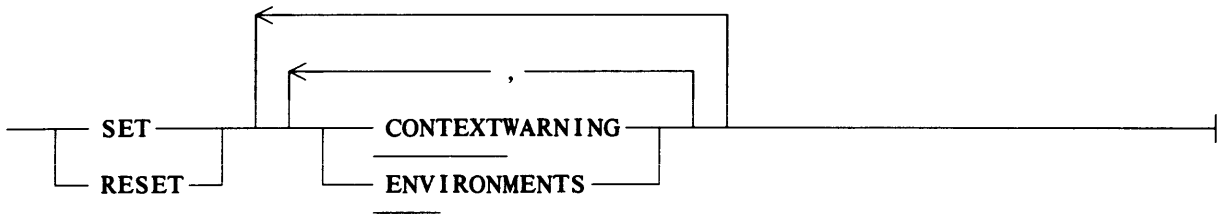
Sets the maximum width of an output line to the terminal to 80 characters.

TERM PAGE 23 WAIT

Causes the program to wait after each output page of 23 lines is printed. Note that the count starts at the beginning of each command.

SET and RESET

Syntax



Semantics

The user may set or reset the following run-time options:

CONTEXTWARNING When the EXPAND command is forced to expand a define in the context of its first use, a warning is printed. If the option is reset, the warning is suppressed. The default is SET.

ENVIRONMENTS When the description (or header line) of a non-global identifier is printed, a description of the environment, or procedure(s) within which it is declared is included. If the option is reset, environment information is not included. Default is SET.

STOP

Syntax

Semantics

Use of the STOP command terminates the program.

APPENDIX 6A

LOAD DURING INITIALIZATION

If, when the INTERACTIVEXREF program is initialized, it finds that the title of file LOAD has been label equated, it attempts to load XREF information files corresponding to the code file with that title. If, in addition, the title of file SYMBOL has been label equated, the program attempts to load a symbol file using that title.

Example

```
RUN $SYSTEM/INTERACTIVEXREF; FILE LOAD=OBJECT/IBFRITZ; %
FILE SYMBOL=IBFRITZ
```

This CANDE command initiates the INTERACTIVEXREF. The XREF information files for OBJECT/IBFRITZ are loaded, and IBFRITZ is loaded as the symbol file.

This feature may be used in conjunction with the CANDE DO command to save typing when the same XREF information files are used often.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

APPENDIX 6B

USE WITH IMPROPERLY SEQUENCED SOURCE

Care has been taken to maintain the environment information such that it is valid even if the source seen by the compiler was improperly sequenced or had no sequence numbers. The source might be improperly sequenced if, for example, it came from several different files by means of a $INCLUDE, and each file had its own sequencing scheme.

The environment information in each header line clearly identifies an identifier even if the sequence number is meaningless. The ENVIRONMENTS ONLY and GLOBALENVIRONMENTS ONLY reference options clearly list those procedures within which an identifier is used. If individual procedures are properly sequenced, then the ENVIRONMENTS and GLOBALENVIRONMENTS reference options give the procedure names and sequence numbers where the references are located. As references are sorted first by procedure, then by sequence number, all references within a given procedure are grouped together.

All forms of identifier qualification work except AT<seq no>. The form AT<seq no> IN <procedure spec> has been provided especially for cases where the specified procedure is sequenced properly and contains nested blocks and/or procedures that redeclare some of its identifiers.

Environment ranges work regardless of the sequencing of the source. Sequence number ranges produce undefined results if the source was improperly sequenced.

The EXPAND and LIST commands, the TEXT reference option, and the EXPAND declaration option do binary searches on the SYMBOL file to obtain needed text; therefore they work only if the needed text is from the currently loaded SYMBOL file and if that file is properly sequenced.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

APPENDIX 6C

USE WITH FORTRAN

FORTRAN does not keep track of environments for XREF; therefore, no environment information is included in the header line. The ENVIRONMENTS and GLOBALENVIRONMENTS reference options are not available nor are environment ranges or identifier qualifications with the exception of AT FIRST and AT<seq no>.

The EXPAND command is not available for FORTRAN XREFs.

A number (FORTRAN label) or an identifier containing dollar signs ($) is accepted as a valid identifier when FORTRAN XREF files are loaded.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

INDEX SEQUENTIAL ACCESS METHOD

TABLE OF CONTENTS

1. INTRODUCTION

The material contained in this chapter documents a set of software routines that implement indexed sequential access methods of storage and retrieval of data records. Indexed sequential, hereinafter referred to as ISAM, provides the ability to process a file sequenced by a key in both random and serial fashion. This material is intended for use as a reference document for experienced programmers; it is not intended for use as a primer.

The ISAM facility is only used by the COBOL (with or without the $ANSI74 option), PL/I, and ALGOL compilers; the COBOL74 compiler does not use this facility.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

2. STRUCTURE OF ISAM FILES

ISAM files are bound by normal file convention and must utilize features available to all files. Burroughs Data Management Software (DMSII) provides additional capabilities beyond the scope of ISAM. For ISAM, the file is considered to consist of three logical sections, which are defined in the following subsections.

PRIME DATA AREA

A prime data area is the occupied portion of the file, immediately after creation of the file. The maximum size of this area (in records) can be determined by multiplying two file attributes: AREAS and AREASIZE. AREAS and AREASIZE must be specified when creating (opening output) an ISAM file and do not need to be specified at any other time. The amount of file space reserved for nondata purposes (coarse and fine tables) is determined by the attributes AREAS and AREASIZE. The number of prime data area rows is specified by the AREAS attribute. The number of records in each prime data area row is specified by the AREASIZE attribute. ISAM files do not assume the default values of a normal file for AREAS and AREASIZE because these should be carefully chosen for optimum performance. At file creation time, all unused space contained in the final row of the prime data area is incorporated into overflow space for the final row, and totally unused prime data area rows are incorporated into the file overflow area.

DATA OVERFLOW AREA

This portion of the file is the unoccupied data area. Records added after file creation are always placed in an overflow area. Two types of physical overflow areas are provided. Area overflow space may be provided in each row containing prime data and is specified when the file is created (opened output). When records are deleted, the occupied space can be returned to the overflow pool where the record resides. The deleted record option is set at file open when the file is created and specifies the disposition of the space occupied by the deleted record. A file overflow area may also be specified at file open when the file is created. Records are placed in the file overflow area only after all available overflow space in the specific row where the record would normally reside has been filled.

TABLES FOR LOCATING DATA

Two levels of tables are used by the ISAM procedures: fine tables and coarse tables. Each prime data area row of the file contains a fine table. The fine table is a list of keys and file addresses with one-to-one correspondence. One key (and address) is placed in the table for each N records, where N is a program selected value at file creation time. The fine table is stored at the physical end of its corresponding file area.

The entire file has one coarse table that contains pairs of keys and addresses. Each key entry is identical to the first key entry of the corresponding fine table and the address entry is the address of the fine table rather than the address of a data record. The coarse table is stored at the physical beginning of the file overflow area. Therefore, an ISAM file must have at least one physical row of file overflow space.

DATA RECORD LINKS

ISAM data records are linked together in a logical sequence. Each record contains both forward and backward links to its logical successor and predecessor. A link is an address of a data record. The first data record contains a backward link that is zero, and the last data record contains a forward link that is zero. Locating data records makes use of forward links. Inserting and deleting data records makes use of both links. Data record links are the innermost level of file structure in an ISAM file.

The coarse table serves to locate a fine table, and the fine table locates a data record. The data record links are utilized in following the trail to the desired record when necessary. Data records are not physically moved to accommodate additions and deletions. Instead, links are modified as file changes are handled in a logical rather than physical fashion. Since links are physically contained in every data record, ISOPEN must increase the record size to provide space for the links. Increasing record size is accomplished by rounding the original up to the nearest full word and then adding one more word to contain the links.

MANAGEMENT OF OVERFLOW AREAS

When the file is created, unoccupied space may be reserved in each prime data area row and at least one entire row reserved for overflow records. The fine table that corresponds to a row also contains information that provides a link to the next available unoccupied space. Overflow space that is reserved at file creation is allocated in a serial fashion. If deleted record space is made available for reuse (an option selected by the program), the deleted record(s) are linked into the available record chain for the corresponding area and reassigned on a last in, first out (LIFO) basis. Record space made available for reassignment is reused prior to assignment of unused space.

The coarse table contains the link to the next available space in the file overflow area. Space assignment in the file overflow area is the same as overflow assignment in a prime data row. New records are not placed in the file overflow area if they can be placed in the prime data area. A given record is never eligible for placement in more than one prime data area row, and the only alternative placement for it is in the file overflow area.

3. GENERAL IMPLEMENTATION INFORMATION

ISAM is implemented by a set of procedures that are bound into the intrinsic file. Symbolics for these procedures are contained in the PLINTRINSICS symbol file. The procedures called directly from programs are as follows:

1. ISOPEN - Open and set up file.
2. ISCLOSE - Close file.
3. ISREAD - Randomly read a record.
4. ISWRITE - Add a record to the file.
5. ISREADNEXT - Read the next sequential record.
6. ISREWRITE - Rewrite the record just read.
7. ISKEYWRITE - Randomly rewrite a record.
8. ISDELETE - Delete a record.

These procedures must be utilized to OPEN, CLOSE, create, and access ISAM files. Files that are not indexed sequentially may not be accessed by these procedures. Normal file (non-ISAM) OPEN, CLOSE, READ, and WRITE statements are not disallowed; however, the use of normal I/O statements may be detrimental to the integrity of the ISAM file. The ISCLOSE should be used to close the ISAM file.

PROGRAM INTERFACE

The following paragraphs define the two ways of invoking ISAM procedures.

Primitive ISAM Procedure

The primitive ISAM procedure is a direct call on the procedure by name, passing the required parameters and receiving the procedure results. This method, which provides the primitive interface, allows the highest amount of selection and control. ALGOL must use the primitive method; COBOL may select either primitive or standard.

Standard ISAM Procedure

The standard interface simplifies programming effort by allowing normal, higher level language, input/output statements such as READ and WRITE. PL/I must utilize the standard method; COBOL may select standard or primitive. Different features are implemented specifically for PL/I and COBOL and are uniquely available in a particular language. This implementation level is intended to meet requirements of a language standard.

ISAM procedures must be utilized directly by the programmer for the primitive method but may be utilized indirectly by the programmer using the standard method. Indirect means the compiler supplies the procedure call and does not imply loss of efficiency.

Functionally, ISAM file options are similar to file attributes but exist only for ISAM files. Unlike file attributes, ISAM file attributes may not be set or modified by control cards or programatic file attribute statements. The options are set at file creation when the file is opened output.

ISAM procedures return an information word to reflect the results of the ISAM invocation. For the standard implementation, the compiler emits code for observing the results and initiating appropriate action. The programmer must detect the exception conditions in the primitive implementation. ISAM error conditions do not cause program termination in the primitive method but may cause termination in the standard method. The program normally contains provision for processing the exceptions in both methods. The primary difference between the methods is in the detection and analysis of the condition.

PRACTICAL CONSIDERATION

In an unstable environment, ISAM files can become volatile (a condition where the coarse table, fine tables, or data record links do not concur). This situation can exist when the physical file (on disk or disk pack) has not yet been updated to reflect the changes that have been made to buffers in memory. If an event occurs that prevents writing the updated buffers into the physical file, the file may suffer a loss of integrity. Use of the standard method helps prevent this situation. In those cases where the program terminates prematurely (invalid index, divide by zero, DSed, and others), the standard method performs an orderly close of the ISAM file while the primitive method may not be able to properly close the file. However, the file must be opened INPUT-OUTPUT and writes or deletions must occur as prior conditions. File damage is by no means a certainty, and two file options are available to further reduce such possibility. See the WAITUPDATEIO and PHYSICALUPDATE options of the ISOPEN discussion.

ISAM files may not be specified as input or output files to the SORT, except in PL/I. They must be read and written by input or output procedures. Other system software may also encounter similar situations when attempting to process ISAM files in a direct fashion without use of the ISAM procedures.

ISAM files may be accessed simultaneously by several programs, if they all open the file as INPUT. Only one program may access the file while it is open OUTPUT or INPUT-OUTPUT.

Direct I/O is used by ISAM procedures to access the data. Therefore, the ISAM file must be a direct file. In the primitive method, the program must declare the file as direct. The compiler properly declares the file in the standard method. The direct arrays used by the ISAM procedures are created by ISOPEN and returned by ISCLOSE. The program does not need other direct arrays or record areas (in COBOL) to access the data.

ISAM files may not be used in an IPC environment where the file is passed from one task to another.

ISAM PROCEDURES

The following subsections discuss the system procedures that collectively institute the ISAM methodology. Each procedure is defined in terms of its function or functions within the general ISAM operating method and in terms of the interaction, if any, with other ISAM procedures. Direct knowledge of these procedures and their parameters is not needed for the standard program interface.

The language reference manuals for PL/I (form 5001530) and COBOL (form 5001464) should be consulted for the description of all syntax and operations on ISAM files using the standard program interface.

ISOPEN

This procedure opens an ISAM file for INPUT, OUTPUT, or INPUT-OUTPUT. ISAM files require additional information not provided for non ISAM files. ISOPEN utilizes and creates the additional information according to the method of file opening. Non-ISAM files may not be opened by this procedure.

PROGRAM CALLING SEQUENCE:

ALGOL: RS := ISOPEN(FILE,VALUE,STACK);

COBOL: (NON-STANDARD)

```
COMPUTE RS = ISOPEN (FILE, VALUE, STACK).
```

RS is the result word returned to the program. It is type BOOLEAN in ALGOL and 77 level COMP-1 in COBOL.

FILE is the ISAM file being opened. It must be declared as a DIRECT FILE in ALGOL and COBOL.

VALUE specifies how the file is to be opened.

1 = open as INPUT

2 = open as OUTPUT

3 = open as INPUT-OUTPUT

INPUT and INPUT-OUTPUT require an existing ISAM file. OUTPUT always means creation of a new file.

OUTPUT requires specification of additional file information. High order bits in this parameter (VALUE) are utilized to convey certain information used for file creation. Bits and fields contained in this parameter are as follows:

47:1 Separate key (PL/I only).

46:15 Offset of the key, in bytes, from the start of the record. It is the true (zero relative) offset. A value of zero means the start of the record.

31:2 Open action (open input or I/O only).

- 0 - Open the file.
- 1 - Use PRESENT attribute to open.
- 2 - Use AVAILABLE attribute.
- 3 - Not used.

29:14 Actual key length in bytes.

15:4 Mode of key. Values are:

- 0 - BINARY (6-byte maximum)
- 1 - 8-Bit character
- 2 - 8-Bit unsigned numeric (max 11 bytes)
- 3 - 8-Bit MSD signed numeric (max 11 bytes)
- 4 - 8-Bit LSD signed numeric (max 11 bytes)
- 5 - 4-Bit characters
- 6 - 4-Bit unsigned numeric (max 5 bytes)
- 7 - 4-Bit MSD signed numeric (max 6 bytes)
- 8 - 4-Bit LSD signed numeric (max 6 bytes)

11:1 Duplicate key option. If zero, records with duplicate keys may not be added to the file. If one, duplicates are chained in first in, first out (FIFO) sequence. A duplicate key condition exists when the keys in two records are equal.

10:1 Deleted record option. If zero, deleted records are physically delinked and their record space becomes available for reuse. If one, deleted records are flagged by having 4"FF" (all bits on) placed in the first byte of the record. Records marked as deleted can be retrieved using READNEXT if bit 2 of this parameter word equals 1.

9:1 Sequence option. If zero, the file is in ascending sequence. If one, the file is in descending sequence.

8:6 Fine table ratio. During file creation, this field controls the number of entries made in the fine table(s). It specifies the number of unique records to be added to the file between fine table entries.

2:1 See deleted record option. If zero, deleted records are not "seen" by the program. If one, deleted records may be "seen" if the deleted record option is set and READNEXT is used.

1:2 Open type (previously described).

- 0 - invalid
- 1 - INPUT
- 2 - OUTPUT
- 3 - INPUT-OUTPUT

STACK specifies the first of four consecutive words in the programs stack. The location of the first word is utilized by ISOPEN to build data descriptors in all four words. The location is retained in the FIB for use as long as the file remains open. The program must provide the space by declaring the four consecutive stack locations preferably with four type REAL variables in ALGOL and four usage COMP-1 in COBOL. The four words are not usable by the program while the ISAM file is open. A program reference to any of the four words during the time the file is open causes immediate program termination with an invalid operator.

Additional file information is conveyed to ISOPEN by use of the first of the four consecutive words.

47:24 Number of overflow records per prime record area row. This field is used only when the file is opened output. At file creation, this field is used to increase the AREASIZE specified for the file. The new, larger AREASIZE becomes a permanent attribute of the file. Unoccupied space, large enough to contain the number of records specified by this field, is allocated in each row of the file.

23:1 Wait update I/O option. If one, this option causes ISAM procedures to wait for I/O completion of all outstanding I/Os before returning to the program. This option is not available if ANSI74 is set in COBOL.

22:1 Physical update I/O option. If one, this option causes the ISAM procedures to initiate I/Os for all buffers and tables that have been modified and need to be rewritten. This option is not available if ANSI74 is set in COBOL.

21:6 Unused at present but reserved for future implementation.

15:16 Number of file overflow area rows. This field is used only when the file is opened output. At file creation, this field is used to increase the AREAS attribute specified for the file. The new, larger area becomes a permanent attribute. Any areas represented by this field are not used to contain prime data. Prime data area rows unused at file creation are, however, placed in the file overflow area pool. Therefore, when in doubt, it is better to make the AREAS attribute larger.

ISCLOSE

This procedure closes an ISAM file in an orderly but necessary fashion. The normal CLOSE statement is not sufficient to properly close an ISAM file. Certain additional file information is saved within the file by this procedure, and the four consecutive stack words are cleared or restored. Non-ISAM files may not be closed by this procedure.

PROGRAM CALLING SEQUENCE:

ALGOL: RS := ISCLOSE (FILE, TYPE);

COBOL: (Non-Standard)

```
COMPUTE RS = ISCLOSE (FILE, TYPE).
```

RS is the result word returned to the program. It is type BOOLEAN in ALGOL and 77 level COMP-1 in COBOL.

FILE is the ISAM file to be closed.

TYPE is a numeric value that specifies how the file is to be closed.

0 - Close the file and release it from the program. This is a normal close. The file does not remain on disk unless it has been locked previously.

1 - Close the file with lock. The file is entered into the directory and remains on disk. Any previous file with a duplicate name may be removed.

2 - Close the file and purge its entry from the directory. Any disk space occupied by the file becomes available for reassignment by the system.

ISREAD

This procedure reads a record in a random fashion using the program-supplied key. If the program-supplied key matches a record in the file, the matching record is returned. When no matching record exists, the next logically sequential record is returned.

PROGRAM CALLING SEQUENCE:

ALGOL: RS := ISREAD (FILE, KEY, AREA);

COBOL: (non-standard)

```
COMPUTE RS = ISREAD (FILE, KEY, AREA).
```

RS is the result word returned to the program. It is type BOOLEAN in ALGOL and 77 level COMP-1 in COBOL.

FILE is the ISAM file.

KEY is supplied by the program and is used to find a record with a matching key.

AREA is supplied by the program and provides space to contain the record. The area must be as large or larger than the record. Before returning to the program, the matching (or next logical) record is placed in the program-supplied area.

The file must be opened INPUT or INPUT-OUTPUT in order to read records. This procedure may not be used to read non-ISAM files. The ISAM file must be opened by ISOPEN prior to use of this procedure to read records.

ISWRITE

This procedure writes a record, using the provided key, from the provided area. This procedure never overwrites or rewrites previously existing records but always adds (or attempts to add) records to the file.

When the file is opened OUTPUT, a new file is created; this procedure is used to create coarse and fine tables in addition to placing records into the file. Records must be presented in the sequence specified by the program during file creation. Duplicate record acceptance depends on the setting of the duplicate key option.

When the file is opened INPUT-OUTPUT, a previously existing file is utilized. Records need not be presented in any special sequence. The records are written into area overflow or file overflow space and appropriately linked into the file.

PROGRAM CALLING SEQUENCE:

ALGOL: RS := ISWRITE (FILE, KEY, AREA);

COBOL: (non-standard)

```
COMPUTE RS = ISWRITE (FILE, KEY, AREA).
```

RS is the result word returned to the program. It is type BOOLEAN in ALGOL and 77 level COMP-1 in COBOL.

FILE is the ISAM file.

KEY is supplied by the program and is used to identify the record. The key contained in the record must match the parameter.

AREA is supplied by the program and provides space to contain the record. The area must be as large or larger than the record. The record contained in this area is logically placed in the file.

The file must be opened OUTPUT or INPUT-OUTPUT. This procedure may not be used for non-ISAM files. The file must be opened by ISOPEN prior to execution of this procedure.

ISREADNEXT

This procedure reads the next logically sequential record. The record returned to the program is the record whose key immediately follows in sequence after the most recent record obtained by ISREAD or ISREADNEXT.

PROGRAM CALLING SEQUENCE:

ALGOL: RS := ISREADNEXT(FILE, AREA);

COBOL: (non-standard)

```
COMPUTE RS = ISREADNEXT (FILE, AREA).
```

RS is the result word returned to the program. It is type BOOLEAN in ALGOL and 77 level COMP-1 in COBOL.

FILE is the ISAM file.

AREA is supplied by the program to provide space for the record. The area must be as large or larger than the record. Before returning to the program, the next logical record is placed in the area. The file must be opened INPUT or INPUT-OUTPUT to use this procedure. Non-ISAM files may not be accessed with this procedure. The file must be opened by ISOPEN.

The purpose of this procedure is to provide a sequential processing capability. In combination with ISREAD and ISREWRITE, records may be sequentially processed and updated for all or part of any ISAM file. ISREADNEXT may be used to read an entire ISAM file in a sequential manner.

ISREWRITE

The ISREWRITE procedure replaces the record previously read with the data currently in the record area. The immediately preceeding file operation must be ISREAD or ISREADNEXT. The key contained in the record to be rewritten must match the key in the record that was read by the immediately preceeding file operation.

PROGRAM CALLING SEQUENCE:

ALGOL: RS := ISREWRITE(FILE, AREA);

COBOL: (non-standard)

```
COMPUTE RS = ISREWRITE (FILE, AREA).
```

RS is the result word returned to the program. It is type BOOLEAN in ALGOL and 77 level COMP-1 in COBOL.

FILE is the ISAM file.

AREA is supplied by the program to provide space for the record to be rewritten. The area must be as large or larger than the record. Before returning to the program, the record contained in the area replaces or rewrites the latest record read.

The file must be opened INPUT-OUTPUT and must be an ISAM file. The purpose and function of this procedure is to provide an update capability. Additional records may not be added to the file by ISREWRITE.

ISKEYWRITE

The ISKEYWRITE procedure provides a random access update capability for ISAM files. It replaces a currently existing record from the file with the record provided by the program.

PROGRAM CALLING SEQUENCE:

ALGOL: RS := ISKEYWRITE(FILE, KEY, AREA);

COBOL: (non-standard)

```
Compute RS = ISKEYWRITE (FILE, KEY, AREA).
```

RS is the result word returned to the program. It is type BOOLEAN in ALGOL and 77 level COMP-1 in COBOL.

FILE is the ISAM file.

KEY is supplied by the program and is used to find a record with a matching key. This key must also match the key contained in the record area.

AREA is supplied by the program to provide space for the record to be written. The area must be as large or larger than the record. Before returning to the program, the record contained in the area replaces or rewrites the record identified by the program supplied key.

The file must be opened INPUT-OUTPUT and must be an ISAM file. This procedure provides update capability and does not add additional records to the file.

ISDELETE

This procedure is used to drop or delete records from the file. When duplicate records are allowed, the first (oldest) record is deleted. This procedure provides random access delete capability.

PROGRAM CALLING SEQUENCE:

ALGOL: RS := ISDELETE(FILE, KEY);

COBOL: (non-standard)

```
COMPUTE RS = ISDELETE (FILE, KEY).
```

RS is the result word returned to the program. It is type BOOLEAN in ALGOL and 77 level COMP-1 in COBOL.

FILE is the ISAM file.

KEY is supplied by the program and is used to find a record with a matching key.

The file must be opened INPUT-OUTPUT and must be an ISAM file. This procedure deletes the first (and oldest) record with a matching key. The record may be physically or logically deleted depending on the DELETED RECORD OPTION.

4. ISAM I/O RESULT INFORMATION

The ISAM procedures return result values to the calling program which indicate success or failure of the program request. The value returned is a 48-bit word. In the primitive method, the word is type BOOLEAN in ALGOL and COMP-1 or COMP in COBOL. In the standard method, PL/I uses CONDITION CODES and COBOL uses FILE STATUS. The B 7000/B 6000 Series PL/I Reference Manual, form 5001530, describes CONDITION CODES. FILE STATUS is described in the B 7000/B 6000 Series Cobol Reference Manual, form 5001464.

PRIMITIVE METHOD

The value returned for the primitive method is a 48-bit word that is non-zero when an exception condition exists and zero when no exception condition occurs. Specific, individual bits are utilized to indicate the exception condition. If several different exceptions occur, the corresponding bit is turned on for each condition and creates the possibility of reporting back several exceptions for a single request. The rightmost and least significant bit (bit 0) is used for a specific purpose. Bit 0 is turned on when any exception condition occurs and turned off when no exception exists. The remaining bits convey the following meanings:

1:1 A hardware error, a parity error for example, occurred while processing the request. Another bit (7, 8, 9, or ten) is set to further define the problem.

2:1 An attempt was made to read or write beyond end-of-file.

3:1 No record was found in the file whose key matches the requested key.

4:1 No space is available in the file to contain the record just written. (Applies for adding records to the file; does not apply to file creation.)

5:1 A request was made to add a record to the file, and the key contained in the record matched a record that existed in the file. Refer to bit 6.

6:1 A record was added to the file (the key of the record matched an existing record of the file). The duplicate key option permits or disallows this situation. When duplicates are allowed, both bit 5 and bit 6 are on to indicate adding a duplicate record. See also bit 5.

7:1 A hardware error occurred in reading a data record. Bit 1 is also on.

8:1 A hardware error occurred in writing a data record. Bit 1 is also on.

9:1 A hardware error occurred in reading an ISAM table. Bit 1 is also on.

10:1 A hardware error occurred in writing an ISAM table. Bit 1 is also on.

11:1 This bit is not used.

12:1 An attempt was made to open the ISAM file, and the parameters passed to ISOPEN failed to meet one or more requirements. The "first of four stack words" parameter must be an SIRW. The file must be declared in a block that will be entered no sooner than the block where the four stack words reside. The file must not reside in a different stack from the program doing the open. The block containing the four stack words must also contain a file, array, or something that causes a tag-6 word for the block. The tags of all four stack words must be zero. The key must be defined to be contained in the records, have a size greater than zero, and have a valid mode.

13:1 An attempt was made to open a non-ISAM file.

14:1 The file has not been opened or the open type does not permit the request. For example, a write on file opened INPUT.

15:1 A rewrite was requested, and the key of the record being rewritten does not match the key in the last record read or the previous request was not a read or read next.

16:1 The ISAM file is being created, and the record just written did not maintain proper file sequence. Records must be presented in sequence during file creation. A duplicate record also causes this bit to be set when duplicates are not allowed.

17:1 The number of AREAS specified is not large enough to contain the data records written in the prime data area during file creation.

18:1 ISOPEN is requested to open an already open file.

19:1 In an IPC environment, one program closed an ISAM file and another attempted an I/O after the file was closed.

20:1 A write is requested, and the key supplied does not match the key contained in the supplied record.

21:1 The ISAM file is not a direct file.

22:1 An attempt was made to write a record containing the deleted record indicator (hex FF in first byte).

23:1 This bit indicates a PL/I program error condition. The program is requesting an I/O that is not allowed for keyed files. An on condition is raised in the PL/I program.

24:1 This bit is on if ISOPEN is requested (by way of open action) to open the ISAM file using the PRESENT or AVAILABLE file attributes; this bit also indicates that the desired file could not be located. Refer to bits

43:8.

43:8 This field contains the result of testing the PRESENT or AVAILABLE file attributes in the ISOPEN procedure. If the file could not be opened, bit 24:1 is also on.

46:1 This bit is on if the physical update I/O action is on, the wait update I/O option is off, and an I/O error occurred as the result of doing an update I/O in the previous invocation of an ISAM procedure. Bit 1:1 is on, and 8:1 or bit 10:1 is on.

CONDITION CODES FOR PL/I KEYED I/O

CODE	CONDITION
0806	End-of-file occurred.
0809	End-of-file occurred on keyed write during file creation.
1303	A hardware error occurred on an I/O while processing a KEYED I/O request.
1304	A hardware error occurred on a keyed record read.
1305	A hardware error occurred on a keyed record write.
1306	A hardware error occurred on a keyed table read.
1307	A hardware error occurred on a keyed table write.
2401	The record for the supplied key was not found.
2402	No space exists to add the record to the file.
2403	An attempt was made to add a duplicate record.
2404	A duplicate record was added to the file.
2405	Not used for keyed I/O.
2406	An attempt was made to create a file with records not in sequence.
2407	Not used for keyed I/O.
2408	Read did not precede rewrite.
2409	Not used for keyed I/O.
2410	Keyfrom does not match record key
2411	Record contains hex FF in first byte.
2412	Keyto variable is shorter than key in file.
2413	Key or keyfrom is longer than key in file.
2505	Parameter error occurred on keyed file open.
2506	An attempt was made to open an existing nonkeyed file as a keyed file.
2507	Conflicting file usage occurred. (For example, read on file opened output.)
2510	Keyed I/O was attempted when the file was in an improper state.
2511	The keyed file is not a direct file.

INDEX SEQUENTIAL ACCESS METHOD

2512 An illegal I/O was attempted on a keyed file.

2513 An attempt was made to open a nonpresent file.

FILE STATUS IN COBOL

COBOL FILE STATUS is available in the standard implementation only. A COBOL compiler control card (dollar option card) which sets the ANSI74 option enables the use of FILE STATUS and indexed I/O. FILE STATUS is a two character EBCDIC item where the first character indicates the problem area, if any, and the second character provides further definition.

CODE	DEFINITION
00	Indicates no problem or exception exists.
02	A duplicate record was added to the file.
10	End-of-file occurred while processing the request.
21	An attempt was made to write a record out of sequence during file creation.
22	An attempt was made to add a duplicate record to the file.
23	The record required to fulfill the request cannot be found. For sequential files, improper sequence of requests can cause this exception condition.
30	A hardware error occurred while processing the request.
90	An attempt is being made to use the file in an incorrect manner (for example, write on a file opened input).

5. PLANNING FOR ISAM FILES

ISAM provides a specific set of capabilities that must be considered during preparation for application programs and systems. Trade-offs can be made to favor a particular course of action. All features of the ISAM procedures are not available to every mode of operation and language. The following discussion of various items is intended to provide some practical insight into ISAM usage.

MAXIMUM NUMBER OF RECORDS

The maximum number of records that can be contained in a single ISAM file is 16,777,215. Some space is required for a coarse table, fine tables, and an INFO record. Data records can occupy the remaining space.

COARSE TABLE SIZE

One coarse table is created for the entire file.

Coarse table size is determined by the number of prime data area rows used during file creation; however, the number of rows used cannot exceed the number of AREAS requested because the coarse table cannot expand. Not more than 999 prime data area rows may be requested because at least one row is required for file overflow. One entry is made in the coarse table for each prime data area row. The table is contracted when fewer prime data area rows are used than specified.

For file creation, a few more areas than needed should be specified. Key length also has a direct effect on table size. The default number of areas is one.

COMPUTING COARSE TABLE SIZE

All units are bytes (8-bit characters).

Coarse table size = (number of table entries * (key length + 3)

+ 24 + BLOCKSIZE - 1) DIV BLOCKSIZE * BLOCKSIZE.

Record space loss to coarse table = coarse table size DIV

BLOCKSIZE * number of records per block.

FINE TABLE SIZE

One fine table is created for each prime data area row. A prime data area is a row of the file that was written into while the file was being created. All other rows of the file are allocated to file overflow and do not contain fine tables. In a given file, all fine tables have identical size. A fine table ratio is specified to determine the number of entries in each fine table. The ratio may range between 1 and 63; the default value is 1. When duplicate records are permitted, only the first record in the duplicate set is eligible for entry in the fine table or is counted in meeting the fine table ratio. Key length also directly affects fine table size.

COMPUTING FINE TABLE SIZE

All units are bytes (8-bit characters).

Fine table size = (number of table entries * (key length + 3)

+ 24 + BLOCKSIZE - 1) DIV BLOCKSIZE * BLOCKSIZE.

Record space loss to fine table = fine table size

DIV BLOCKSIZE * number of records per block

* number of fine tables.

INFO RECORD SIZE

The first record of each ISAM file is a special record that contains attribute type information and is essential for proper access of the ISAM data records. The current length of the INFO record is 7 words. One data record space is normally required to contain the INFO record but more may be used if the data record length is less than 7 words (42 bytes).

AREAS AND AREASIZE

The file attributes AREAS and AREASIZE are more important for ISAM files than other normal files. Both attributes must be specified when creating a new file and are not needed at other times. The default value is 1 for either attribute (normal file defaults are different). AREAS indicates the number of prime data area rows expected and cannot exceed 999. AREASIZE, as specified by the program, gives the number of data records per area. The AREASIZE attribute is increased to allow the fine table to be written in the same area (or row) of the file as the data it represents. AREASIZE is also increased by the number of overflow records per area which are specified by the program. The AREAS attribute is increased by the number of file overflow areas specified by the program. This increase is very similar to allowing the file to expand via the flexible attribute and does not affect coarse table size. When the ISAM file is closed, the AREAS and AREASIZE attributes are reset to their original values.

MINIMUM RECORD SIZE

ISAM does not provide for variable length records. Therefore, the attribute MINRECSIZE should be zero or identical to the attribute MAXRECSIZE.

MAXIMUM RECORD SIZE

The value chosen for the MAXRECSIZE attribute is entirely dependent upon the needs of the program and the absolute limits allowed by the system. ISAM increases the value of this attribute by at least 1 word (6 bytes) and at most 11 bytes. The program should not set this file attribute, except when creating the file. When the ISAM file is closed, the MAXRECSIZE attribute is reset to its original value. The maximum usable values are 65,534 words or 65,535 characters.

BLOCKSIZE

The BLOCKSIZE attribute is used in association with the MAXRECSIZE attribute to determine the number of records per block. BLOCKSIZE is always changed by ISAM. When the program specifies a non-zero value for BLOCKSIZE, ISAM retains the number of records per block specified by the program. The BLOCKSIZE is increased to accommodate larger records. When the program specifies a BLOCKSIZE of zero, ISAM computes a value for BLOCKSIZE specifically to conserve disk storage space. ISAM computes the smallest number of records, after MAXRECSIZE has been increased, that exactly fits into a multiple of 30-word disk segments. The BLOCKSIZE attribute is reset to its original value when the file is closed.

EXCLUSIVE USE

The EXCLUSIVE attribute is set true by ISAM when the file is opened I/O. ISAM files may not be shared except when all programs open the file as input only.

FINE TABLE RATIO

The program may select any value between 1 and 63 with 1 being a table entry for each record for the fine table ratio. The most successful choice is highly data and program dependent. Lower ratios are favorable when the file is volatile, and ratios that are close to the number of records per block perform well for files that are more constant. Programatic reorganization of the file may be performed after a number of changes have occurred in order to improve or restore performance.

KEY LENGTH

Some key types (modes) allow specific maximum key lengths of 5, 6, or 11 bytes. Character keys of 4-bit or 8-bit characters are limited only by the 14-bit field that contains key length. The maximum usable key lengths are (1) 8-bit for 1020 bytes and (2) 4-bit for 508 bytes (1016 hex characters). Shorter keys yield faster performance.

KEY OFFSET

A 15-bit field is allowed which permits an offset of 32767 for 8-bit characters and 16382 for 4-bit characters.

TABLE OF CONTENTS

1. INTRODUCTION

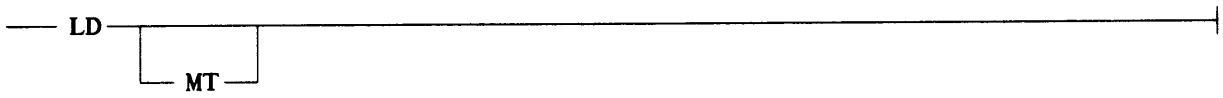
LOADCONTROL is a utility procedure contained in the MCP and is initiated by the LD Operator Display Terminal (ODT) input message.

LOADCONTROL provides two major functions. The first function copies control cards and associated data decks, if any, to magnetic tape. The second function causes the card images on tape to be processed by the system at a later time as a stream of control cards and data decks.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

2. LD INPUT COMMAND

Syntax



Semantics

All options of the LD input command require an input file titled CONTROLDECK. The last image of the file must be ?END CONTROL.

The LD option causes the input file to be copied to the Halt/Load unit and causes the control cards contained within the file to be processed.

The LD MT option causes the input file to be copied to an output tape titled CONTROLDECK. .The tape contains 14-word card-image records and is identified as a load control tape by the fact that byte 31 (USASI tape type) of the VOL1 label record is a 4. The tape may be used as input to the LD option.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

MATHEMATICAL INTRINSICS

TABLE OF CONTENTS

1. INTRODUCTION

This chapter describes the mathematical intrinsics for the B 7000/B 6000 series of computers.

RELATED INFORMATION MANUALS

The compiler languages in which the mathematical intrinsics are used are described in the following manuals:

B 7000/B 6000 ALGOL Reference Manual, form 5001639;
B 7000/B 6000 Series FORTRAN Reference Manual, form 5001506;
B 7000/B 6000 Series COBOL Reference Manual, form 5001464; and
B 7000/B 6000 Series BASIC Reference Manual, form 5001407.

RELATED PUBLICATIONS

The following publications provide information on mathematical intrinsics:

IBM System/360 FORTRAN IV Library Subprograms C28-6596.

Computer Approximations, John Wiley & Sons Inc., New York, 1968 (Edited by the Society of Industrial and Applied Mathematics).

The Art of Computer Programming, Knuth, Vol. 2 (Chapter 3 is on the generation of pseudorandom numbers).

CONTENTS AND ORGANIZATION OF THIS DOCUMENT

This document describes the mathematical algorithms used in the software system. These algorithms are part of the Master Control Program (MCP), but the compilers in which intrinsics are used may not refer to them by names given in this document. For example, in FORTRAN, the function CDABS actually refers to the intrinsic CABS described herein. For information about the use of these intrinsics by the various compilers, refer to the respective compiler and language documents. In general, the names given to the intrinsics in this document are those commonly accepted for mathematical functions.

LISTING OF CONSTANTS

Certain specialized constants, such as pi and e, are used throughout the algorithms. A list of these constants is given in Appendix 9A. These values are stated in both single and double precision, where necessary; the appropriate value should be chosen depending on whether the algorithm under consideration is single or double precision.

EXPONENTIATION

Several procedures are used in exponentiation and are called implicitly by the compilers. These procedures, because of their similarity, are listed singularly under the heading EXPONENT in each section.

DISCUSSION OF INTRINSIC FUNCTIONS

A brief description of each intrinsic function is given. In each case, this description is followed by the algorithm used in computing the function. Additionally, notes are sometimes provided regarding the derivation of the algorithm.

GROUPING OF INTRINSIC FUNCTIONS

The descriptions of the intrinsic functions are grouped into three sections, as follows:

Section 2: Single-Precision Intrinsics
Section 3: Double-Precision Intrinsics
Section 4: Complex Intrinsics

Within each section, the intrinsics are arranged in alphabetical order.

2. SINGLE-PRECISION INTRINSICS

ALGAMA

The ALGAMA function is the natural logarithm of the GAMMA function and is defined for positive real numbers. The algorithm used varies depending on the value of x.

```
For x < 3.28
ALGAMA(x) = LN (GAMMA(x))
```

For x >= 3.28, the calculation is more direct, relying on Stirling's approximation:

```
GAMMA(x)=(e**(-x)*x**(x-1/2)*SQRT(2*pi))g(x)
```

where the function g(x) is an error polynomial.

ARCOS

ARCOS is the inverse cosine function. A number between -1 and 1 is used, and the angle returned has that cosine in radians. The angle is chosen between 0 and +pi.

The arccosine is calculated entirely using the arcsine intrinsic and the identities of trigonometry, as follows:

```
ARCOS(x)= pi/2-ARSIN(x).
```

The value of pi/2 is given in Appendix 9A.

ARCTAN

ARCTAN is the arctangent function of mathematics. A number is used that returns the angle with that tangent in radians. The angle is chosen between -pi/2 and +pi/2. The arctangent is also called the inverse tangent.

ARSIN

ARSIN is the arcsine function of mathematics. A number between -1 and 1 is used, and the angle returned has that sine in radians. The angle is chosen between -pi/2 and +pi/2. Arcsine is also called the inverse sine.

The arcsine is calculated for x in the range 0 to .5 only. If x is outside this range, x is first reduced to being in the range, as follows:

If x < 0, the relationship ARSIN(x) = -ARSIN(-x) is used.

Then, if x >.5 = SIN(30 degrees) = SIN(pi/6), x is further reduced to be in the range by the identity

```
ARSIN(x)= pi/2-2(ARSIN(SQRT(1-x)/2))
```

The value of pi/2 is listed in Appendix 9A.

ATAN2

ATAN2 is the arctangent of the quotient of two numbers but is adapted to fall in the range of -pi to +pi by choosing it in a quadrant determined by the signs of X and Y, the two values it is given. In effect, this function is used in complex arithmetic as follows: given a complex number x+iy, ATAN2 (x,y) returns the argument of that number between -pi and pi.

ATAN2 is defined for all real x and y values except when x=y=0 and is calculated from the function ARCTAN, as follows:

If y = 0, then ATAN2(x,0) = sign(x)*pi/2.

If y < 0, then ATAN2(x,y) = ARCTAN(x/y) + sign(x)*pi.

If y > 0, then ATAN2(x,y) = ARCTAN(x/y).

In these cases, Sign(x) is a function that has the value +1 if x >= 0 and the value -1 otherwise.

COS

The cosine of a real number is computed by the use of the algorithm for sine and the identity

COS(x) = SIN(x + pi/2).

COSH

Depending on the value of the argument, the hyperbolic cosine of a real number is computed either directly from the definition by the use of the intrinsic EXP or by approximation.

COTAN

The trigonometric cotangent accepts a number, expressed in radians, and returns its cotangent.

The method used for calculating the cotangent is identical with the method for calculating the tangent except that since cotangent is the reciprocal of the tangent, the final calculation for the function is obtained by use of the relationships in the following table:

OCTANT	COTAN(x)
0	T/R
1	R/T
2	(-R)/T
3	(-T)/R

ERF

The error function (ERF) is used in calculating probability. The function accepts any number and returns a value between -1 and 1. The error function is defined as follows:

ERF(x)=2/SQRT(pi)*(INTEGRAL(e**((-t)**2)dt)from 0 to x).

This function is slightly different from the "normal" probability curve - Gauss's probability integral, which is

phi(x) = 1/SQRT(2*pi)*(INTEGRAL(e**(((-t)**2)/2)dt) from 0 to x).

The relationship between them is that

ERF(x)=2*phi(SQRT(2)*x).

ERFC

ERFC is the complement of the error function.

ERFC=1-ERF

EXP

The exponential function (EXP) raises an argument x to the base of e=2.71828... (see Appendix 9A). Thus, EXP(x) = e**x.

EXPONENT

Single-precision exponentiation is performed by the intrinsic **RTOR** (real-to-real).

GAMMA

The GAMMA function is defined for positive numbers by the integral

GAMMA(x)=INTEGRAL((t**(x-1)*e**(-t))dt) from 0 to infinity.

In addition, this integral can be extended analytically onto the complex plane and yields a function that exists at all points except the negative integers and 0; it is real for all other negative numbers. Therefore, the intrinsic GAMMA(x) accepts any number except 0 or a negative integer as an argument.

LN

The LN function is the natural logarithm of a positive real argument. This function results in a positive number if given an argument greater than one and results in a nonpositive number otherwise.

LOG10

The common logarithm (log to the base 10) is computed by using the intrinsic LN (for the natural logarithm) by use of the identity

LOG10(x)=LN(x)*LOG10(e).

The value of the common logarithm of e is listed in Appendix 9A.

RANDOM

The intrinsic RANDOM generates a pseudorandom real number x in the range 0 <= x < 1. The number is generated by the mixed congruential method, which is designed to give a uniform distribution. RANDOM is the starting point for procedures that generate a pseudorandom number satisfying a given distribution function. RANDOM takes one parameter, an integer called by name, and returns a real number between 0 and 1. RANDOM is the only intrinsic described herein that takes a call-by-name input parameter. That is, the parameter is both used by RANDOM and changed by it to the value that is normally to be used for input the next time the intrinsic is called. Therefore, the parameter for RANDOM is generally given an initial value that is thereafter changed to give succeeding values by the procedure itself.

Starting values to obtain a good sequence of pseudorandom numbers can generally be obtained by picking odd numbers close in value to 2**19, 2**20, or 2**21.

The procedure for RANDOM generates integers in the range of 0 to 2**(39-1) and then returns those integers divided by 2**39. Calling the value given to RANDOM as an integer variable N, it is changed as follows:

```
ABS(N):=(A * ABS(N) + 116177073375) MOD (2**39),
```

where A is a constant dependent on the sign of N (which is never changed) and the operator ":=" is the replacement operator.

The value of A is given as follows:

```
For nonnegative N: A=152587890725
For negative N:    A=277626315293
```

These values allow two different pseudorandom sequences depending on whether the starting value was positive or negative.

Then RANDOM(N) = ABS(N)/(2**39).

SIN

The trigonometric sine of an angle, expressed in radians, accepts a number and returns a value between -1 and +1.

```
SIN(x) = SIN(pi-x)
```

SINH

The hyperbolic sine (SINH) of a real number is computed either directly from the definition by the use of the intrinsic EXP or by an approximation, depending on the value of the argument.

SQRT

The square root (SQRT) of a nonnegative number results in a nonnegative number. The algorithm for square root is essentially the traditional Newton-Raphson method; however, an initial estimate is first derived.

TAN

The trigonometric tangent (TAN) of a number, expressed in radians, returns a positive or negative number depending on the argument. To compute the tangent of an angle, the angle is reduced if it is outside the range 0 to pi.

TANH

The hyperbolic tangent (TANH) of a real number is computed either directly from the definition by the use of the intrinsic EXP or by approximation, depending on the value of the argument.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

3. DOUBLE-PRECISION INTRINSICS

Many double precision intrinsics are calculated in the same way as the equivalent single precision intrinsics with all references to single precision intrinsics changed to references to double precision ones or with all references to the arithmetic operations assumed to be to double precision operations.

DARCOS

DARCOS is the inverse cosine function that accepts a double precision number between -1 and 1 and returns the angle which has that cosine in radians. The angle is chosen between 0 and +pi.

The double precision arccosine is calculated using the arcsine intrinsic and the identities of trigonometry as follows:

ARCOS(x) = pi/2-ARSIN(x).

The value of pi/2 is given in Appendix 9A.

DARSIN

DARSIN is the double precision inverse sine intrinsic that accepts a number between -1 and 1 and returns the angle which has that sine in radians. The angle is chosen between -pi/2 and +pi/2.

DATAN

The double precision arctangent takes a real number argument in radians. The argument, x, is reduced to the range $0<x<1$, where 1=tan(45 degrees)=tan(pi/4), as in the calculation of ARCTAN.

DATAN2

DATAN2 is calculated from DATAN in the same way as ATAN2 is calculated from ARCTAN.

DCOS

DCOS is computed from DSIN in the same way as COS is calculated from SIN:

DCOS(x) = DSIN(x + pi/2)

DCOSH

DCOSH is the double precision hyperbolic cosine of a real number and is computed either directly from the definition by the use of the intrinsic DEXP or by approximation, the same as COSH.

DERF

DERF is the double precision error function used in calculating probability. The function accepts any number and returns a value between -1 and 1. DERF is defined the same as ERF.

DERF(x)=2/SQRT(pi)*(INTEGRAL(e**((-t)**2)dt) from 0 to x)

DERFC

DERFC is the complement of the double precision error function.

DERFC=1-DERF

DEXP

The double precision exponential function (DEXP) is calculated similarly to the single precision exponential function. It is converted to an exponent to the base 2 and written as 2**I*2**F, where I is the integer portion and F the fractional portion of the exponent.

DGAMMA

DGAMMA is the double precision equivalent of the GAMMA function previously described as:

GAMMA(x)=INTEGRAL((t**(x-1))(e**(-t))dt)from 0 to infinity.

In addition, this integral can be extended analytically onto the complex plane and yields a function that exists at all points except the negative integers and 0; it is real for all other negative numbers. Therefore, the intrinsic DGAMMA(x) accepts any number except 0 or a negative integer as an argument.

DLGAMMA

DLGAMMA returns the double precision natural log of the GAMMA function.

DLOG

The double precision natural logarithm (DLOG) is calculated in a similar way to LN in single precision.

DLOG10

The common logarithm is computed in double precision by the use of the same identity as in single precision.

DLOG10(x) = DLOG(x)*DLOG10(e)

The value of DLOG10(e) is given in Appendix 9A.

DSIN

The sine of a double precision argument is reduced to 0<=x<pi/2 by the method used in calculating the single precision sine.

DSINH

DSINH is the double precision hyperbolic sine of a real number and is computed either directly from the definition by the use of the intrinsic DEXP or by approximation, depending on the value of the argument (the same as SINH).

DSQRT

The square root of a double precision argument is computed in two steps: (1) the single precision square root of the most significant half of the argument is computed using the algorithm for finding the single precision square root; and (2) one additional Newton-Raphson iteration, using the original double precision argument, double precision arithmetic, and the single precision square root is computed. The exponent is shifted if necessary.

DTAN

DTAN is the double precision trigonometric tangent of a number, expressed in radians. It returns a positive or negative number depending on the argument. The calculation of the DTAN is the same as that of the TAN.

DTANH

The double precision hyperbolic tangent (DTANH) of a real number is computed either directly from the definition by the use of the intrinsic DEXP or by approximation, depending on the value of the argument (in the same manner as TANH).

EXPONENT -- DOUBLE PRECISION

Double precision exponentiation is performed similarly to single precision exponentiation by the use of two routines: RTOD (real-to-double) and DTOD (double-to-double). In either case, the result is double precision.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

4. COMPLEX INTRINSICS

All of the complex intrinsics are derived by the use of the real intrinsics. In this description, several methods are used to write a complex number. When a complex number is to be described as a simple variable, the letter z is used. Several equivalent ways of writing z exist; for example,

z = x + iy, where x and y are real numbers and i = SQRT(-1).

z=re**(i*phi) where r and phi are the absolute value and the displacement, respectively. They are related to x and y by the relationships

x = r COS(phi) and
y = r SIN(phi)
(also, r**2 = x**2 + y**2 and TAN(phi) = x/y).

These identities also indicate DeMoivre's formula:

e**(i*phi) = COS(phi) + iSIN(phi).

These basic relationships are used to determine most of the complex algorithms.

CABS

The absolute value of a complex number z is defined to be ABS(r) (see the definitions at the beginning of this section).

Therefore,

CABS(x + iy) = SQRT(x**2 + y**2).

If ABS(x) >= ABS(y), the right side is evaluated as

SQRT(1 + (y/x)**2)*ABS(x).

Otherwise, the right side is evaluated as

SQRT(1 + (x/y)**2)*ABS(y).

CCOS

The cosine of a complex number z is calculated by the use of the identity for COS (a+b) on the number x+iy. Then the identities COS(iy) = COSH(y) and SIN(iy)=iSINH(y) are applied. These relationships are derived by the use of the definitions at the beginning of this section. When the definitions of the hyperbolic sine and cosine are substituted in the equation, the algorithm becomes

CCOS(x+iy) = (+or-)1/2 SQRT(1-SIN**2(x)) (e**y + e**(-y)) - i/2*
SIN (x)*(e**y - e**(-y)).

The value of x is taken modulo 2pi before the sine intrinsic is called. The negative sign is taken on the square root if the original x was in the second or third quadrants.

CEXP

By the use of DeMoivre's relationship (see the introduction to this section) and the basic identity that COS**2(x) = 1 - SIN**2(x), the value of the exponential can be calculated as follows:

```
e**(x+iy)=e**x ((+or-)SQRT(1-SIN**2(y))+i SIN(y)).
```

The value of y is taken modulo 2(pi) before the sine intrinsic is called. The negative sign on the square root is chosen if the original y is in the second or third quadrants.

CLOG

The complex natural logarithm (CLOG) of a number is calculated by the use of DeMoivre's relationship and the relationships between x, y, r, and phi (see the introduction to this section). Since the complex logarithm is not a single-valued function, the value returned by CLOG is in the range from -pi to +pi.

The algorithm is basically

```
CLOG(x+iy) = LN(r)+i*phi,
```

where phi is chosen to fall in the principal range noted and both r and phi are as previously defined. The logarithm is computed as a real number. The value of phi is computed with the real intrinsic ATAN2, which is designed for use in this application. Then the algorithm becomes

```
CLOG(x+iy) = LN(SQRT(x**2+y**2))+ i ATAN2(y,x).
```

CSIN

The sine of a complex number z is calculated by the use of the identity for SIN(a+b) on the complex number x+iy. Then the identities COS(iy) = COSH(y) and SIN(iy) = SINH(y) are applied. These relationships are derived by use of the definitions at the beginning of this section. When the definitions of the hyperbolic sine and cosine are substituted in the equation, the algorithm becomes

```
CSIN(x+iy)=(+or-)1/2*SQRT(1-COS**2(x))*(e**y+e**(-y))
+i/2 COS(x)*(e**y-e**(-y)).
```

The value of x is taken modulo 2pi before calling the COS intrinsic. The negative sign is taken on the square root if the original x was in the third or fourth quadrants.

CSQRT

The complex square root of a number is calculated first by using DeMoivre's relationship and taking its square root.

CSQRT(z)=(r)**1/2(COS(phi/2) + i SIN(phi/2)).

Using the half-angle formulas for the cosine and sine and rearranging the above relationship, the following is derived:

CSQRT(z)=SQRT(r(1+COS(phi))/2)+(SQRT(r(1-COS(phi))/2)*i).

The identity x/r=COS(phi) and algebraic manipulation result in the algorithm that is used.

For x >= 0, let r = CABS(x+iy), then

CSQRT(x+iy)=SQRT((r+x)/2)+iy/(2 SQRT((r+x)/2)).

If x < 0, then the trigonometric functions in the polar form are complemented and the following algorithm results, where r = CABS (x+iy):

(SQRT(x+iy)=y/(2*SQRT((t+ABS(x))/2))+ i SIGN(y)*SQRT((r+ABS(x))/2).

SIGN(y) is a function that has the value +1 if y is nonnegative and the value -1 otherwise.

EXPONENT--COMPLEX

Exponentiation of a complex number is performed by two routines: CTOR, for complex numbers to a real power, and CTOD, for complex numbers to a double precision power. The only difference between the double precision power and the real power is that computations are performed by the use of the double precision intrinsics. Because the final result must be a complex number and no double precision complex exists, exponentiation to a double precision power may result in little increased accuracy at a high cost in time, depending on the particular case.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

APPENDIX 9A.

COMMON CONSTANTS

This appendix lists common constants used in computing the intrinsics. Real and, where necessary, double precision values are given for each of the constants. Since fewer double precision intrinsics than real intrinsics exist, some of the constants are unnecessary in the double precision cases.

Constant	Single-Precision Value	Double-Precision Value
pi	3.14159265359	3.1415926535897932384626
pi/2	1.57079632697	1.570796326794896619231 3
pi/6	.523598775598	.523598775598298873077 11
SQRT(3)	1.732050807570	1.732050807568877293527 5
LN(2)	.693147180560	.693147180559945309417 23
e	2.71828182846	2.718281828459045235360 3
LOG10(e)	.434294481903	.434294481903251827651 13
TAN(pi/24)	.267949192431	.267919243112270647253
LN(SQRT(2pi))	.918938533205	
LOG2(e)	1.4426950409	
pi/4	.785298163397	
3(pi)/4	2.35619449019	
SQRT(2)/2	.707106781187	
TAN(pi/40)	.0787017068246	
TAN(pi/20)	.158384440326	
TAN(3pi/40)	.240078759080	
TAN(pi/10)	.324919696234	

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

APPENDIX 9B.

STORAGE ESTIMATES AND PERMISSIBLE ARGUMENT RANGE

This appendix lists the storage estimates (in words placed in the stack) for the code of each of the intrinsics. Although many of the descriptions of intrinsics in this document indicate that one intrinsic may call another during its execution, these storage estimates include all such calls. Where those calls are not included, an asterisk (*) is placed next to the storage estimate.

In listing permissible argument ranges, several abbreviations are used. The word "All" signifies that all single precision numbers (or double precision numbers if the intrinsic is double precision) are permitted. "All" is often modified in some obvious manner. Where an intrinsic has more than one argument, the requirements for each are listed separated by commas. The notation (0,0) means that both the first and second arguments are zero.

Intrinsic Para. No.	Intrinsic Name	Storage Estimate (In Words)	Permissible Argument Range
1	ALGAMA	36*	All positive
2	ARCOS	9*	(-1,1)
3	ARCTAN	47	All
4	ARSIN	51*	(-1,1)
5	ATAN2	70	All except (0,0)
6	COS	39	All
7	COSH	27*	All
8	COTAN	58	All
9	ERF	84	All
10	EXP	44	All
11	EXPONENT(RTOR)	30*	All for exponent, all for base except negative numbers to non-integral exponent
12	GAMMA	64*	All except negative integers and 0
13	LN	49	All positive
14	LOG10	51	All positive
15	RANDOM	16	ABS < 2**39
16	SIN	37	All
17	SINH	25*	All
18	SQRT	31	All nonnegative
19	TAN	58	All
20	TANH	26*	All
21	DATAN	103	All
22	DATAN2	29*	All except (0,0)
23	DCOS	100	All
24	DEXP	103	All
25	DLOG	96	All positive
26	DLOG10	100	All positive
27	DSIN	96	All
28	DSQRT	43	All nonnegative

MATHEMATICAL INTRINSICS

29	EXPONENT		For both;
	(RTOD)	30*	All for exponent
	(DTOD)	30*	All nonnegative for base
30	CABS	15*	All
31	CCOS	20*	All
32	CEXP	16*	All
33	CLOG	7*	All
34	CSIN	21*	All
35	CSQRT	20*	All
36	EXPONENT		For both;
	(CTOR)	55*	All for base
	(CTOD)	69*	All real or double precision for exponent

---

*Does not include storage space for intrinsics called by this intrinsic, if necessary.

PATCH

TABLE OF CONTENTS

1. INTRODUCTION

The patch merge program (SYSTEM/PATCH) is an ALGOL utility program used to merge one or more patch decks into a single patch deck (on disk or pack) which may be used as the input CARD file for an ALGOL, ESPOL, DCALGOL, COBOL, or FORTRAN compilation.

SYSTEM/PATCH merges all input patch records by sequence number. Only numeric or blank sequence numbers are accepted. The program allows resequencing and patching into resequenced areas of a patch.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

2. FILES USED BY SYSTEM/PATCH

CARD	The input file containing patches to be merged by the program.
LINE	The output printer file.
NEWTAPE	The output disk file created by merging the PATCH file with the TAPE file.
PATCH	The output disk file containing the merged patches.
PATCHES	The output disk file containing the input specified by the $. OUT option.
TAPE	The symbolic disk file to which the patches and $ options of GO TO, SEQ, MERGE, and the $. options of INSERT, MOVE, COMPARE, LISTN, CYCLE, VERSION, and NEW are applied.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

3. DOLLAR ($) CARDS RECOGNIZED BY SYSTEM/PATCH

Seven categories of dollar cards are acceptable as input to SYSTEM/PATCH. These categories are distinguished by a unique character, or blank, immediately following the dollar sign. Each category is described as follows:

$Cards

The following compiler dollar option cards are recognized by SYSTEM/PATCH:

```
SEQ <sequence base> +/- <sequence increment>
VOID
VOIDT
MERGE
GO TO
BUMP TO
```

These option functions are performed by SYSTEM/PATCH and not by the compiler. For this reason, they are (except for MERGE) erased from the card they are on before that card is written to the patch file. SYSTEM/PATCH creates SET and POP VOIDT cards as needed to simulate the functions of these options.

All other options are passed to the host compiler via the PATCH file, but ignored by SYSTEM/PATCH. Such compiler options as AREACLASS, INSTALLATION, LEVEL, LIMIT, VERSION, INCLUDE, MAKEHOST, and CHECKPOINT are checked for format since their associated parameters could cause invalid actions if improperly specified. Unlike the compilers, if no option action is present, SET is assumed.

A dollar card is defined to be a card with a dollar sign in column 1 (or column 7 for COBOL). A dollar sign in any other column is not recognized. Normally, SYSTEM/PATCH protects a dollar card from being suppressed by a card with the same sequence number in a succeeding patch. This protection is removed for dollar cards which have nothing on them.

$& Cards

$& cards are $ cards that the user does not want SYSTEM/PATCH to handle. SYSTEM/PATCH replaces the & character by a blank and puts the card into the PATCH file.

Example

```
$& SET SEQ 90000 + 1000

becomes

$  SET SEQ 90000 + 1000
```

$# Cards

$# cards are patch delimiter cards. Each individual patch within the input deck must be immediately preceded by a card with a $# in columns 1 and 2 (columns 7 and 8 for COBOL; that is, when the COBOL control option is set). The remainder of the card may be used for comments. (See discussion of LABEL, MARK, and COUNT control options in later paragraphs.)

$: Cards

$: cards are comment cards. They are listed if LISTP is SET and written to the output PATCHES file if OUT is SET; otherwise, they are ignored. They can occur anywhere in the patch input; no limit exists on the number that can occur.

$- Cards

$- cards are used to patch a patch. A $- card is treated as a regular card, in that it must have a sequence number and may delete a card in a previous patch at that sequence number; however, it is not included in the PATCH file. The effect is to let the original source filter through with the original patch number (if any) without changing an established patch or repunching the source and losing the patch number.

$* CARDS

$* cards contain WFL commands (without comments) which SYSTEM/PATCH puts into an array and performs an ALGOL ZIP WITH ARRAY. SYSTEM/PATCH modifies the $* cards by putting a semicolon at the end of each statement and precedes each statement with a question mark. By placing a hyphen in column 80 of a $* card, the user can suppress the insertion feature described above (for that card).

$. Cards

$. cards are control cards to SYSTEM/PATCH which are similar to compiler $ cards. They are used to control SYSTEM/PATCH and are not included in the PATCH file. Many of these options may be SET, RESET, or POPped in a manner similar to the compiler options. If no action is specified, SET is assumed. The text field of a $.CARD consists of columns 3 thru 80 (9 thru 80 if $.COBOL is set). Parsing of this text field is terminated by a percent character. Unlike compiler options, no action is taken on options not specifically mentioned. Listed below are the $. options and a description of each.

$. BCL

When the CARD file has some input that is BCL punched, BCL should be SET before such input and RESET afterwards. SYSTEM/PATCH does a software translation of the specified input. This is not necessary if the intmode of the card file is BCL. The BCL option may be SET, RESET, or POPped. The default value is RESET.

$. BRIEF

This option is used with the COMPARE option to suppress printing of more than six consecutive voided lines. The first card voided, the last card voided, and the number of cards voided is printed instead. BRIEF may be SET, RESET, or POPped. The default value is RESET.

$. COBOL

This option tells SYSTEM/PATCH to expect input in COBOL format: Sequence numbers in columns 1 through 6, dollar signs in column 7. After this option is SET, all special $ cards recognized by SYSTEM/PATCH must be in column 7, but the card actually setting this option must have a $. starting in column 1. This option may be SET, RESET, or POPped. The default is RESET.

$. COMPARE

This option causes SYSTEM/PATCH to compare the PATCH file with the TAPE file, listing all patch cards and cards affected in the TAPE file. COMPARE may be SET, RESET, or POPped. The default value is RESET.

$. COMPILE

The COMPILE option causes SYSTEM/PATCH to ZIP the compilation of the TAPE file with the PATCH file if no fatal errors are discovered. The CARD and TAPE files are label equated automatically by SYSTEM/PATCH. Other information for the compile must be passed by $* cards supplied by the user.

Example

```
$. SET COMPILE
$* COMPILE A/B WITH ALGOL LIBRARY
$* ALGOL FILE NEWTAPE( TITLE = S/A/B)
```

COMPILE may be SET, RESET, or POPped. The default value is RESET. COMPILE and EXECUTE may not be SET at the same time.

$. CONFLICT

This option controls printing of patch conflicts (cards deleted in previous patches by cards in later patches). When SET, these conflicts are listed in the LISTP output section. CONFLICT may be SET, RESET, or POPped. The default value is SET.

$. COUNT

Syntax

$. COUNT <number>

Semantics

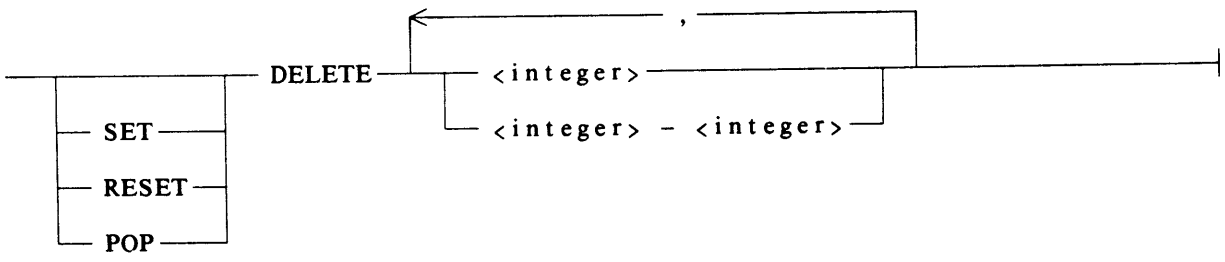
If the action is SET or no action is specified, the COUNT option must be followed by an unsigned integer number. If the action is RESET or POP, the COUNT option must not be followed by this number. When SET, SYSTEM/PATCH gets the number of cards to be found in each patch from the $# card for that patch. It checks the number found against the number specified and issues an error if the numbers differ. The specified card count on the $# card must begin in the column specified by the number in the $. COUNT command. This allows flexibility in that different areas on the $# card may be used to specify the card count for different patches. $ cards are counted, non $ cards are counted, and $. cards with MOVE or INSERT commands are counted.

$. CYCLE

Refer to the $. VERSION option.

$. DELETE

Syntax



Semantics

When SET, SYSTEM/PATCH deletes the patches specified in the number list. Patches already processed are not affected. Each patch may have its DELETE option SET, RESET, or POPped as desired. A deleted patch is listed if LISTP is SET but is otherwise ignored. DELETED patches are not included in the PATCHES file.

$. DISK

Syntax

$. DISK <file title>

Semantics

This option tells SYSTEM/PATCH to get input from the file specified. Input from this file is expected to be in the same format as all other input to SYSTEM/PATCH with the following exceptions:

1. The input from the specified file may not contain $. DISK, $. DISK $, $. PATCHDECK , or $. FILE commands.

2. If the file specified has a maxrecsize 15 (11 if intmode is BCL), SYSTEM/PATCH does not change the MARK numbers even if MARK is SET. This allows MARK numbers already present to be preserved.

Example

```
$. DISK X/Y/Z
```

Label equation is allowed for file title specification of one node for $.FILE, $.DISK, $.PATCHDECK, and $.DISK$ commands.

Other $. commands may appear on the same $. card after $.FILE, $.DISK, $.PATCHDECK, and $.DISK$.

Example

```
? BEGIN JOB PATCHER(STRING PATCHFILE);
RUN SYSTEM/PATCH;
FILE TAPE(TITLE = SYMBOL/SOURCE ON PACK01);
FILE INPUT(TITLE = #PATCHFILE);
DATA CARD
$#PATCH SEPARATOR CARD 1
$ SET LIST MERGE NEW
$#PATCH SEPARATOR CARD 2
$.FILE INPUT COMPARE
? END JOB
```

CANDE, SPO, or WFL input would be:

```
START PATCH/RUN (" PATCH/SOURCE/23 ON PACK02 ")
```

The result of this START command is a SYSTEM/PATCH run in which the file PATCH/SOURCE/23 ON PACK02 is used, and a compare listing is generated.

$. DISK $

Refer to the $. FILE option.

$. DUMP

If the first fatal error occurs in patch N (N > 1) and DUMP is SET, SYSTEM/PATCH merges the first N-1 patches and locks them on a disk file. If the PATCH file had the title X/Y/Z, then this file has the title DUMP/X/Y/Z. The $. DISK option can then be used to restart the merge without rereading the first N-1 patches. DUMP may be SET, RESET, or POPped. The default value is RESET.

$. EOF

This option tells SYSTEM/PATCH that this is the end of all input. $. EOF may occur in any input file. Option actions have no effect on it.

$. ERRLIST

This option is only for execution from a remote terminal. If ERRLIST is SET, all errors and warnings are displayed at the the terminal. If ERRLIST is RESET, this listing is suppressed. ERRLIST may be SET, RESET, or POPped. The default value is SET. If execution is not from a remote terminal, changing the value of ERRLIST has no effect.

$. EXECUTE

The EXECUTE option tells SYSTEM/PATCH to ZIP a specified program if no fatal errors are discovered. This option uses the $* cards in a similar manner to the COMPILE option, except that no label equation occurs. The terminal END JOB control statement is supplied by SYSTEM/PATCH. EXECUTE may be SET, RESET, or POPped. The default value is RESET. EXECUTE, and COMPILE may not be SET at the same time.

$. FILE, $. DISK $, and $. PATCHDECK

Syntax

$. FILE <file title>
$. DISK $ <file title>
$. PATCHDECK <file title>

Semantics

These commands are synonymous. They are extensions of the CARD file. When one of these commands is encountered, SYSTEM/PATCH reads from the specified file until it reaches end-of-file or until another $. FILE, DISK $, PATCHDECK, or DISK command is encountered. The differences between these commands and the $. DISK command is that input from these commands is marked if MARK is SET. Each may be nested within the others up to 10 levels. Each may contain $. DISK commands. If LISTP is SET, the file title of the disk file from which a record is read is printed to the right of the sequence number in the printer listing. Titles of files specified by $.DISK $, $.PATCHDECK, and $.FILE commands can be label equated. Refer to $.DISK for this feature.

Examples

```
$. FILE MY/FILE  ON MYPACK
$. PATCHDECK A/B
$. DISK $ MY/OTHER/FILE ON MYPACK
```

$. GUARD

Syntax

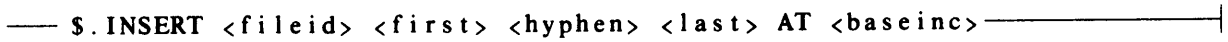
$. GUARD <m> - <n> <comment>

Semantics

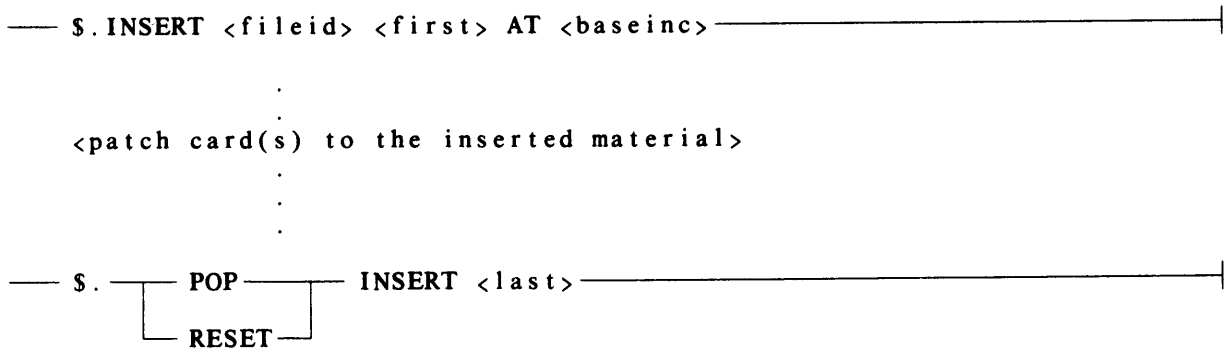
The GUARD option specifies that all patch cards within the specified sequence range (<m>-<n>) are to be flagged in a special report in the printer output with the specified <comment>. The <comment> may be any character string. The GUARD option must be the last control option specified on the $. card. No more than 100 areas may be guarded in this manner. If a fatal error occurs, no GUARD output is generated.

$. INSERT

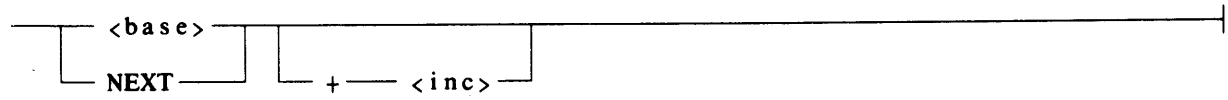
Syntax 1



Syntax 2

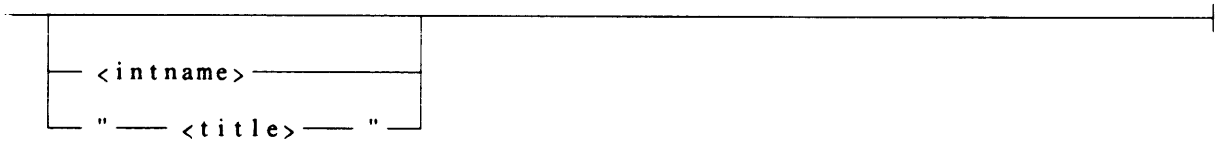


<baseinc>



PATCH

<fileid>



NOTE

<hyphen> indicates the character "-".

Semantics

The INSERT command serves two functions. Syntax 1 is used to insert a copy of a portion of the virtual TAPE file (the TAPE file plus previous patches) at the base specified and to insert a portion of an external file (indicated by <fileid>) at the base specified. Syntax 2 allows text that is being INSERTed to be patched.

<baseinc> must be either a number or the mnemonic NEXT. <baseinc> specifies the starting sequence number at which the INSERTed text is to be put. If no increment (+<inc>) is specified, then the last value of the sequence increment is used. Since the base and increment used for the INSERT command are the same as the base and increment used for handling the SEQ $ option and the MOVE command, their values may be changed during the INSERT by a $ integer card or $ +integer card via syntax 2. They are not reset to their default values until the next patch ($# card).

The NEXT version of specifying the base has two meanings. If the value of SEQ is SET, then NEXT simply means to use the present value of the sequence base. If the value of SEQ is RESET, then NEXT means to use the sequence number of the last card in this patch plus the value of the increment as the base.

NOTE

An INSERT may not be done while VOID is SET, MERGE is RESET, or while doing a MOVE. VOID may not be SET, MERGE may not be RESET, a MOVE may not be done, a $ GO TO may not occur, and SEQ may not be changed while INSERTing. $ cards may not occur in text INSERTed from an external file. If the INSERT is from an external file, then VOIDT may be SET when the INSERT begins but may not be changed during the INSERT. If the INSERT is not from an external file, then VOIDT may not be in a SET state at any time during the INSERT. INSERT commands may not be nested. The range to be INSERTed may not overlap the destination range if the INSERT is from the virtual TAPE file. The destination range may not overlap sequence numbers in the virtual TAPE file.

$. LABEL

Syntax

$. LABEL <number>

Semantics

If the action is SET or no action is specified, the LABEL option must be followed by an unsigned integer number. If the action is POP or RESET, the COUNT option must not be followed by this number. When SET, SYSTEM/PATCH gets the label to be used for a patch from the column on the $# card for that patch specified by the unsigned integer of the LABEL command. The label information is terminated by the first blank character. If COBOL is SET, this label information is right justified in column 80 for a maximum length of 8. If COBOL is not SET, the label information is prefaced by a percent character and right-justified in column 72. If a nonblank character is present in the destination field (of the card to be labeled), the label for that card is suppressed.

$. LIST

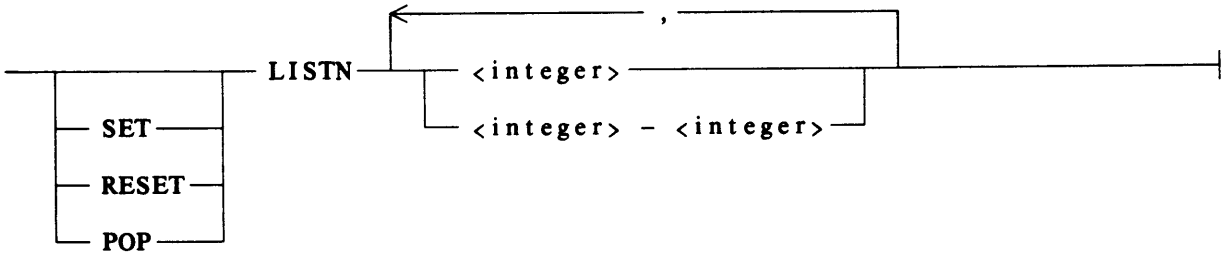
The LIST option tells SYSTEM/PATCH to list the created PATCH file (if no fatal errors occured) in the LINE file. LIST may be SET, RESET, or POPped. The default value is RESET.

$. LISTI

If LISTI is SET, SYSTEM/PATCH lists input inserted from external files, as specified by the INSERT option, in the LISTP section of the LINE file. LISTI may be SET, RESET, or POPped. The default value is SET.

$. LISTN

Syntax



Semantics

If the value of LISTN is SET, then ranges of the virtual NEWTAPE file (that is, the NEWTAPE file SYSTEM/PATCH creates or the NEWTAPE file SYSTEM/PATCH would create if NEW had been SET) as specified by the <number list> are listed in the LINE file. If <number list> is <empty>, then the complete virtual NEWTAPE file is listed.

If the value of LISTN is RESET, then ranges of the virtual NEWTAPE file as specified by the <number list> are not listed in the LINE file. If <number list> is <empty>, then none of the virtual NEWTAPE file is listed. This allows the user to RESET all or parts of ranges that were previously SET.

LISTN may be SET, RESET, or POPped as desired. The default value is RESET with an <empty> number list (no portion of the virtual NEWTAPE file is listed).

$. LISTP

If LISTP is SET, SYSTEM/PATCH lists the input to the LINE file. All $ cards, $# cards, $: cards, $& cards, $* cards, $- cards, and $. cards are listed in this section as they are found. LISTP may be SET, RESET, or POPped. The default value is SET.

$. MARK

The MARK option is for use with ALGOL, ESPOL, and DCALGOL symbolics. When SET, SYSTEM/PATCH places the mark level information in columns 81 thru 90 (81 thru 88 for ESPOL symbolic files) of the merged patch. This information is taken as the first item immediately after the first nonblank ("noise") character string on each $# card.

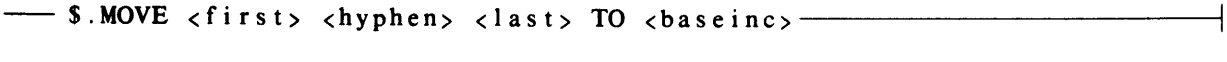
Example

```
$. MARK LABEL 5  COUNT 3
$#12XYZ  27.380.056
```

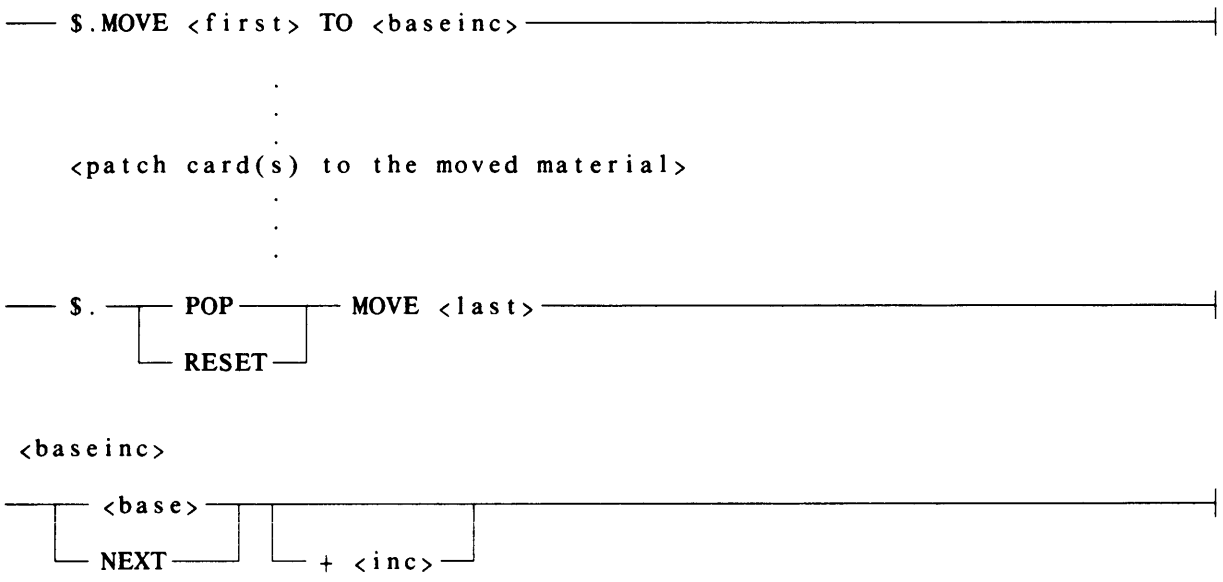
In this example, all cards from this patch (except cards read in from a file specified by a $ DISK command) contain 27.380.056 in columns 81 thru 90 in the merged patch. These cards are also labeled %XYZ in columns 69 thru 72 (see $.LABEL). SYSTEM/PATCH also checks to see that this patch has exactly 12 cards in it (see $.COUNT).

$. MOVE

Syntax #1



Syntax 2



NOTE

<hyphen> indicates the character "-".

Semantics

MOVE commands move portions of the virtual TAPE file (that is, the TAPE file plus previous patches) to a range beginning at the base specified. This is done by putting SET and POP VOIDTs around the range to be moved and creating a copy of the moved text at the new range. Syntax 2 allows text that is being MOVEd to be patched.

<baseinc> must be either a number or the mnemonic NEXT. <baseinc> specifies the starting sequence number at which the MOVEd text is to be put. If no increment is specified, then the last value of the sequence increment is used. Since the base and increment used for the MOVE command are the same as the base and increment used for handling the SEQ $ option and the INSERT command, their values may be changed during the MOVE by a $ integer card or $ +integer card in syntax 2. They are not reset to their default values until the next patch ($# card).

The NEXT version of specifying the base has two meanings. If the value of SEQ is SET, then NEXT simply means to use the present value of the sequence base. If the value of SEQ is not SET, then NEXT means to use the sequence number of the last card in this patch plus the value of the increment as the base.

NOTE

A MOVE may not be done while VOIDT or VOID is SET, MERGE is RESET, or while doing an INSERT; nor may SEQ be changed or a $ GO TO occur while doing a MOVE. MOVE commands may not be nested. The range to be MOVEd may not overlap the

destination range and the destination range may not overlap sequence numbers in the virtual TAPE file.

$. NEW

If NEW is SET and SYSTEM/PATCH finds no fatal errors, the PATCH file is merged with the TAPE file to create the NEWTAPE file. The NEWTAPE file contains no $ cards (even $ cards passed through as $& cards are not included). The blocking factors of the NEWTAPE file are those that a compiler created NEWTAPE would have. NEW may be SET, RESET, or POPped as desired. The default value is RESET.

$. OUT

When OUT is SET, $ cards, $* cards, $# cards, $: cards, $- cards, $. cards with MOVE or INSERT options, and regular patch cards are written to the output disk file PATCHES. This file is locked after all input has been processed. OUT may be SET, RESET, or POPped to allow the user to select only specific portions of the input. The default value for OUT is RESET.

$. PATCHDECK

Refer to the $. FILE option.

$. SINGLE

When SET, SYSTEM/PATCH single spaces the output to LINE. When RESET this output is double spaced. SINGLE may be SET, RESET, or POPped. The default value is SET unless SYSTEM/PATCH was compiled with the compiler user option DOUBLE SET, in which case the default value is RESET.

$. SQUASH

SQUASH may be SET, RESET, or POPped. When SET, each patch in the LISTP listing is separated by a line of equal signs. When RESET, each patch is listed beginning on a new page. The default value for SQUASH is SET.

$. TOTAL

Syntax

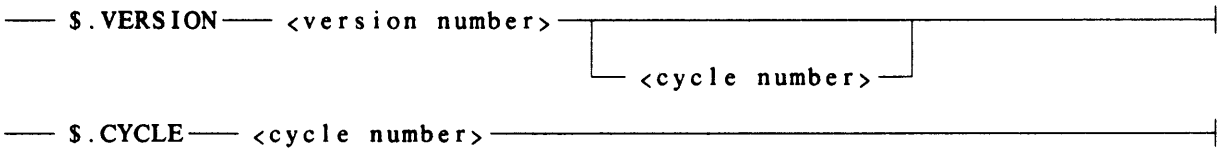
$. TOTAL <number>

Semantics

If the action is SET or no action is specified, the TOTAL option must be followed by an unsigned integer number. When all input has been processed and the value of TOTAL is SET, SYSTEM/PATCH checks the number of patches actually found against the number specified. If a discrepancy occurs, a fatal error is issued and the PATCH file is not locked. TOTAL may be SET, RESET, or POPped. The default value is RESET.

$. VERSION and $. CYCLE

Syntax



Semantics

When used, SYSTEM/PATCH concatenates the <version number>, the <cycle number>, and the <patch number> (from the $# card) and uses this as the mark number. If the TAPE file is an ESPOL symbolic or has an intmode of BCL, the periods (.) are not used as separators in this concatenation. All three numbers (<version number> , <cycle number> , and <patch number>) must be in the correct range. Both a <version number> and a <cycle number> must be specified. Each may be changed separately at any time. The <patch number> can only be changed at the beginning of each patch by the $# card.

4. EXAMPLE OF SYSTEM/PATCH INPUT

```
? RUN SYSTEM/PATCH
? FILE TAPE(TITLE=SYMBOL/PATCH ON SYSPACK)
? FILE PATCH(TITLE=SYSTEM/PATCH/NEWPATCH)
? FILE PATCHES(TITLE=SYSTEM/PATCH/NEWPATCHES)
? FILE INCL1 (TITLE=(USCODE) INCLUDE/FILE/1)
? FILE INCL2 (TITLE=(USCODE) INCLUDE/FILE/2)
? FILE NEWTAPE(TITLE=SYMBOL/NEW/PATCH ON SYSPACK)
? DATA
$. SET NEW COMPARE BRIEF EXECUTE LIST
$* RUN MY/PROGRAM ON MYPACK
$* FILE CARD(KIND=DISK,TITLE=KARD/FILE)
$#  D O L L A R    C A R D S
$ SET MERGE
$ SET NEW
$ SET LISTP
$ SET LINEINFO
$ SET SEQERR NEWSEQERR
$: THESE ARE SOME STANDARD $ CARDS FOR A COMPILE
$. MARK TOTAL 5  OUT DELETE 5
$#  N-O-I-S-E__W-O-R-D      27.099.001
$. PATCHDECK  MY/PATCH/FILE ON MYPACK
$# GARBAGE  27.099.002
                :
                :
          <patch cards>
                :
                :
$. VERSION 28.020
$# BUZZZZZ   001
$. MOVE 50000-52000 TO 600100+100
$. MOVE 800000  TO NEXT+20
                :
                :
   <patches to the moved material>
                :
                :
$. POP MOVE 8001000
$. INSERT INCL1 0-500 AT 900000 + 30
$. INSERT INCL2 6000-7000 AT NEXT
$. INSERT "MY/THIRD/INCL/FILE ON MYPACK"  61000  AT NEXT+300
                :
                :
   <patches to the inserted material>
                :
                :
$. RESET INSERT 75000
$. INSERT 2100-2500 AT NEXT  % THIS IS A COMMENT
$#  MORE_NOISE   2
$: THIS IS A COMMENT ABOUT THIS PATCH
$. FILE A/B
$. POP MARK RESET LISTP
$. DISK $    MY/OTHER/A/B ON MYPACK2
$. EOF
$: THIS CARD WILL NOT BE READ BY SYSTEM/PATCH
$: NOR THIS ONE
? END
```

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

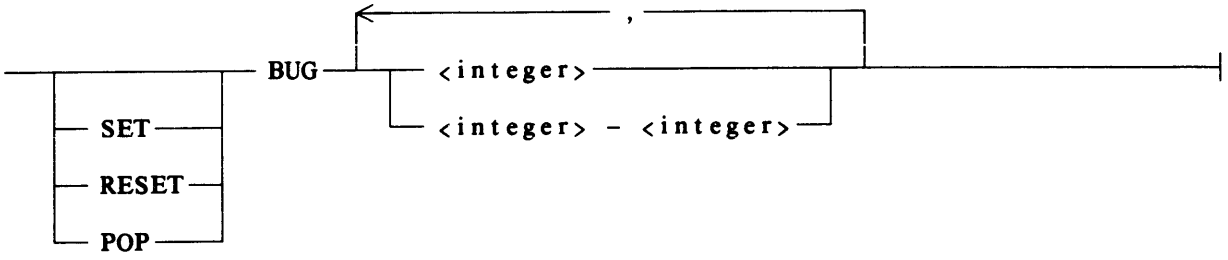
5. DEBUG COMPILE-TIME OPTION

A compile-time option DEBUG is available to facilitate the debugging and development of SYSTEM/PATCH. When SYSTEM/PATCH is compiled with a $ SET DEBUG card in the symbolic, the flow of control through many critical procedures and the values of many important variables may be traced at will.

$. Options Available With The DEBUG Option

$. BUG

Syntax



Semantics

The <number list> specifies which BUG options are to be SET, RESET or POPped. The specific action of each BUG option is subject to change and can be found in the BUG DIRECTORY near the beginning of the symbolic.

$. CANDE

CANDE may be SET, RESET or POPped. The default value is RESET. This option only concerns input from a remote terminal using CANDE, and changing its value has no effect when input is initiated from cards. When SET, the user may input text that has sequence numbers by typing the sequencing numbers first.

Example

```
$# PATCH 005
$. SET COMPARE
500$ SET VOIDT
01000$ POP VOIDT
```

Is equivalent to:

```
$# PATCH 005
$. SET COMPARE
$ SET VOIDT                                    00000500
$ POP VOIDT                                    00001000
```

$. DISCARD

This command causes SYSTEM/PATCH to close the LINE printer file with purge. The effect is to eliminate all printer output up to this point.

$. END

When SYSTEM/PATCH encounters a $. END option, it treats this as the end of all input to be merged for this particular set of patches. If no errors occur, SYSTEM/PATCH then creates the PATCH file and does the COMPARE and other optional output that may have been specified. It then starts reading from the primary input file (CARD file or remote file) and expects input for another SYSTEM/PATCH RUN. This capability allows multiple SYSTEM/PATCH runs in one run. Refer to $. EQUATE option.

$. EQUATE

Syntax

$. EQUATE TAPE = <file title>
$. EQUATE PATCH = <file title>
$. EQUATE PATCHES = <file title>
$. EQUATE NEWTAPE = <file title>

Semantics

The EQUATE option causes SYSTEM/PATCH to change the title of one of the four files specified above to the title given. This option must be the last option on the $. card. When used with the $. END option, multiple SYSTEM/PATCH runs may be done in one run against different pieces of software with different PATCH, PATCHES, and NEWTAPE files created. Separate printer files are created for each run.

$. PDUMP

This option may be SET, RESET, or POPped. The default value for PDUMP is RESET. When PDUMP is SET, SYSTEM/PATCH takes a PROGRAMDUMP on any error encountered.

TABLE OF CONTENTS

1. INTRODUCTION

This section describes the design and use of the MCP SORT facility, hereafter referred to as SORT. Sections 2 through 5 give a detailed design description of all phases of the SORT. Sections 6 through 9 provide a user's guide to writing efficient sorts in various modes and languages. The appendices supply reference information pertinent to sorting.

OVERALL SORT DESIGN

SORT is a procedure of the MCP. This procedure is designed to sort or merge a number of files into a single file of ordered records.

SORT workfiles are selected by the user. Workfiles may reside on disk or tape (or both) or in memory. the SORT procedure uses these workfiles to order records of a single input file.

The SORT can also merge a set of presorted files into a single ordered file.

Sorting is performed in two phases:

1. The sorting or stringing phase.

2. The merging phase.

When SORT is activated, it initially determines array sizes, number of tapes, buffer sizes, and blocking information from parameters provided by the user. SORT begins reading records from the input file and sorts them into groups, called "strings", on the sort workfiles.

After the last input record is read, the merging phase begins. The strings of sorted records are merged into larger strings until the result is one string containing the ordered input file. The ordered input file is written to the user's output file, and the SORT terminates.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES

2. DISK ONLY MODE

DISK SORTING

When this sorting mode is specified, all SORT workfiles are maintained on disk. The size of the disk workfile may be specified by a user-supplied disk estimate. If this estimate is not supplied, the default disk size for ALGOL and PL/I is 600,000 words and for COBOL, 900,000 words. Normally, SORT allocates 20 disk areas, with varying area sizes depending on the user's disk estimate.

Disk estimates must be large enough to accommodate the user's input file. The amount of disk required for merging depends on several factors. However, a safe disk estimate is 1.5 to 2 times the input file size.

Stringing Phase

Strings are written serially to the SORT workfile, titled DISKF, as they are formed during the stringing process. For each string, a disk control word is retained for use during the merge phase. A disk file, titled DISKC, is allotted for these control word records.

If disk space is exhausted during the stringing phase, the SORT is aborted.

Merging Phase

The disk merging phase begins after completion of the stringing phase and merges strings into longer strings on disk. As each new merged string is formed, a new control word is built. When the number of strings remaining to be merged is less than or equal to the number of strings that can be merged at one time, SORT writes the records to the user's file or output procedure.

During the merging phase, "wraparound" on the workfile is possible. Wraparound means merged records are written at the beginning of the workfile. Wraparound is possible because the strings occupying the space at the beginning of the the workfile have already been handled by the merge operation. This wraparound means that sorting can be done into the same disk file being used for input. This action is not recommended because a Halt/Load on a program fault leaves the disk file in an undefined state. A preferred method is to sort into a new file of the same name that will be locked at sort completion. In all cases, exhausting disk space (such as sorting a larger file into a crunched file) causes an abort.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES

3. TAPE ONLY MODE

TAPE SORTING

In a tape only sort, all SORT workfiles are maintained on magnetic tape. From 3 to 8 tapes may be used as workfiles; the user specifies the number of tapes to be used. The sorting method used for tape sorting is the polyphase merge/reverse technique.

Unlike a disk only sort where strings are written serially on the workfile as they are formed, special string distributions and string sequencing techniques are required for a tape sort. String distributions are based on a generalized Fibonacci number series. The string sequencing (ascending or descending) is specifically designed for reverse tape reads.

Stringing Phase

Initially, in the stringing phase, one work tape is designated as the first merge output tape and thus is not used during the stringing process. For example, in a three-tape sort, strings are written to only two tapes. Strings are then dispersed to the stringing tapes in the special pattern until the current level in the distribution is satisfied. The distribution is transferred to the next level and stringing continues. When the last input record is strung, the stringing phase is complete.

A special string sequencing pattern is required by SORT because of the reverse, tape-read technique used in the merging phase. The pattern is as follows:

1. Strings are written to an individual tape in alternating sequence (ascending, descending, ascending, and so on).

2. In an ascending sort, all tapes except the tape with the odd number of strings (that is, the last tape) begins with a descending string. The odd tape begins with an ascending string. In a descending sort, the sequence pattern is reversed.

The following example depicts the stringing phase of a five-tape ascending sort.

Example

	Tape 1	Tape 2	Tape 3	Tape 4
13 Strings	D	D	D	A
Distribution Is: 2,4,4,3	A	A	A	D
		D	D	A
		A	A	

D = descending string. A = ascending string.

During the stringing phase, SORT moves cyclically on the tapes - that is, from tape 1 to tape n - looking for a tape to string. If the distribution is satisfied on a given tape, SORT moves to the next tape. When the distribution is satisfied on all tapes, the Fibonacci distribution is transferred to the next level.

At the completion of the stringing phase, if the number of strings distributed is less than the desired Fibonacci distribution level, the tapes are padded with "dummy" strings to fill out the distribution. The dummy strings are recorded internally but are not physically written on the tapes.

Merging Phase

The merging phase of the tape sort uses the "polyphase merge/reverse tape read" technique. In the polyphase method, strings from working tapes are merged to a designated output tape until one of the tapes contains no more strings. This tape now becomes the output tape and thus, is the end of a merge pass or level. The string totals on the remaining tapes now correspond to the next lower level in the distribution table. The merging operation continues until one final string can be written to the user's file.

4. INTEGRATED TAPE DISK (ITD) SORTING

The ITD or disk/tape mode of sorting uses disk work files with tape backup. In ITD mode, the user supplies both a disk estimate and a number of work tapes.

SORT begins stringing records on disk; however, if disk is exhausted during stringing operations, a special merge is performed to tape, and SORT is not aborted. This merge creates strings on tape in the normal tape distribution, but the number of strings written on tape is less than that resulting from a tape only sort. Stringing then resumes normally on disk until disk is exhausted again. When the stringing phase is complete, a regular tape merge is performed.

During an ITD sort, label equations are not used for the internal tape files used during the stringing phase. If SORT requires a scratch tape, either a scratch tape may be mounted or the sort program may be terminated.

If disk is exhausted during the merging phase of an ITD sort, the strings are merged to tape, and the remaining merging operations are completed on tape.

Advantages of the ITD sort mode include the ability to circumvent a limited disk resource to sort large files and a reduction in tape merge time because of the use of disk to consolidate many short strings into a few longer strings.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES

5. MEMORY ONLY MODE

MEMORY SORTING

To initiate a memory only sort, the user does not specify tapes and specifically sets the disksize to zero. Failure to set disksize to zero results in the default value for disk being utilized, and a disk only sort occurs.

SORT does not open sort files and attempts to read the user input into memory. If sort memory is filled before the last user input record is read, the SORT terminates with SORT ERROR 3. If the specified amount of memory can contain all input records, it proceeds normally to produce user output.

When a memory only sort is desired, the correct specification for sort memory size is the number of records to be sorted multipled by the size of a record (in words). For COBOL sorts, the user determines the size from the SD description and rounds the size up to the number of words required to contain the record.

The core sort is of particular value when the number of records to be sorted is small.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

6. USE OF SORT IN COBOL LANGUAGE

COBOL SORTING

The SORT procedure is activated by executing the SORT or MERGE COBOL verbs. This section describes the required COBOL constructs for use of these verbs. (Refer to the B 7000/B 6000 Series COBOL Reference Manual, form 5001464, for additional information.)

SORT

COBOL Sort Syntactic Definition

```
IDENTIFICATION DIVISION.
.
.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
.
OBJECT-COMPUTER.    object computer entry
  [ . MEMORY SIZE  integer-1  { WORDS   } ]
                              { MODULES }
  [ . DISK SIZE    integer-2  { WORDS   } ]
                              { MODULES }
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT file-name-1 ASSIGN TO [integer-3] { SORT DISK                                                  }
                                           { SORT DISKPACK                                              }
                                           { SORT DISKPACKS                                             }
                                           { SORT-TAPE    [ AND integer-4 { TAPE     } ]               }
                                           { SORT-TAPES                   { TAPES    }                 }
                                           { MERGE                        { SORT-TAPE }                }
                                                                          { SORT-TAPES }               }
.
.
.
DATA DIVISION.
FILE SECTION.
SD file-name-1
  [ ; RECORD CONTAINS [ { data-name-1 } TO ] { data-name-2 } CHARACTERS ]
                      { integer-5   }      { integer-6   }
  [ ; DATA { RECORD IS  } record-name-1 [ . record-name-2 ] ... ]
           { RECORDS ARE }
01 record-name-1
.
.
.
PROCEDURE DIVISION.
.
.
.
  [ SORT ] file-name-1
  [ MERGE ]

            ON { ASCENDING  } KEY data-name-3 [ . data-name-4 ... ] ...
               { DESCENDING }
     [ . ON { ASCENDING  } KEY data-name-5 [ . data-name-6 ] ... ] ...
            { DESCENDING }

    { USING file-name-2                                                  }
    { INPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ]   }
                                        [ { THRU    }                ]   }
    { GIVING file-name-2                                                 }
    { OUTPUT PROCEDURE IS section-name-3 [ { THROUGH } section-name-4 ]  }
                                         [ { THRU    }                ]  }
      [ MEMORY SIZE { formula-1   } ]
                    { data-name-7 }
                    { literal-1   }
      [ DISK SIZE   { formula-2   } ]
                    { data-name-8 }
                    { literal-2   }
      [ RESTART IS  { formula-3   } ]
                    { data-name-9 }
                    { literal-3   }
.
.
.
section-name-1
.
.
.
  RELEASE record-name-1 [ FROM identifier-1 ]
.
.
.
section-name-3
.
.
.
  RETURN file-name-1 RECORD [ INTO identifier-2 ] ;
    AT END statement [ ELSE statement ]
.
.
.
```

SORT MODE

OBJECT-COMPUTER

If the MEMORY SIZE specification is omitted, SORT assumes a size of 12,000 words. If the DISK SIZE specification is omitted, SORT assumes a size of 900,000 words.

SELECT

If a file is assigned to integer-3 SORT-TAPES, then integer-3 must be between 3 and 8 inclusive. If a file is assigned to SORT DISK and integer-4 TAPE, TAPES, SORT-TAPE, or SORT-TAPES, then integer-4 must be between 3 and 8 inclusive. Integer-3 has no meaning when the file is assigned to MERGE or SORT DISK.

No additional SELECT statement options are permitted when a file is assigned to a sort/merge hardware device. File-name-1 must not be used in any other SELECT entries.

SD

File-name-1 must have been previously assigned to a sort/merge device in a SELECT statement and may not be used in any other DATA DIVISION entries.

The RECORD CONTAINS clause specifies the length of the logical records to be sorted. The data-name-1/integer-5 option is used to designate variable length records where the actual length of each record is specified by a decimal number contained in a four-character field at the beginning of the record. Data-name-1/integer-5 defines the minimum record length of the file; data-name-2/integer-6 defines the maximum length. Data-name-2/integer-6 may be used alone to define a file of fixed-length records; however, this use is not required, as the absence of a RECORD CONTAINS clause designates a file of fixed-length records. The record length is then determined by the first 01 level entry. Data-name-1 and data-name-2 may be used to specify the minimum and maximum record lengths at execution time and must contain the desired values prior to the execution of any SORT statement for file-name-1.

The DATA RECORD clause is used for documentation purposes only.

SORT

The ON ASCENDING/DESCENDING KEY clause defines the keys to be used in the sort. The order of precedence of the sort keys is determined by the order of appearance within the SORT statement (data-name-3 is the major sort key, and so forth).

Sort keys are subject to the following rules:

1. Each key must be defined within a record description of file-name-1.

2. Sort keys may not be variable length.

3. All signed, numeric, elementary items are compared algebraically; all negative values are considered lower than positive values. Anything else is compared as alphanumeric. DISPLAY keys are compared according to the EBCDIC collating sequence. DISPLAY-1 keys are translated to EBCDIC and compared according to the EBCDIC collating sequence.

4. KEY items cannot contain, nor can they be subordinate to, entries that contain the OCCURS clause.

5. All records presented to SORT must have a fixed length, and the key(s) must be in the same location within each record. An INPUT PROCEDURE may be used to modify any records that do not meet these standards.

6. The data names may be qualified.

The USING/INPUT PROCEDURE portion of the SORT statement specifies the input to the SORT. If file-name-2 is used, the minimum and maximum record sizes must be consistent with file-name-1.

The INPUT PROCEDURE defines sections of user code which are executed to select or alter records prior to the actual sorting. The RELEASE statement passes the current logical record to the SORT. At least one RELEASE statement is required in the INPUT PROCEDURE.

The GIVING/OUTPUT PROCEDURE portion of the SORT statement specifies the output from the SORT. If file-name-3 is used, a file is created containing the sorted records. The minimum and maximum record sizes must be consistent with file-name-1.

The OUTPUT PROCEDURE defines sections of user code which are entered when the sorting process is complete. The RETURN statement causes the sequential retrieval of one sorted record from the SORT. At least one RETURN statement is required in the OUTPUT PROCEDURE.

INPUT and OUTPUT procedures are subject to the following rules:

1. INPUT and OUTPUT PROCEDURES must not contain SORT statements.

2. The remainder of the PROCEDURE DIVISION must not contain statements that cause the transfer of control to, or to points within, the INPUT and OUTPUT PROCEDURES (for example, GO, PERFORM, or ALTER statements).

3. The INPUT and OUTPUT PROCEDURES may transfer control to points outside the range of the procedures; however, control must always return to the procedures.

4. A STOP RUN statement may not be executed in or by an INPUT or OUTPUT PROCEDURE. Such execution results in program failure.

5. Any attempt to execute a RELEASE or RETURN statement when the program is not under the control of a SORT statement results in program failure.

The logic chart on the following page shows the interaction of the SORT facility with a COBOL procedural sort.

SORT INPUT/OUTPUT PROCEDURE LOGIC FLOW

Sort 1. Begin SORT initialization phase.

Sort 2. Open SORT workfiles.

Sort 3. Go to beginning of the user's INPUT PROCEDURE.

IP1. Open input file.

IP2. Read input file record (at end, go to IP6.)

IP3. If record is to be used, place in record area of sort file; otherwise, go to IP2.

IP4. RELEASE sort file record (transfer to sort 4).

Sort 4. Place the RELEASED record in sorting process.

Sort 5. Execute internal sorting, creating strings on SORT workfiles.

Sort 6. Return to INPUT PROCEDURE at IP5.

IP5. Execute "using" logic then go to IP2.

IP6. Execute AT END logic including close of input file (transfer to sort 7).

Sort 7. Complete SORT stringing of all input records.

Sort 8. Begin merge phase of SORT.

Sort 9. Merge all strings on SORT workfiles until one string remains.

Sort 10. Go to beginning of OUTPUT PROCEDURE.

OP1. Open output file (transfer to sort 11).

Sort 11. Execute final internal merging operation.

Sort 12. Pass merged record to OUTPUT PROCEDURE at OP2.

OP2. RETURN sort file record to user record area (at end, go to OP4).

OP3. Execute user logic (transfer to sort 11).

OP4. Execute AT END including close of output file (transfer to sort 13).

Sort 13. Close all SORT workfiles.

Sort 14. Exit from SORT.

MERGE MODE

The rules for using the MERGE mode are the same as those for using the SORT mode, with the following exceptions:

1. The INPUT PROCEDURE option of the SORT statement may not be used.

2. At least two file names must appear in the USING portion of the SORT statement.

3. A maximum of eight files can be used as input to the merge.

4. All input files must be compatible with SD record descriptions as to key locations and record lengths.

An example of a COBOL disk sort including the actual input and output files is given on the following two pages.

COBOL SORT EXAMPLE

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  B-6700.
OBJECT-COMPUTER  B-6700
    DISK SIZE 20000 WORDS
    MEMORY SIZE 3000 WORDS.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT NEWTRANS ASSIGN TO CARD-READER.
    SELECT CREDITFILE ASSIGN TO 5000 DISK.
    SELECT DEBITFILE ASSIGN TO 5000 DISK.
    SELECT SORTFILE ASSIGN TO SORT DISK.
DATA DIVISION.
FILE SECTION.
FD NEWTRANS        VALUE OF ID "TRANSACTIONS".
01 NEWTR SZ 80.
FD CREDITFILE      VALUE OF ID "NEW"/"CREDITS"
    BLOCK CONTAINS 15 RECORDS.
01 CR-REC SZ 80.
FD DEBITFILE      VALUE OF ID "NEW"/"DEBITS"
    BLOCK CONTAINS 15 RECORDS.
01 DR-REC SZ 80.
SD SORTFILE.
01 SRT.
    03 CODE-KEY PIC 99.
    03 ACCOUNT-KEY PIC 9(10).
    03 DATE-KEY PIC 9(6).
    03 FILLER PIC X(62).
PROCEDURE DIVISION.
SORTIT SECTION.
SRTRN.
    SORT SORTFILE ON DESCENDING KEY CODE-KEY
        ASCENDING KEY ACCOUNT-KEY DATE-KEY
        USING NEWTRANS
        OUTPUT PROCEDURE RECORDS-OUT.
ENDIT.  STOP RUN.
RECORDS-OUT SECTION.
CREDITS-OUT.
    OPEN OUTPUT CREDITFILE DEBITFILE.
LOOP-CR.
    RETURN SORTFILE AT END GO TO XIT.
    IF CODE-KEY > 49
        WRITE CR-REC FROM SRT INVALID KEY GO TO IVK
        ELSE GO TO LOOP-CR.
LOOP-DR.
    WRITE DR-REC FROM SRT INVALID KEY GO TO IVK.
    RETURN SORTFILE AT END GO TO XIT
    ELSE GO TO LOOP-DR.
XIT.
    CLOSE CREDITFILE LOCK DEBITFILE LOCK.
    GO TO ENDIT.
IVK.
    DISPLAY "xxERROR TERMINATIONxx".
    DISPLAY CODE-KEY ACCOUNT-KEY.
ENDIT.  EXIT.
```

Input File to be Sorted:

```
121000000233040770
121000000233040970
121000000233041270
031200000042041270
031200000042041370
031200000042041670
031200000042040970
551000000012050170
551000000012052370
551000000012051470
551000000012050570
471000000012050570
471000000012052270
720900000243060270
720900000243061970
720900000243062170
710900000243062170
710900000243062670
124000000035062670
900000000017070370
```

Sorted Output Files:

New/Credits Output File

```
900000000017070370
720900000243060270
720900000243061970
720900000243062170
710900000243062170
710900000243062670
551000000012050170
551000000012050570
551000000012051470
551000000012052370
```

New/Debits Output File

```
471000000012050570
471000000012052270
121000000233040770
121000000233040970
121000000233041270
124000000035062670
031200000042040970
031200000042041270
031200000042041370
031200000042041670
```

7. USE OF SORT IN ALGOL LANGUAGE

ALGOL SORTING

The SORT procedure is activated by executing either the SORT or MERGE statements. This section describes the form and use of these statements. (Refer to the B 7000/B 6000 ALGOL Reference Manual, form 5001639, for exact syntactic definitions.)

SORT STATEMENT

The SORT statement provides a means for data, as specified by the <input option>, to be sorted and returned to the program, as indicated by the <output option>. The order in which data are returned is determined by the <compare procedure>. All procedures required by the SORT are in the form of standard procedures but have specific parameter requirements .

When the SORT statement is executed, the input and output files must be closed.

The format of the SORT statement is as follows. (The symbol "::=" denotes "is defined as".)

```
<sort statement>          ::= SORT (<output option>,
                                    <input option>,
                                    <number of tapes>,
                                    <compare procedure>,
                                    <record length>,
                                    <size specifications>)
                                    <restart specifications>

<output option>           ::= <file designator>|
                              <output procedure>

<output procedure>        ::= <procedure identifier>

<input option>            ::= <file designator>|
                              <input procedure>

<input procedure>         ::= <procedure identifier>

<number of tapes>         ::= <arithmetic expression>

<compare procedure>       ::= <procedure identifier>

<record length>           ::= <arithmetic expression>
```

```
<size specifications>     ::= <empty>|
                              <memory size>|
                              <memory size><disk size>|
                              <memory size><pack size>

<pack size>               ::= PACK <size>

<size>                    ::= <empty>|
                              <arithmetic expression>

<restart specifications>  ::= <empty>|
                              [RESTART = <arithmetic expression>]
```

Sort Parameters

<output option>

If a <file designator> is specified as the <output option>, the file is opened and SORT writes the sorted output on this file. On completion of the SORT statement, the file is closed.

If an <output procedure> is specified as the <output option>, SORT calls this procedure once for each sorted record and once to allow end-of-output action. This procedure must be untyped and must use two parameters. The first parameter must be call-by-value Boolean, and the second parameter must be a one-dimensional array with a lower bound of zero. The Boolean parameter is FALSE as long as the second parameter contains a sorted record. When all records have been returned, the first parameter is TRUE, and the second parameter must not be accessed.

In addition, SORT accumulates various statistics while sorting. This array of statistics is available in the record array parameter if the last Boolean parameter is TRUE. This statistical information is accessible only when using the <output procedure> option. (Refer to Appendix B.)

<input option>

If a <file designator> is used as the <input option>, the file is opened, and the records in that file are used as input to SORT. This file is closed after all records in the file have been read by SORT.

If an <input procedure> is used as the <input option>, the procedure is called to furnish input records to SORT. This <input procedure> must be a Boolean function with a one dimensional array as its only parameter. This procedure, on each call, either inserts the next record to be sorted into its array parameter or returns the value TRUE, indicating the end of the input data.

When a TRUE is returned by the <input procedure> SORT does not use the contents of the array parameter and does not call the <input procedure> again.

<number of tapes>

If an integrated tape/disk sort is desired, the <number of tapes> must be greater than zero, and this value specifies the number of tape files to be used in the sorting process. If the value of the <arithmetic expression> is less than three, three tapes are used. If the value of the <arithmetic expression> is greater than eight, eight tapes are used; otherwise, the number of tapes specified by the <arithmetic expression> is used.

<compare procedure>

The <compare procedure> is called by SORT to determine which of two records should be used next in the sorting process. This procedure must be a Boolean function with two parameters. Both parameters must be one-dimensional arrays. The Boolean value returned by the function should be TRUE if the array given as the first parameter is to appear in the output before the array given as the second parameter.

For the actual comparison, two strings may be compared according to the EBCDIC collating sequence by using a string relation, or an arithmetic comparison may be performed by using an arithmetic relation. Also, different keys or fields in the records may be compared. The comparison technique is determined entirely by the user in the <compare procedure>.

<record length>

The <record length> represents the length in words of the largest item that is presented to SORT. If the value of the <arithmetic expression> is not a positive integer, the largest integer not greater than the absolute value of the expression is used; that is, a <record length> of 12 would be used if an expression had a value of -12.995. If the value of the <arithmetic expression> is zero, the program terminates.

<size specifications>

The <size specifications> allow the programmer to specify the amounts of memory and disk or pack storage that may be used.

The <memory size>, if present, specifies the number of words of memory to be used in sorting. If <memory size> is unspecified, a default size of 12,000 words is assumed.

The <disk size>, if present, specifies the amount of disk storage, in words, to be used for the workfile. If <disk size> is unspecified, a default size of 600,000 words is assumed.

The <pack size>, if present, specifies the amount of storage on system resource pack, in words, to be used for the SORT workfile. If <size> is not specified, a default size of 600,000 words is assumed.

<restart specifications>

The <restart specifications> give SORT the ability to resume processing at the most recent checkpoint following the discontinuance of a program. The program must provide the logic to restore and maintain necessary information for it to continue from the point of interruption (such as, stack variables, arrays, files, and pointers).

The restart capability is implemented only for disk sorts.

Input, Output, and Compare Procedures in ALGOL Sorts

SORT functions by calling the input, output, or compare procedures (as appropriate) contained in the user program. The SORT passes array descriptors to the desired procedure of the user program. The descriptor normally points to a record area that is a portion of an array large enough to contain many records. If the user program is negligent in accessing the record area passed to the program, adjacent record areas may be accessed and compared incorrectly, or record content may be inadvertently modified. A SEGMENTED ARRAY ERROR, INVALID INDEX, or INVALID OPERATOR are other likely consequences of such action.

Sort Mode

The combination of <disk size> and <number of tapes> determines the sort mode as follows:

Number of tapes	Disk size	Mode
not = 0	0	tape only
not = 0	not = 0	ITD
0	not = 0	disk only
0	0	memory only

MERGE STATEMENT

The MERGE statement causes data in all files specified by the <merge option list> to be combined and returned. The <compare procedure> determines the sequence in which the data are combined.

The format of the MERGE statement is as follows:

```
<merge statement>    ::= MERGE (<output option>,
                                <compare procedure>,
                                <record length>,
                                <merge option list>)

<merge option list> ::= <merge option>|
                        <merge option list>,<merge option>

<merge option>       ::= <input option>
```

The MERGE statement merges two to eight presorted inputs into a single file. The inputs must be sorted in the same sequence, but the files may be of different lengths.

The <output option>, <compare procedure>, <record length>, and <input option> are as specified for the <sort statement>.

Because no sort workfiles are created, the <size specifications> are not required on the merge.

An example of an ALGOL disk sort, including the actual input and output files, appears on the following two pages.

ALGOL SORT EXAMPLE

```
BEGIN

COMMENT THIS IS AN EXAMPLE OF AN ASCENDING DISK SORT.  THE
        PROGRAM SORTS A CARD FILE WITH NAMES IN THE FIRST TWELVE
        COLUMNS AND OUTPUTS THE SORTED FILE ON THE LINE PRINTER.
        IF COLUMN 80 CONTAINS AN "S", THAT CARD IS THROWN AWAY
        AND NOT SORTED.  THE SORT USES AN INPUT PROCEDURE OPTION,
        AN OUTPUT FILE OPTION, A MEMORY SIZE OF 10000 WORDS, AND A
        DISK SIZE OF 100000 WORDS.
        ;

FILE CARD(BUFFERS=2,MAXRECSIZE=14,BLOCKSIZE=14);

FILE LINE(MYUSE=2,KIND=39,BUFFERS=2,MAXRECSIZE=20);

DEFINE SKIPCODES="S"#,SKIPFIELD=POINTER(R[13],8)+1#;
BOOLEAN PROCEDURE INPRO(R); ARRAY R[0];
      BEGIN
      LABEL GETACARD,XIT,EOFL;

GETACARD:  READ(CARD,14,R[*])[EOFL] ;
        IF SKIPFIELD = SKIPCODE THEN GO TO GETACARD
        ELSE BEGIN
             REPLACE POINTER(R[13],8)+2 BY "    ";
             % PAD BLANKS
             GO TO XIT;
             END;

EOFL:    INPRO := TRUE;
XIT:END;

BOOLEAN PROCEDURE COMP(R1,R2); ARRAY R1,R2[0];
       COMP := POINTER(R1,8) LSS POINTER(R2,8) FOR 12;

SORT(LINE,INPRO,0,COMP,14,10000,100000); % SORT CALL

END.
```

SORT

INPUT FILE TO BE SORTED

GLENN		CARD #0001
	S	CARD #0002
JACK		CARD #0003
EARL		CARD #0004
	S	CARD #0005
ROLLIE		CARD #0006
STEVE		CARD #0007
	S	CARD #0008
DAVE		CARD #0009
JOEL		CARD #0010
HARRY		CARD #0011
SHERRY		CARD #0012
RICHARD		CARD #0013
DICK		CARD #0014
BILL		CARD #0015
DON		CARD #0016
JIM		CARD #0017
	S	CARD #0018
CAROLYN		CARD #0019
	S	CARD #0020

SORTED OUTPUT

BILL
CAROLYN
DAVE
DICK
DON
EARL
GLENN
HARRY
JACK
JIM
JOEL
RICHARD
ROLLIE
SHERRY
STEVE

8. USE OF SORT IN PL/I LANGUAGE

PL/I SORTING

PL/I sorting is performed by the execution of the SORT statement. This section describes the form and use of this statement. (Refer to the B 7000/B 6000 Series PL/I Reference Manual, form 5001530, for exact syntactic definitions.)

SORT STATEMENT

The SORT statement provides a means for data to be sorted and returned to the program according to the specified options.

The format of the SORT statement is as follows:

```
⟨sort statement⟩   :: =  SORT ⟨sort identifier⟩ [On] ⟨sort option⟩
⟨sort option⟩      :: =  ⟨key option⟩ [⟨input option⟩]
                         [⟨output option⟩] [⟨memory option⟩]
⟨key option⟩       :: =  {[Ascending/Descending] [Key]
                         (⟨identifier⟩ ... )} ...
⟨input option⟩     :: =  USING FILE (⟨file expression⟩)
                         INPUT (⟨entry constant⟩)
⟨output option⟩    :: =  GIVING FILE (⟨file expression⟩)
                         OUTPUT (⟨entry constant⟩)
⟨memory option⟩    :: =  ENVIRONMENT (TAPES = ⟨constant expression⟩,
                         CORESIZE = ⟨constant expression⟩,
                         DISKSIZE = ⟨constant expression⟩)
```

MV1778

Sort Parameters

<sort identifier>

The <sort identifier> must be an aggregate that describes the individual records to be stored; it may not be controlled or based.

<key option>

Because the <key option> specifies the order in which records are to be sorted and the keys to be used in the sort, the key option must always appear in the SORT statement. The order of precedence of the keys is determined by the order of appearance of the key in the <key option>. Sort keys are subject to the following rules:

1. Each key must be defined in the <sort identifier>.
2. No variable-length keys are allowed.
3. All records must be of some fixed length, and the keys must be in the same location in each record.

<input option>

The <input option> may either be a file designation or an input procedure designation. If the <input option> is not explicitly stated, the <input option> USING FILE (SYSIN) is assumed. If an explicit file designation is used, the file must be declared as an input file. The input file passed to the SORT must be CLOSED PRIOR to the call of the SORT. If an input procedure is used, the procedure must have SORTINPUT declared in the <options list> of the procedure declaration. The input procedure is subject to the following rules:

1. The input procedure must have one parameter; it must be declared as CHAR(*).

2. The input procedure must return a bit (1) value.

3. A FALSE value ('0'B) must be returned by the input procedure until the end of the input data is encountered; at that time, a TRUE value ('1'B) must be returned.

4. As long as a FALSE value is being returned, the input procedure inserts the next record to be sorted into its parameter.

<output option>

The <output option> may either be a file designation or an output procedure designation. If the <output option> is not explicitly stated, the <output option> GIVING FILE (SYSPRINT) is assumed. If a file designation is used, the file should be declared as an output file. The SORT then writes the sorted output to this file. As with an input file, the output file must be CLOSED PRIOR to the call of the SORT. If an output procedure is used, the procedure must have SORTOUTPUT declared in the <options list> of the procedure declaration. The output procedure is subject to the following rules:

1. The output procedure must have two parameters. The first parameter must be declared as BIT (1), and the second parameter must be declared CHAR (*).

2. The first parameter contains a FALSE value ('0'B) as long as the second parameter contains a sorted record. When all records have been returned to the output procedure by SORT, the first parameter contains a TRUE value ('1'B), and the second parameter is not accessed.

<memory option>

The <memory option> specifies the number of tapes to be used by the SORT as well as the CORESIZE and the DISKSIZE to be allocated for the sort. The options may appear in any order, and any or all of the options may be deleted. The default values for any option not explicitly stated are as follows:

```
TAPES = 3
CORESIZE = 12000
DISKSIZE = 600000
```

The number of tapes used in the sorting process must be greater than or equal to three and less than or equal to eight.

An example of PL/I sort, including the actual input and output files, is as follows:

PL/I SORT EXAMPLE

```
SORTEXAMPLE: PROC; /* OPTIONS (MAIN) */

DCL AA FILE INPUT RECORD ENV (KIND='READER', MAXRECSIZE=80);

DCL BB FILE OUTPUT RECORD ENV (KIND='PRINTER',MAXRECSIZE=132);

DCL 1 COURSES,
       2  DEPT              CHAR (5),
       2  COURSENAME        CHAR (20),
       2  INSTRUCTOR        CHAR (10),
       2  REQUIRED          CHAR (1);

SORTIN: PROC(A) RETURNS (BIT (1)) OPTIONS (SORTINPUT);
       DCL A CHAR (*);
       ON ENDFILE (AA) GO TO EOF;

LOOP: READ FILE (AA) INTO (COURSES);
       IF REQUIRED = '*' THEN RETURN ('0'B);
                         ELSE GO TO LOOP;

EOF: RETURN ('1'B);
       END SORTIN;

SORT COURSES ON
       ASCENDING KEY (COURSES.DEPT, COURSES.COURSENAME,
       COURSES.INSTRUCTOR)
       INPUT (SORTIN)
       GIVING FILE (BB);

END SORTEXAMPLE;
```

The following is a listing of the input to the PL/I example:

```
MATH    TRIGONOMETRY          LAMBERT     *
HIST    AMERICAN HISTORY      JERONIMO    *
MUS     MIXED CHORUS          PAINTER
ENG     ELECTRONICS           SHERIDAN    *
MATH    CALCULUS I            GUILFORD    *
ENGL    READING COMP          WOODS       *
MATH    CALCULUS II           GUILFORD    *
MATH    STATISTICS            GALLOP
MATH    MATRIX THEORY         WAIHAU      *
HIST    WESTERN CIV           GREENLEAF
MUS     MUSIC APPREC          PAINTER     *
ENG     SURVEYING             SHERIDAN    *
ACCT    ACCOUNTING I          BLOCK       *
BUS     BUSINESS LAW          BAILEY      *
MATH    ALGEBRA               MULBERRY    *
MUS     INSTRUMENTAL MUSIC    BLAIR       *
ACCT    ACCOUNTING II         BLOCK       *
MATH    LINEAR ALGEBRA        GUILFORD
MATH    CALCULUS I            AMSDALE     *
ACCT    COST ACCOUNTING       BLOCK       *
ACCT    ACCOUNTING I          ANOLA       *
HIST    AMERICAN HISTORY      LIVERMORE   *
```

After execution of the above PL/I SORT, the output appears as follows:

```
ACCT    ACCOUNTING I          ANOLA       *
ACCT    ACCOUNTING I          BLOCK       *
ACCT    ACCOUNTING II         BLOCK       *
ACCT    COST ACCOUNTING       BLOCK       *
BUS     BUSINESS LAW          BAILEY      *
ENG     ELECTRONICS           SHERIDAN    *
ENG     SURVEYING             SHERIDAN    *
ENGL    READING COMP          WOODS       *
HIST    AMERICAN HISTORY      JERONIMO    *
HIST    AMERICAN HISTORY      LIVERMORE   *
MATH    ALGEBRA               MULBERRY    *
MATH    CALCULUS I            AMSDALE     *
MATH    CALCULUS I            GUILFORD    *
MATH    CALCULUS II           GUILFORD    *
MATH    MATRIX THEORY         WAIHAU      *
MATH    TRIGONOMETRY          LAMBERT     *
MUS     INSTRUMENTAL MUSIC    BLAIR       *
MUS     MUSIC APPREC.         PAINTER     *
```

9. EFFICIENT USE OF THE SORT

SORT EFFICIENCY

This section explains some of the factors that affect the overall efficiency of the sorting process.

Core Estimate

In sorting, production of a small number of long strings is desired, because fewer merge passes will be required. Appendix B illustrates that the user-supplied memory estimate determines stringing and merging vector sizes which, in turn, control string length and merging.

Generally, a memory estimate of 4000 or more is recommended. Memory estimates of 20,000 or more are recommended only on large systems.

Number of Work Tapes

Above a certain number, the number of work tapes provided results in decreases in total sorting time. However, this decrease in sorting time does not justify their use. Gains made by using more than five sort tapes are marginal. The sort tape limit is eight tapes.

User Input/Output Files

A large portion of sort time is consumed in communicating with user files. The I/O procedures on these files should be made as efficient as possible. The files should be blocked in large increments (greater than 500 words).

Character Sets

Since character set "translation" is required when sorting with BCL records, use of the EBCDIC character set is recommended whenever possible.

Comparison Technique

The comparison method is an important factor when sorting. Because the compare procedure may be called millions of times in very large sorts, the procedure should be made as efficient as possible.

In COBOL, the length and number of keys directly affects the amount of time required for comparison. A large number of keys scattered in the record should not be used because setup time increases comparison overhead. Arithmetic or numeric comparisons are generally faster than string comparisons.

In ALGOL, the compare procedure should return TRUE when the two arrays are unequal, and the first array must precede the second array. A FALSE should be returned when the arrays are equal or when the second array must precede the first array.

Variable-Length Records

Variable-length records should be edited into fixed-length records by an input procedure.

SORT MODE

Table 9-1 illustrates some characteristics of various sort modes.

Table 9-1. Characteristics of Sort Modes.

Disk Only	Tape Only	ITD
Disk only is generally the fastest mode.	Input file may be indefinite length.	Disk develops longer strings on tape.
Disk is the most reliable peripheral.	A particular machine configuration is required.	Input file may be definite length.
Less operator intervention is required.		
Sort is limited by disk resource.		

KINDS OF SORTS

Two kinds of sorts are record and tag.

Record Sort

A record sort is the usual method of sorting. A record sort occurs unless the user intentionally causes a "tag sort". A record sort is defined as a sort in which records are continually handled throughout the stringing and merging phases. In other words, the sort algorithms are only concerned with the various keys, but the entire record (including the keys) is carried along throughout the sorting process.

Tag Sort

A tag sort is more sophisticated than a record sort and, under certain conditions, can be substantially faster. The basic idea of a tag sort is that only the key(s) plus the address of each record is handled throughout the sorting process. The data records remain in place until the final merge of the tags begins; at that time, the addresses contained within the tags are used to retrieve the records in the correct sequence.

In creating a tag sort, the user is required to provide an INPUT PROCEDURE that filters the incoming records, writing them to a disk file after extracting the sort key(s) and appending the disk address of the location of the entire record. This tag is then submitted to the stringing phase. Figure 9-1 shows the extraction of the key(s) from the incoming record and the building of the tag.

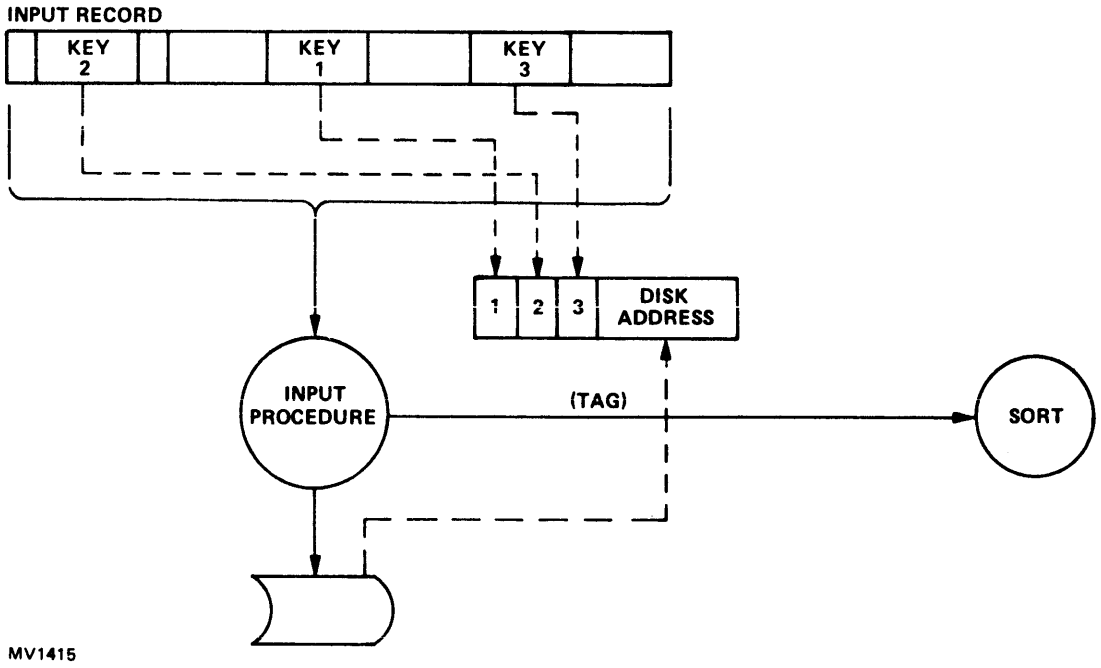


Figure 9-1. Creating a Tag.

The keys are arranged by the INPUT PROCEDURE in a contiguous string. This structure eliminates the need to "bounce" over the record in accumulating the keys whenever a compare is required. The information actually handled by the SORT is smaller than in a record sort (only the key[s] and disk address are handled).

During the final merge phase, the OUTPUT PROCEDURE uses the address portion of the tag to access the records and do whatever is required.

Figure 9-2 shows a functional diagram of a tag sort that has a nondisk input file(s). The input file must be copied in its entirety to disk.

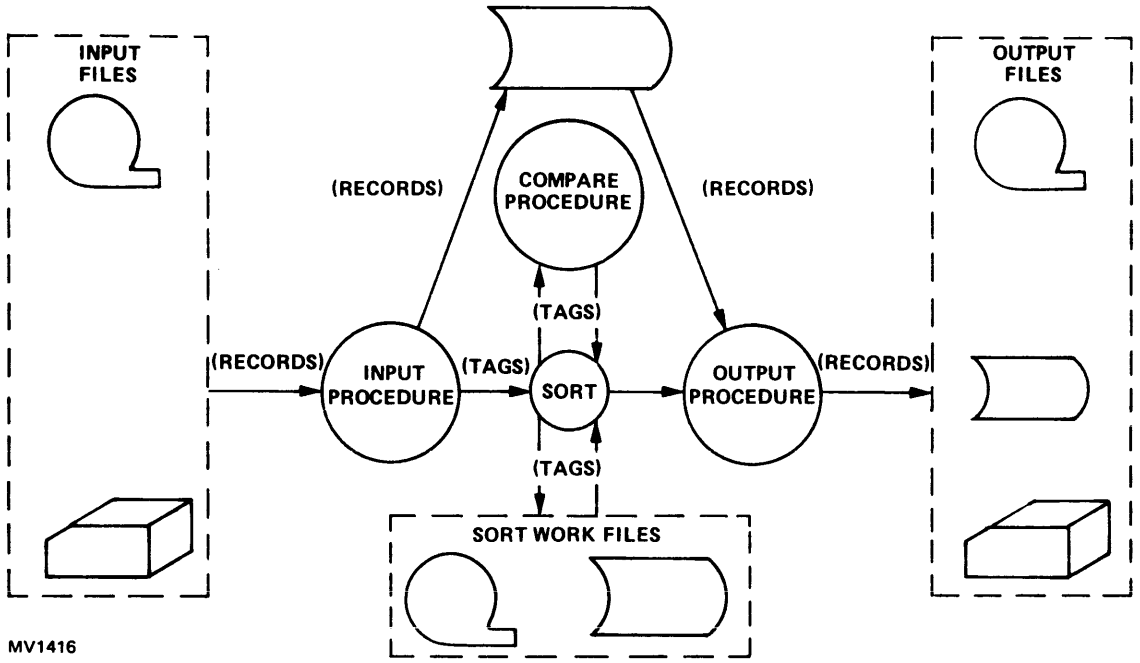


Figure 9-2. Tag Sort, Nondisk Input File.

Figure 9-3 shows the functional diagram of a tag sort that has a disk file as its input medium. The input file is read twice: once by the INPUT PROCEDURE (in a serial fashion) for the stringing phase, and once again by the OUTPUT PROCEDURE (in a random fashion) for the final merge phase.

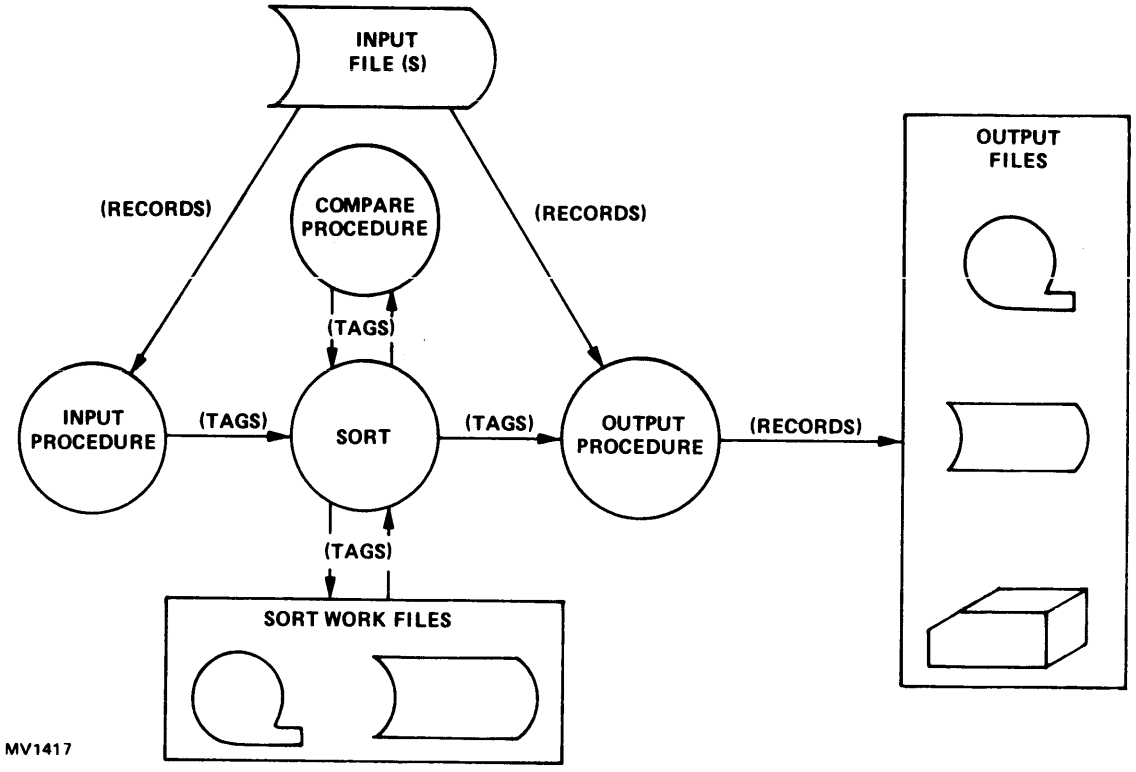


Figure 9-3. Tag Sort, Disk Input File.

RECORD SORT VS TAG SORT

The decision to do a record or tag sort depends on many variables (such as, site disk capacity, record size, and time allotted to write the sort). If the sort is for "production" and is heavily used, both record and tag sorts should be used to determine which sort produces the best results.

SUGGESTIONS FOR MORE EFFICIENT SORTING

Selection of SORT parameters is not always a simple matter when the amount of data varies widely from run to run and the user is concerned with total system efficiency. Sorting can be a significant part of the work load of a computer installation and should be used judiciously. In many instances, the default parameters do not provide efficient utilization of system resources. The relative priority of the job should be a definite consideration in the selection of SORT parameters. Memory size is certainly the most volatile SORT parameter. Because the SORT may have a 65,535-record Sort Vector and can do a 65,535-order merge with as many as 256 buffers per string, the upper limit on the amount of memory which could be utilized is 1.099 trillion words; therefore, something less than maximum sorts must be considered. Providing more memory (until the sort is entirely completed in memory) should yield faster sorts. However, some sets of data reach a point where slower sorts occur as memory size is increased. A point of diminishing return usually occurs before an entirely in-memory sort is reached.

Some general guidelines for disk sorting are as follows:

Fast Sorts -	Memory size should provide enough space to contain at least 2,000 records.
Reasonably Fast Sorts -	Memory size should provide enough space to contain at least 1,200 records.
Adequate Sorts -	Memory size should provide enough space for 600 records as a general rule.

To use the above guidelines, the sort record size must be converted to the number of words required to contain a single record. For example, a one-character record requires one word, and fifteen 6-bit characters require two words, while fifteen 8-bit characters require three words. When record size is converted to words, three additional words must be added to record size (used by SORT) and then this new record size must be multiplied by the desired number of records. After memory has been computed for the number of records times record size, 1,500 words must be added for sort working space.

Some general guidelines for tape sorting are as follows:

Fast Sorts -	Memory size should provide enough space for 300 records per work tape.
Reasonably Fast Sorts -	Memory size should provide enough space for 200 records per work tape.
Adequate Sorts -	Memory size should provide enough space for 100 records per work tape.

Record size must be converted to words with three additional words added (as stated above) and then multiplied by the number of tapes specified in the SORT statement. Again, 1,500 words must be added to provide for sort working space.

Tape sorts are similar to disk sorts in that providing more memory generally yields faster sorts. However, the point of diminishing return is more data dependent for tape sorting. Generally, using more sort work tapes rather than providing large additional increments of memory is more efficient. Providing more memory and more tapes is ideal when speed is the most important factor. Tape sorts should run satisfactorily in the background while other jobs are in the mix. Use of good tapes when doing tape sorts is highly recommended.

ITD sorts are capable of improving tape sorts by a substantial factor (50 percent or more). The reason for this degree of improvement is that fewer strings are created on tape which causes tape merging to be completed much sooner. The amount of improvement depends on the inherent sequence of the data and the amount of disk provided. In most cases, 100,000 words of disk (or less) is sufficient to obtain the increased speed from an ITD sort.

Disk sorting speeds can be affected by the amount of disk specified by the program. This circumstance results from SORT being unable to merge as many disk strings because insufficient disk is available to contain the output of the merge phase. SORT, therefore, merges fewer strings (when possible) and attempts to reduce the risk of being terminated with a SORT ERROR 5. Estimating a proper amount of disk is difficult because of the gaps created by unfilled buffers at the end of strings. However, a method for estimating disk space is suggested below:

1. Convert the record size to words as stated above (do not add three additional words).

2. Multiply the record size (in words) by the number of records to be sorted.

3. The number obtained by step 2 should then be multiplied by:

    a. 2.25 to obtain a safe estimate.

    b. 1.5 to obtain a near minimum estimate.

    c. 3.5 or more if a restartable sort is being done.

Much of the memory used by SORT is nonoverlayable or save space. For this reason, SORT can have a definite impact on the throughput of other jobs that are executing concurrently with the SORT. Sorting programs that contain lengthy INPUT or OUTPUT procedures can contribute in large measure to this condition. The use of INPUT or OUTPUT procedures should not be discontinued but should be considered in proper perspective for the job to be accomplished. Overall system performance can be improved, in some cases, by having the INPUT procedure produce a file which is read by SORT and having SORT produce a file which is processed by the OUTPUT procedures. The process of calling INPUT or OUTPUT procedures does not present any undue burden to SORT or the system.

Blocking factors of input and output files can have a definite effect on sort timings even if INPUT or OUTPUT procedures are used to read or write the files. To prevent this I/O bound situation, the output file should contain approximately 100 (or more) records per block; 50 (or more) records per block is usually sufficient for the input file. Sorts have been run as much as four times faster when input and/or output file blocking was improved. Other cases could be stated in which both larger and smaller improvements were seen.

Historically, sorts are characterized as being processor bound during the stringing phase and I/O bound during the merge phase. SORT can and does operate in this fashion; however, it always attempts to be processor bound in both the stringing and merge phases. Unless the memory size specified is

relatively small (for the particular sort), SORT achieves the goal of being processor bound. When this condition is obtained, speed improvements can be realized only by methods that reduce processor time. When INPUT or OUTPUT procedures are used, they are candidates for this kind of improvement. The largest potential gain lies in improvement of the COMPARE procedure. The COMPARE procedure is called many times for each record; the INPUT or OUTPUT procedures, however, are called only once for each record. Programs can obtain improvement by simplifying the individual keys and consolidating them into a single key. In ALGOL sorts, partial word compares are normally faster than string compares when characters within a word are tested. COBOL sorts should use 8-bit characters for string comparison (translate them to 8-bit on input and back to 6-bit on output) whenever possible. Decreasing the amount of processor time helps system throughput as well as reducing sort timings. With certain kinds of sorts characterized by small keys and large record sizes, tag sorting may be advantageous. The total amount of disk required to complete the job is probably smaller also, because the SORT requires less. Retrieving the output records is likely to be the most time-consuming factor and is highly data dependent.

Knowledge of the particular sort and experimentation can provide better retrieval methods.

APPENDIX 11A.

SORT ERROR MESSAGES

SORT ERROR MESSAGES

All error messages are displayed on the system display in the form:

<job number> SORT ERROR #NN

Error Number	Definition
1	The record size specified was zero.
2	Count of sort input records and output records does not agree (internal sort problem).
3	Insufficient memory was specified for a memory sort. Disk size and number of tapes are both zero.
4	Sort disk was exhausted during the stringing phase and no tapes were specified.
5	Sort disk was exhausted during the merge phase and no tapes were specified.
6	Input file passed to SORT was already open (the input file must be closed when passed to SORT).
7	During a tape or ITD sort the block number of the last record read from a sort work tape did not match the expected block number.
8	The output file passed to the SORT was not large enough to contain the output, and the SORT was unable to expand the output file. (Increase the size of the output file or decrease the number of records to be sorted.)
9	The output file passed to SORT was already open (the output file must be closed when passed to SORT).
10	An irrecoverable I/O error occurred while reading a sort work tape or work disk file.
11	An irrecoverable I/O error occurred while writing a sort work tape or work disk file.
12	An irrecoverable I/O error occurred while reading or writing control records in the SORT control file.
13	An irrecoverable I/O error occurred while writing the user output file.
14	An irrecoverable I/O error occurred while reading

the user input file.

15 A restart was attempted, but the record size (or character size of the record) did not match the originating sort. When a restartable sort is not able to continue, it saves restart information so that this error occurs on subsequent restart attempts.

16 A restart was attempted, and the user input file reached end-of-file before the restart record was read. The user input file is shorter at restart time than the original file.

17 A restart was attempted, but the SORT was unable to obtain the necessary restart information from the control file. (May be due to parity errors.)

19 The SORT is terminating because an irrecoverable error occurred while reading the sort workfile and some output records have already been passed to the user OUTPUT procedure. This error termination only occurs for restartable sorts with OUTPUT procedures. Since the sort is restartable, a restart should be made so that the SORT does the neccessary recovery.

84 This error occurs because the control file is not large enough to contain all control records. It is an internal sort error and can be circumvented by specifying more disk and/or a different memory size.

The following errors appear on the display like any other sort error message. The SORT does not terminate as a result of these errors.

30 The control file is not large enough to contain two copies of the control records. Two copies are maintained for sorts with error recovery. This error is an internal sort problem and can be circumvented by specifying more disk and/or a different memory size. The SORT continues when this error occurs; however, this function of error recovery is disabled.

31 An irrecoverable I/O error occurred while writing the control file for an error recovery sort. One copy of the control records is discarded, and the SORT continues using one copy of the control records.

32 An irrecoverable I/O error occurred while writing the sort workfile. Error recovery mode is abandoned, and the SORT continues as a normal restartable sort.

33 Insufficient disk space was provided for the workfile to contain three copies of the data. This situation only happens when error recovery mode is requested. The SORT continues as a normal restartable sort with error recovery reset.

APPENDIX 11B.

SORT RUN-TIME INFORMATION

FILE NAMES

The following list shows the file names associated with the sort workfiles during execution of the sort.

```
Sort Disk Files = SORT/DISKC
                  SORT/DISKF

Sort Tape Files = SORT0M<task number>
                  SORT1M<task number>
                  SORT2M<task number>
                  SORT3M<task number>
                  SORT4M<task number>
                  SORT5M<task number>
                  SORT6M<task number>
                  SORT7M<task number>
```

SORT STATISTICAL ARRAY

The sort statistical array contains data collected while the SORT was executing. A copy of the array is written into a file titled SORT/STATISTICX and in some cases is returned to the user program. A program SYMBOL/SORTSTAT (source language) or SYSTEM/SORTSTAT (object code) is provided on the system release tapes. This program reads the SORT/STATISTICX file and produces a report from the sort statistical information. The sole purpose of this program is to guide those users who are interested in the sort statistical information. Any other use of this program or the sort statistical information is left entirely to the user. When the contents of the sort statistical array are changed, those changes are reflected in the SYMBOL/SORTSTAT and SYSTEM/SORTSTAT program. The MCP must be compiled with the $ option SORTSTAT set. This option must be set prior to the first reference to the option.

The following describes the content of the sort statistical array. (Unless otherwise specified, units are words.)

WORD	EXPLANATION OF CONTENTS
0	A code word containing various parameters used by the SORT.
1	Sort memory size specified by user program.
2	Stringing matrix size determined by SORT in record units.
3	Output block size of sort workfile.
4	Input block size of sort workfile.
5	Disk input. Contains six fields of eight bits. Each field reflects the number of rows (mod 256) for various types of disk files.

6	Disk output. Contains six fields of eight bits. Each field reflects the number of rows (mod 256) for various types of disk files.
7	Merge matrix size determined by SORT in record units.
8	Number of strings created during stringing phase.
9	Processor Time. [47:24] - Stringing phase (units of 2.4 microseconds). [23:24] - Beginning of sort (units of 2.4 microseconds).
10	Time check 1. Beginning of sort (units of 2.4 microseconds).
11	Run date. TIME (15).
12	Time check 2. End of stringing phase (units of 2.4 microseconds).
13	Time check 3. End of merge phase (units of 2.4 microseconds).
14	MCP level.
15	Disk size specified by user program.
16	Number of comparisons (calls on user compare procedure) during sort.
17	Record size specified by user program.
18	Number of input records.
19	Number of work tapes specified by the program.
20	Work tape or merge input unit types. Contains eight fields of six bits. Each field reflects the unit type of the merge file or work tape.
21	Unit information. Contains 10 fields of three bits that reflect the recording density of a specific file and two fields of six bits that reflect the unit type of the input and output file.
22	Disk workfile. Contains six fields of eight bits. Each field reflects the number of rows (mod 256) for various types of disk files.
23	Not used.
24-29	Program name. Format is standard form representation of file names.

The following example illustrates the actual sort statistics for the COBOL sort program illustrated previously.

```
************************************************************
************************* S O R T ************************
****** S T A T I S T I C A L  A N A L Y S I S ******
************************************************************

                              SORTINFO

FUNCTION:  SORT        DATE:  8/6/76      TIME:  15:5:20.95     MCP:  MARK 2.8.0

DESCRIPTION:

     RECORD COUNT:      20    RECORD SIZE(CHARS):     80       SOURCE LANGUAGE:  COBOL

     INPUT  OPTION:       FILE            OUTPUT OPTION:       PROCEDURE

USER PARAMETERS:

     MEMORY SIZE:    12000    DISK SIZE(WORDS):     20000      NO. WORK TAPES:   0

CONFIGURATION:

     INPUT:          CARD READER

     SORT DISK:      MODULAR DISK

ALLOCATION:

     SORT DISK BLOCK SIZE(WORDS): 840        SORT TAPE BLOCK SIZE(WORDS):       0

     STRINGING VECTOR SIZE:       549        MERGING VECTOR SIZE:               5

PERFORMANCE STATISTICS:          SUMMARY     STRINGING          MERGING

     NO. COMPARISONS:                60           60                0
     COMPARISONS PER RECORD:       3.00         3.00             0.00

     NO. STRINGS CREATED:              DISK     0            TAPE            0

     AVERAGE STRING LENGTH(RECORDS):   DISK  0.00            TAPE         0.00

     NO. MERGE PASSES REQUIRED:        DISK  0.00            TAPE         0.00

TIMING ANALYSIS:                 ELAPSED       PROCESSOR     INPUT/OUTPUT

     STRINGING PHASE:       0: 0:38.81     0: 0: 0.52     0: 0: 2.02

     MERGING PHASE:         0: 0: 0.00     0: 0: 0.00     0: 0: 0.00

     TOTAL:                 0: 0:38.81     0: 0: 0.52     0: 0: 2.02

     RECORDS PER MINUTE:         30.92          2305.26         593.66

INPUT RECORD SEQUENCE:               DISK  0.00 %            TAPE      0.00 %

MV1418
```

APPENDIX 11C.

SORT COLLATING SEQUENCE

EBCDIC GRAPHIC	EBCDIC INTERNAL	HEX. GRAPHIC
BLANK	0100 0000	40
[	0100 1010	4A
.	0100 1011	4B
<	0100 1100	4C
(	0100 1101	4D
+	0100 1110	4E
|	0100 1111	4F
&	0101 0000	50
]	0101 1010	5A
$	0101 1011	5B
*	0101 1100	5C
)	0101 1101	5D
;	0101 1110	5E
	0101 1111	5F
-	0110 0000	60
/	0110 0001	61
,	0110 1011	6B
%	0110 1100	6C
_	0110 1101	6D
>	0110 1110	6E
?	0110 1111	6F
:	0111 1010	7A
#	0111 1011	7B
@	0111 1100	7C
'	0111 1101	7D
=	0111 1110	7E
"	0111 1111	7F
(+)PZ	1100 0000	C0
A	1100 0001	C1
B	1100 0010	C2
C	1100 0011	C3
D	1100 0100	C4
E	1100 0101	C5
F	1100 0110	C6
G	1100 0111	C7
H	1100 1000	C8
I	1100 1001	C9
(!)MZ	1101 0000	D0
J	1101 0001	D1
K	1101 0010	D2
L	1101 0011	D3
M	1101 0100	D4
N	1101 0101	D5
O	1101 0110	D6
P	1101 0111	D7
Q	1101 1000	D8
R	1101 1001	D9
ç	1110 0000	E0
S	1110 0010	E2
T	1110 0011	E3
U	1110 0100	E4
V	1110 0101	E5
W	1110 0110	E6
X	1110 0111	E7
Y	1110 1000	E8
Z	1110 1001	E9
0	1111 0000	F0
1	1111 0001	F1
2	1111 0010	F2
3	1111 0011	F3
4	1111 0100	F4
5	1111 0101	F5
6	1111 0110	F6
7	1111 0111	F7
8	1111 1000	F8
9	1111 1001	F9

APPENDIX 11D.

B 7000 COMPARE ANALYSIS

COMPANALYZER

Enhanced SORT performance is provided through the use of the SORT compile-time option, COMPANALYZER. When set, COMPANALYZER causes code to be included in the SORT which performs analysis of the compare procedure provided by the user. This analysis attempts to move the compare procedure local to the environment of the SORT in order to reduce the overhead of compare procedure entry, which normally occurs via an SIRW.

In order to successfully complete the analysis, address couples of local and global reference must be properly mapped into a new environment so that the SORT may use normal IRW referencing for compare procedure entry. If the compare procedure cannot be moved local to the SORT, the optimization is terminated and the SORT continues to run in the normal manner referencing the user's compare procedure with an SIRW.

INLINECOMP

INLINER is a SORT procedure whose purpose is to recognize in-line compare procedures in ALGOL and to map them into existing structures within the existing SORT program.

Functionally, INLINER builds a code word in the manner of the COBOL compiler. This process allows the existing mechanisms (for COBOL in-line cases) to be utilized and prevents extensive duplication of code.

Significant savings are achieved in single-key cases by completely eliminating all calls on the user's compare procedure.

INLINER is required to build one of two types of code words depending on the class of the compare being performed (that is, numeric or character). The user's object code is examined, and a determination of the class is made.

A copy of the compare procedure is moved local to the SORT by the COMPMOVERL procedure. During this process, the code string is examined and an in-line determination is made.

The user is advised to run with both COMPANALYZER and INLINECOMP set, since significant savings in sort times usually result with relatively little overhead.

STRINGING PROCEDURES

The stringing procedures are reorganized and optimized in order to take advantage of B 7000 hardware features.

OPTIONS

All existing B 6000 options have been retained in the B 7000 SORT. In addition, four options have been added:

COMPANALYZER
INLINECOMP
COMPTRACE
DEBUGDUMP

The above option cards are embedded in the MCP symbolic and require recompilation in order for the default value to be changed.

COMPANALYZER (SET)	When reset, causes omission of COMPMOVEL and associated procedures on recompilation of the MCP.
INLINECOMP (SET)	When reset, causes omission of INLINER and associated procedures on recompilation of the MCP.
COMPTRACE (RESET)	When set, produces debugging aids for COMPMOVERL and INLINER.
DEBUGDUMP (RESET)	When set, produces debugging aids for COMPMOVERL and INLINER.

Both COMPMOVERL and INLINER are produced by default. INLINER is dependent on COMPMOVERL; therefore, compilation of the SORT with COMPANALYZER reset causes an implicit resetting of INLINECOMP. INLINER may be explicitly omitted by recompiling with INLINECOMP reset.

10. SORT RECOVERY CONSIDERATIONS

This section describes use of the restart feature of SORT in detail and provides information about error recovery during sorting.

RESTART

The SORT can resume processing at the most recent checkpoint following the discontinuance of the program. Operation of the SORT in this mode provides the necessary restarting information for the SORT and requires certain program inputs (defined in subsequent text). The program must provide logic to restore and maintain stack variables, arrays, files, pointers, and so forth, that are defined for and by the program. In other words, the program must provide the means to restore everything necessary for it to continue from the point of interruption. This capability may be simple or complex and is entirely program dependent.

Restart capability is implemented for disk sort only; however, a partially restartable ITD sort may also be possible. When tape files are not in the tape phase of any ITD sort, it functions as a disk sort. After the data are written from disk to tape (during an ITD sort), the SORT cannot be restarted. A sort can be started as a disk-only restartable sort with insufficient disk provided to accomplish the sort. When this situation exists, the SORT terminates with SORT ERROR 4 or 5. After error termination, the sort is restarted as an ITD sort by indicating a RESTART and specifying the number of tapes desired. Any other kind of restart is impossible under this condition. If a restartable sort terminates during the first output of data from disk to tape (possibly as a result of an irrecoverable tape I/O error), the SORT can be restarted and the data written to tape as if no problem had occurred.

When using the SORT in restartable mode, unique file titles should be given to the two sort disk files. This procedure is accomplished by using the title attribute of the files. (This procedure is described in subsequent text.) Conflicts may arise when two or more sorts use identical file titles for their sort disk files.

When the SORT is attempting to restart a previously incomplete sort, a minimal amount of information is verified to ensure that continuation is compatible with the previous sort. The two items verified are sort record size and character size of the sort record characters. For ALGOL programs, record size is explicitly specified by the program while character size is zero (default size is eight). COBOL programs use the SD to determine sort record and character size. When record size or character size does not match the previous sort, error termination occurs. Modification of other sort parameters (except number of tapes as previously stated) is allowed. Different values for memory size or disk size are ignored and the original values are used. However, both values must be valid non-zero values. Different procedures can be specified (for input, output, or comparing) if desired, and files may be interchanged with INPUT or OUTPUT procedures. The program requesting the restart need not be the originating program. Because the SORT can only attempt to meet the user request and cannot determine appropriateness of requests, the user must ensure that the desired results are obtained.

Language Syntax Extensions

Implementation of restart and I/O error recovery requires extensions to the SORT statements in the COBOL and ALGOL compilers.

The extended syntax for COBOL is as follows:

```
SORT file-name-1
ON {ASCENDING/DESCENDING}
KEY data-name-1 [, data-name-2] ...
[, ON {ASCENDING/DESCENDING}
KEY data-name-3 [, data-name-4] ... ] ...
{USING file-name-2/INPUT PROCEDURE IS
section-name-1 [{THROUGH/THRU}section-name-2]}
{GIVING file-name-3/OUTPUT PROCEDURE IS
section-name-3 [{THROUGH/THRU}section-name-4]}
[MEMORY SIZE{formula/data-name-5/literal-1}]
[DISK SIZE{formula/data-name-6/literal-2}]
[RESTART IS{formula/data-name-7/literal-3}]
```

The value of the least significant (rightmost) five bits of the formula DATA-NAME-7 or LITERAL-3 is passed to the SORT to indicate the desired sort action. Detail information concerning COBOL sort is contained in the B 7000/B 6000 Series COBOL Reference Manual, form 5001464.

The syntax for ALGOL is as follows:

```
<sort statement> ::= SORT (<output option>,
                           <input option>,
                           <number of tapes>,
                           <compare procedure>,
                           <record length>,
                           <size specifications>)
                           <restart specifications>

<restart specifications> ::= <empty>|
                             [RESTART=<arithmetic expression>]
```

The value of the least significant (rightmost) five bits of the restart expression is passed to the SORT to indicate desired action. Refer to the B 7000/B 6000 ALGOL Reference Manual, form 5001639, for more information pertaining to the SORT statement.

RESTART PARAMETER VALUES

The SORT inspects various bits of the restart parameter to determine the requested restart action. The user must supply proper file title attributes for the two disk workfiles if these files were previously label equated. Individual bits and combinations of bits can be set by the program to control the SORT. The bits and their meanings are as follows:

Bit 0: On. The program is restarting a previous sort. The SORT tries to open its two disk files and obtain restart information. After successfully obtaining this information, the SORT continues from the last known restart point.

Off. The SORT is starting from the beginning. If the sort is a restartable sort and previous sort files with identical titles exist, those sort files are removed and replaced by new sort files.

Bit 1: On. The program is requesting a restartable sort. The SORT saves its two internal files and can be restarted on program request. If bit 2 is on, bit 1 is set by default.

Off. A normal sort is requested, and no sort files are saved (unless bit 2 is on, which sets bit 1 by default).

Bit 2: On. The program is requesting a restartable sort and desires extensive error recovery (from I/O errors). With this option set, if I/O errors occur while accessing either of the two sort files, the sort attempts to backtrack and remerge strings as necessary. To use this option, the program must provide at least three times as much disk space as required to contain the input data. When less space is provided, the SORT emits the message "change to restartable only mode" and continues the sort without further capability to backtrack.

Off. Recovery from internal errors is not requested.

Bit 3: This bit has meaning only if a restartable sort is requested. Use of this option controls the SORT during the stringing phase as the user input is being read by the SORT. Use of this bit determines how the SORT restarts (when a restart is requested) only if the restart occurs while the SORT is in the stringing phase.

On. The program desires the SORT to restart at the beginning of the user's input. This restart is the equivalent of starting an entirely new sort. In case the restarted sort had passed from the stringing phase into the merge phase, it continues from the merge phase. This bit may be set during a restart even if it was not initially set. Once set, it cannot be reset by subsequent restarts.

Off. The program desires the ability to restart at

the last restart point that occurred during the stringing phase. If the SORT is still in the stringing phase, it skips over the records already processed and continues from the last restart point. This process is described in more detail in subsequent text. If the SORT is in the merge phase, it continues from the last merge phase restart point. Use of this option (by not setting the bit) is normally less efficient than not using the option because more strings are created during the stringing phase.

Bit 4: This bit is reserved for expansion and is not currently used by the SORT.

When a program is initially starting a sort and desires restart ability the restart value should be:

1. Decimal 2 (bit 1 on) if a restartable sort is desired that is capable of restarting at any point during the stringing or merge phase.

2. Decimal 10 (bits 1 and 3 on) if a restartable sort is desired that can restart at any point during the merge phase but only at the beginning of the stringing phase.

3. Decimal 4 or 6 (bit 2 on or bits 1 and 2 on) if a restartable sort is desired that can attempt extensive recovery from internal sort I/O errors and can restart at any point during the stringing or merge phase.

4. Decimal 12 or 14 (bits 2 and 3 on or bits 1, 2, and 3 on) if a restartable sort is desired that can attempt extensive recovery from internal sort I/O errors and can restart at any point during the merge phase but only at the beginning of the stringing phase.

5. Decimal 1, 3, 5, or 7 (significant bits are bit 0 on and bit 3 off) if a previously incomplete sort is desired that can be restarted. The prior incompleted sort must have been capable of restart, and the two sort disk files must be present. A restart is attempted using the values obtained from the sort files. The previous setting of bit 3 controls the SORT if it is restarted during the stringing phase. The previous values of bits 1 and 2 are used.

6. Decimal 9, 11, 13, or 15 (significant bits are bits 0 and 3 on) if a restart is desired of a previously incomplete sort and if a restart from the beginning of input is desired during the stringing phase. The prior incompleted sort must have been capable of restart, and the two sort disk files must be present. A restart is attempted using the values obtained from the sort files. Bit 3 is set and remains set through all subsequent restarts. Bits 1 and 2 take on their previous values.

7. Decimal 0 or 8 (no bits on or bit 3 on) causes the SORT to do a normal sort with no restart capability.

RESTARTING DURING STRINGING PHASE

Restarting during the stringing phase (while SORT is still reading input records) requires special consideration. If SORT has passed a file, a seek is done or records are read until the desired restart point is reached. An INPUT PROCEDURE presents a different problem, however, because the program must find the proper restart record. To accomplish this desired result, the SORT places values in the first word of the array passed to the INPUT PROCEDURE. The values are negative or positive integers in binary form or zero to indicate that nothing special is happening. A positive integer is placed in the first word (word 0) to tell the INPUT PROCEDURE the relative number of the next record desired by the SORT (if the SORT has previously processed and saved 99 records it requests record number 100). A positive non-zero integer occurs in the first word only once and is then on the first call to the INPUT PROCEDURE. SORT places a negative non-zero integer in the first word to inform the INPUT PROCEDURE that SORT has just established a restart point. The number returned represents (in absolute value) the number of records saved (for restart purposes) by SORT. This information can be used by the program to establish its own separate restart points.

ERROR RECOVERY

SORT contains extensive internal error recovery ability for irrecoverable I/O errors that occur as a result of accessing a sort disk file. The sort disk files under discussion are the workfiles which contain the data being sorted and the control file which contains control information for SORT. These two files are referenced as the workfile and control file respectively; their internal names and external titles are described in subsequent text.

I/O error recovery logically segments into several areas of interest related to the file in question and the kind of error encountered. The degree of recovery possible is always dependent on the request for error recovery by the program. When error recovery is requested, SORT maintains two copies of each record in the control file and makes a second copy of the original strings of input data in the workfile. With error recovery, the control file is logically segmented into two files with a duplicate record maintained in both halves of the file while the workfile is logically segmented into thirds. Duplicate records are created in the workfile during the stringing phase only and are used for error recovery when the primary copy of the original string is unreadable. Data are not duplicated during the merge phase, and error recovery is accomplished by backtracking to remerge previously merged data. In no case, is error recovery attempted beyond one level of recovery. (If recovery is attempted while recovering from a prior error, SORT terminates.) A discussion of the primary areas of error recovery is presented in the following paragraphs.

Error Recovery of Control File Input Errors

An attempt is made to obtain the record by rereading the "error" record several times. If the error record is unreadable and error recovery is not requested, the SORT terminates. If error recovery is requested, the SORT attempts to read its duplicate copy of the error record.

Error Recovery of Control File Output Errors

An attempt is made to successfully write the error record. If writing is not successful after several retries and error recovery is not requested, SORT terminates. If error recovery is requested, SORT marks as bad disk (XD) the row of disk containing the error record. SORT retains the other copy of the XDed row for subsequent use. If possible, SORT continues in full error recovery mode; otherwise, SORT displays SORT ERROR 31 and continues in error recovery mode for the workfile. Further error recovery for the control file is no longer possible. In either case, if SORT is unable to write the error record after the bad row has been XDed, SORT terminates. Only one disk row is marked as bad disk on an error so that it is not possible to get into a loop and XD large quantities of disk.

Error Recovery of Workfile Input Errors

An attempt is made to obtain the record by rereading the error record several times. If the error record is unreadable and error recovery is not requested, SORT terminates. If error recovery is requested and the data have been duplicated, an attempt is made to read the duplicate copy. If the error record was written by the merge phase and no duplicate copy exists, SORT attempts to recreate the string of information containing the error record. Before backtracking to the previous merge, SORT writes and reads a test record in the error record location. If the test is unsuccessful, SORT marks as bad disk the row of disk containing the error record location. After testing and possible XDing of disk, SORT backtracks to the desired point for restarting the merge phase. At most, one row of disk is marked as bad disk for each occurrence of an input error for the workfile.

Error Recovery of Workfile Output Errors

An attempt is made to successfully write the error record. If writing is not successful after several retries and error recovery is not requested, SORT terminates. If error recovery is requested, SORT marks the row of disk containing the error record as bad disk. If possible, SORT continues in full error recovery mode; otherwise, SORT displays SORT ERROR 32 and continues with error recovery reset. If the SORT is in the stringing phase, an attempt is made to write the error record and if the attempt is unsuccessful, SORT terminates. If SORT is in the merge phase, it backtracks to the desired point of restarting the merge phase. At most, one row of disk is XDed for each occurrence of an output error for the workfile.

Error Recovery of User Output File Errors

When the program has given the SORT an output file (rather than an OUTPUT PROCEDURE), the SORT closes and purges the output file and restarts the output from the first output record. If the output file is a disk file and insufficient space was allocated to contain the data, the SORT either: (1) sets the FLEXIBLE attribute before restarting the output, or (2) if setting the FLEXIBLE attribute is not possible, terminates with SORT ERROR 8. Output error recovery is not dependent on program request for error recovery.

Error Recovery of Workfile Input-Errors during User Output

When the user's output is a file, the file is closed and purged, and SORT attempts to remerge the desired string. If the output is a procedure and error recovery is specified, SORT repositions itself to remerge the desired string and subsequently terminates with SORT ERROR 19. When SORT is restarted, it remerges the desired string and starts user output with the first output record.

11. MISCELLANEOUS INTERNAL INFORMATION

This section contains information about the SORT disk files and memory allocation as well as miscellaneous information concerning restart and error recovery.

SORT DISK FILES

SORT uses two disk files (when disk or ITD sorts are requested) for storing the data and control records.

The control file is normally a very small disk file whose size is based on the maximum number of strings which can be produced for the sort currently being executed.

Control file records are three words; blocks are 90 words. The maximum number of control file rows is 64, and the number of physical blocks per row is four (unless the amount of disk provided is extremely large). The last two rows of the control file contain restart information when restartable sorts are requested. The internal name for the control file is DISKC and the title is SORT/DISKC. When a restartable sort is desired, file attribute equation should be used to give a unique title to the control file. An example is:

```
<I> FILE DISKC (TITLE=JOBNAME/SORTCONTROL)
```

Other attributes that may be set by use of file attribute equation are limited to assignment of the control file to disk pack. Use of other attributes is not prohibited, but the SORT cannot function unless extreme care is exercised. A high probability of failure exists if attributes such as AREAS, AREASIZE, MAXRECSIZE, BLOCKSIZE, and so forth, are modified by file attribute equation.

The workfile is used by SORT to contain the data or records being sorted. Workfile size is provided explicitly or implicitly by the user program. SORT first determines a desired blocksize and then computes the number of disk rows provided by the user. The maximum number of rows SORT allocates to the workfile is 183; partial rows are rounded up to a full row. Row sizes do not exceed 1,320 segments unless an extremely large amount of disk is provided. The internal name for the workfile is DISKF and the title is SORT/DISKF. When a restartable sort is requested, file attribute equation should be used to give a unique title to the workfile. An example is

```
<I> FILE DISKF (TITLE=JOBNAME/SORTWORK)
```

Other attributes that may be set by use of file attribute equation are limited to assignment of the workfile to disk pack and the ability to provide a new BLOCKSIZE and MAXRECSIZE. Use of other attributes is not prohibited, but the SORT cannot function unless extreme care is exercised. A high probability of failure exists when attributes such as AREAS, AREASIZE, and so forth, are modified by file attribute equation.

SORT recognizes changes in BLOCKSIZE. However, BLOCKSIZE and MAXRECSIZE must agree in order to open the file. The ability to modify this value is provided as a means of overriding the normal memory allocation algorithms of SORT. However, care should be exercised in the use of this ability to modify the buffer size of the SORT. When the buffer size is increased, the number of disk segments per disk row is proportionately increased, and SORT proceeds using the larger disk rows. If the buffer size is decreased, the number of disk segments per disk row is proportionately decreased, which may result in a workfile not large enough to accomplish the sort. One method of alleviating this condition

is to specify a larger quantity of disk. Modification of the buffer size is usually less effective than providing a different memory size for use by SORT. However, sorts are always data dependent, and with some ordering of data, a better sort may possibly be obtained by judicious selection of buffer size.

SORT MEMORY ALLOCATION

SORT attempts to stay within the memory estimate provided by the user even to the extent of sorting with only six records in memory. Memory allocation proceeds (in general) through the following steps:

1. Memory size provided by the program is reduced by 1,500 words. The reduced size is used for all subsequent calculations. The reduction is a generous estimate of the amount of space required for working storage and the space required for various SORT procedures.

2. A buffer size is selected for the internal disk and/or tape files. SORT tries to select buffer sizes so that it does not become I/O bound. For disk sorting, SORT normally allocates two buffers per string. For tape sorting with N tapes, SORT allocates 1/Nth of memory as buffers for each tape.

3. During executions of the stringing phase, two output buffers are normally allocated; thus, the remainder of memory is left for the sort vector. During execution of the merge phase, virtually all available memory is used to contain buffers.

MISCELLANEOUS INFORMATION

A number of conditions exist in SORT for restart and error recovery. Some of these conditions are documented elsewhere; however, the following text lists the most salient.

1. The sort vector size limit is 65,535 and is not required to be a power of two.

2. The order of merge limit is 65,535.

3. The limit for a disk string is 549,775,813,887 records, and the maximum number of blocks that can be contained in the sort workfile is also 549,775,813,887. The tape limit allows an unlimited string length but limits the number of blocks that can be obtained on any single worktape to between 2,097,151 and 4,294,967,295 depending on the number of records per block. (Reel switches may or may not occur; however, this discussion assumes a large tape reel.) The ultimate limit for tape sorting is the stated tape limit times the number of sort work tapes specified.

4. The file title for sort work tapes is SORTATAPEn (where n is a number between zero and seven). This feature eliminates the necessity for operator intervention to resolve DUP FILE messages.

5. When SORT has been given some work disk to use, it attempts to recognize the condition where the input data is less than 40 percent in sequence and switches comparison from ascending to descending or vice versa. SORT remembers the change of mode and processes the data accordingly. The switch is done as often as necessary in order to produce longer strings. Given a set of input data in exact reverse sequence, SORT produces two strings (rather than the maximum use of strings) and completes the sort much faster.

6. Restart capability can be excluded from SORT by resetting the RESTART option in the SORT portion of the MCP symbolic and by recompiling the MCP. Omitting the code associated with RESTART/ERROR recovery results in sorts that run between 2 and 15 percent faster with an average improvement of 6 to 10 percent. Disk and ITD sorts are the only sorts measurably improved by this omission.

7. Disk buffer size and disk record addressing have been specifically chosen to reduce disk latency for the sort work disk. Whether the SORT is I/O bound or compute bound is also dependent on many other associated factors.

7. SYSTEM/GUARDFILE INPUT EXAMPLE

An example SYSTEM/GUARDFILE input request for a database is shown below. The output from the SYSTEM/GUARDFILE program, indicating the contents of the guardfile created by this deck, is also shown.

```
% CARD DECK FOR TESTING DATA MANAGEMENT SECURITY FUNCTIONS

DEFAULT = NO; % WE WANT THIS TO BE A PRIVATE DATABASE -
              % ONLY THOSE PROGRAMS AND USERCODES ACTUALLY
              % IN THE GUARDFILE MAY ACCESS THE DATABASE

DEFINE OK = ALL EXCEPT (CLOSELOCK);  % DONT WANT TO OVER-WRITE DATABASE

PACKNAME = DMPACK;  % ONLY PROGRAMS RUNNING FROM HERE CAN ACCESS D-BASE

USERCODE STEWART=RW,DMVERBS=OK % UNLESS USING ONE OF THE FOLLOWING PROG.
   USING PROGRAM

   [ OBJECT/FIND            =RW, DMVERBS=ALL EXCEPT (FIND),
     OBJECT/LOCK            =RW, DMVERBS=ALL EXCEPT (LOCK),
     OBJECT/OPENINQUIRY     =RW, DMVERBS=ALL EXCEPT (OPENINQUIRY),
     OBJECT/ASSIGN          =RW, DMVERBS=ALL EXCEPT (ASSIGN),
     OBJECT/CREATESTORE     =RW, DMVERBS=ALL EXCEPT (CREATESTORE),
     OBJECT/DELETE          =RW, DMVERBS=ALL EXCEPT (DELETE),
     OBJECT/LOCKSTORE       =RW, DMVERBS=ALL EXCEPT (LOCKSTORE),
     OBJECT/REMOVE          =RW, DMVERBS=ALL EXCEPT (REMOVE),
     OBJECT/OPENUPDATE      =RW, DMVERBS=ALL EXCEPT (OPENUPDATE),
     OBJECT/CLOSELOCK       =RW, DMVERBS=ALL EXCEPT (CLOSELOCK),
     OBJECT/OPENINITIALIZE=RW, DMVERBS=ALL EXCEPT (OPENINITIALIZE),
     OBJECT/OPENTEMPORARY =RW, DMVERBS=ALL EXCEPT (OPENTEMPORARY),
     OBJECT/GENERATE        =RW, DMVERBS=ALL EXCEPT (GENERATE)
   ];

PROGRAM TESTDEFINE ON TESTPACK = RW, DMVERBS=OK EXCEPT(GENERATE);

PROGRAM A,B,C=RO;

PROGRAM (USR)X/Z = NO; % DONT LET HIM IN AT ALL

PROGRAM *A/B = RW,DMVERBS=OK EXCEPT (INSERT);

PROGRAM "HYPHEN-ATED" = RO;

PROGRAM "USING" = RO USING USERCODE "USING" = RW;

        PROGRAM THIS/IS/A/NAME/WHICH/IS/TOO/BIG/TO/PRINT/
        ON/A/SINGLE/LINE = RW DMVERBS=ALL EXCEPT (OPENINITIALIZE);
```

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

8. SYSTEM/GUARDFILE OUTPUT EXAMPLE

```
GUARDFILE VERSION 002 DEFAULT ACCESS=NO PACKNAME DMPACK

USING USERCODE STEWART  = RW
               DMVERBS: ASSIGN CREATESTORE DELETE FIND GENERATE INSERT
                        LOCK LOCKSTORE OPENINITIALIZE OPENINQUIRY
                        OPENTEMPORARY OPENUPDATE REMOVE

     USING PROGRAM  (STEWART)OBJECT/FIND ON DMPACK = RW
                    DMVERBS: ASSIGN CLOSELOCK CREATESTORE DELETE
                             GENERATE INSERT LOCK LOCKSTORE
                             OPENINITIALIZE OPENINQUIRY OPENTEMPORARY
                             OPENUPDATE REMOVE

     USING PROGRAM  (STEWART)OBJECT/LOCK ON DMPACK = RW
                    DMVERBS: ASSIGN CLOSELOCK CREATESTORE DELETE FIND
                             GENERATE INSERT LOCKSTORE OPENINITIALIZE
                             OPENINQUIRY OPENTEMPORARY OPENUPDATE
                             REMOVE

     USING PROGRAM  (STEWART)OBJECT/OPENINQUIRY ON DMPACK = RW
                    DMVERBS: ASSIGN CLOSELOCK CREATESTORE DELETE FIND
                             GENERATE INSERT LOCK LOCKSTORE
                             OPENINITIALIZE OPENTEMPORARY OPENUPDATE
                             REMOVE

     USING PROGRAM  (STEWART)OBJECT/ASSIGN ON DMPACK = RW
                    DMVERBS: CLOSELOCK CREATESTORE DELETE FIND
                             GENERATE INSERT LOCK LOCKSTORE
                             OPENINITIALIZE OPENINQUIRY OPENTEMPORARY
                             OPENUPDATE REMOVE

     USING PROGRAM  (STEWART)OBJECT/CREATESTORE ON DMPACK = RW
                    DMVERBS: ASSIGN CLOSELOCK DELETE FIND GENERATE
                             INSERT LOCK LOCKSTORE OPENINITIALIZE
                             OPENINQUIRY OPENTEMPORARY OPENUPDATE
                             REMOVE

     USING PROGRAM  (STEWART)OBJECT/DELETE ON DMPACK = RW
                    DMVERBS: ASSIGN CLOSELOCK CREATESTORE FIND
                             GENERATE INSERT LOCK LOCKSTORE
                             OPENINITIALIZE OPENINQUIRY OPENTEMPORARY
                             OPENUPDATE REMOVE

     USING PROGRAM  (STEWART)OBJECT/LOCKSTORE ON DMPACK = RW
                    DMVERBS: ASSIGN CLOSELOCK CREATESTORE DELETE FIND
                             GENERATE INSERT LOCK OPENINITIALIZE
                             OPENINQUIRY OPENTEMPORARY OPENUPDATE
                             REMOVE

     USING PROGRAM  (STEWART)OBJECT/REMOVE ON DMPACK = RW
                    DMVERBS: ASSIGN CLOSELOCK CREATESTORE DELETE FIND
                             GENERATE INSERT LOCK LOCKSTORE
                             OPENINITIALIZE OPENINQUIRY OPENTEMPORARY
                             OPENUPDATE
     USING PROGRAM  (STEWART)OBJECT/OPENUPDATE ON DMPACK = RW
                    DMVERBS: ASSIGN CLOSELOCK CREATESTORE DELETE FIND
                             GENERATE INSERT LOCK LOCKSTORE
                             OPENINITIALIZE OPENINQUIRY OPENTEMPORARY
                             REMOVE
```

```
     USING PROGRAM  (STEWART)OBJECT/CLOSELOCK ON DMPACK = RW
                    DMVERBS: ASSIGN CREATESTORE DELETE FIND GENERATE
                             INSERT LOCK LOCKSTORE OPENINITIALIZE
                             OPENINQUIRY OPENTEMPORARY OPENUPDATE
                             REMOVE

     USING PROGRAM  (STEWART)OBJECT/OPENINITIALIZE ON DMPACK = RW
                    DMVERBS: ASSIGN CLOSELOCK CREATESTORE DELETE FIND
                             GENERATE INSERT LOCK LOCKSTORE
                             OPENINQUIRY OPENTEMPORARY OPENUPDATE
                             REMOVE

     USING PROGRAM  (STEWART)OBJECT/OPENTEMPORARY ON DMPACK = RW
                    DMVERBS: ASSIGN CLOSELOCK CREATESTORE DELETE FIND
                             GENERATE INSERT LOCK LOCKSTORE
                             OPENINITIALIZE OPENINQUIRY OPENUPDATE
                             REMOVE

     USING PROGRAM  (STEWART)OBJECT/GENERATE ON DMPACK = RW
                    DMVERBS: ASSIGN CLOSELOCK CREATESTORE DELETE FIND
                             INSERT LOCK LOCKSTORE OPENINITIALIZE
                             OPENINQUIRY OPENTEMPORARY OPENUPDATE
                             REMOVE
USING PROGRAM  (STEWART)TESTDEFINE ON TESTPACK = RW
               DMVERBS: ASSIGN CREATESTORE DELETE FIND INSERT LOCK
                        LOCKSTORE OPENINITIALIZE OPENINQUIRY
                        OPENTEMPORARY OPENUPDATE REMOVE

USING PROGRAM  (STEWART)A ON DMPACK = RO

USING PROGRAM  (STEWART)B ON DMPACK = RO

USING PROGRAM  (STEWART)C ON DMPACK = RO

USING PROGRAM  (USR)X/Z ON DMPACK = NG

USING PROGRAM  *A/B ON DMPACK = RW
               DMVERBS: ASSIGN CREATESTORE DELETE FIND GENERATE LOCK
                        LOCKSTORE OPENINITIALIZE OPENINQUIRY
                        OPENTEMPORARY OPENUPDATE REMOVE

USING PROGRAM  (STEWART)"HYPHEN-ATED" ON DMPACK = RO

USING PROGRAM  (STEWART)USING ON DMPACK = RO
     USING USERCODE USING  = RW

USING PROGRAM  (STEWART)THIS/IS/A/NAME/WHICH/
IS/TOO/BIG/TO/PRINT/ON/A/SINGLE/LINE ON DMPACK = RW
               DMVERBS: ASSIGN CLOSELOCK CREATESTORE DELETE FIND
                        GENERATE INSERT LOCK LOCKSTORE OPENINQUIRY
                        OPENTEMPORARY OPENUPDATE REMOVE
```

TABLE OF CONTENTS

1. INTRODUCTION

SYSTEM/GUARDFILE is a utility program that creates guardfiles. A guardfile describes the access rights of various users and programs to a program or data file or a database. These are collectively referred to as "structure" in this section. When access to a structure is controlled by a guardfile, every attempt to OPEN that structure causes the MCP to examine the access rules in the guardfile before granting or denying access to the structure.

The SYSTEM/GUARDFILE utility creates a guardfile but does not attach it to a structure. To "guard" a structure, one of the following steps must be taken.

Guarding Programs And Data Files

Programs and data files may specify access rights by using either file attributes or a WFL SECURITY statement:

```
FILE  X(SECURITYTYPE=GUARDED,
         SECURITYGUARD=MY/GUARD ON Y)

FILE  (SECURITYTYPE=CONTROLLED,
        SECURITYGUARD=MY/GUARD/ON Y)

                  or

SECURITY X GUARDED MY/GUARD ON Y

SECURITY CONTROLLED MY/GUARD ON Y
```

Guarding Data Bases

To guard an entire database, make the ACCESSROUTINES PUBLIC and apply the DASDL GUARDFILE construct to the database name.

For a database called MYDB:

```
MYDB (GUARDFILE="MY/GUARD ON Y")
```

To apply different access rights to different structures within the database, use the DASDL GUARDFILE construct to attach the appropriate guardfile to a logical database:

```
LDB DATABASE (. . .) GUARDFILE="MY/GUARD ON Y"
```

Refer to the B 7000/B 6000 Series DMSII DASDL Reference Manual, form 5001480, for a more detailed explanation of DASDL syntax and semantics.

NOTES

1. The guardfile may be created either before or after it is attached to a structure; none of the above techniques reads the guardfile until the guarded structure is opened.

2. Family substitution does not apply to the search for the guardfile when a structure is opened. An ON clause is absolutely necessary if guardfile is not on DISK.

   If the guardfile is not prefixed by an asterisk, the usercode directory of the accessed structure (not the user accessing the structure) is searched; if the guardfile is not found there, the system directory is searched.

   If the guardfile specified is not found or the guardfile is malformed, the file is treated as if SECURITYTYPE=PRIVATE.

3. The same guardfile may be attached to several files.

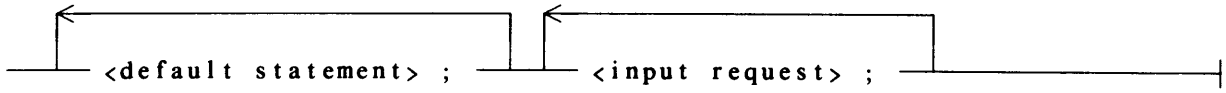
4. GUARDED is a synonym for CLASSB.

5. CONTROLLED is a synonym for CLASSC.

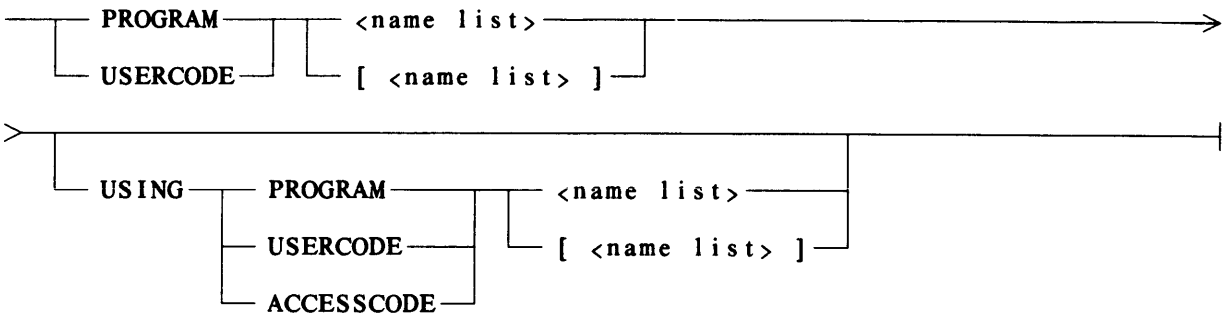
2. SYSTEM/GUARDFILE INPUT

Syntax

<guardfile input>

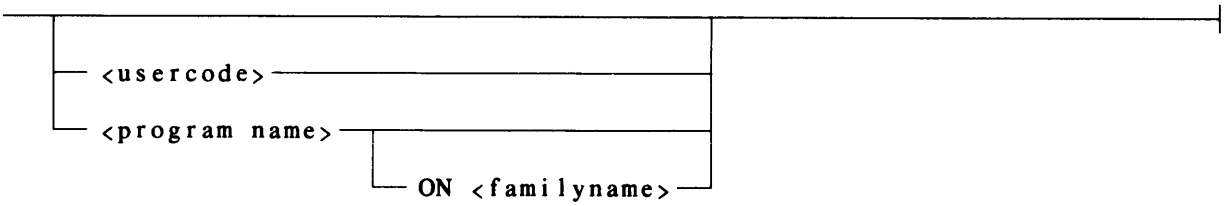


<input request>



<name list>

<simple name>



Semantics

The input to SYSTEM/GUARDFILE consists of a series of <input request>s describing in detail the access rights any user or program is to have. Unless otherwise indicated by a <default statement>, unmentioned users and programs have no access to the file.

The SYSTEM/GUARDFILE program assumes that the 80-character input record contains data pertinent to the usercode/program entries.

SYSTEM/GUARDFILE does not accept sequenced input.

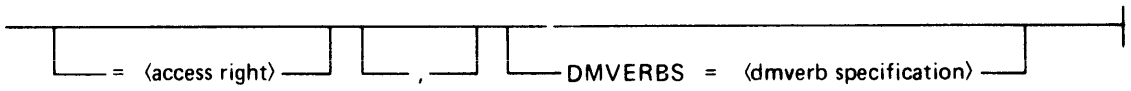
A percent sign terminates scanning of an input record. When a percent sign is encountered by the program, all remaining information on the input record to the right of the percent sign is ignored by the program.

Each <input request> describes the access rights for one or more users or one or more programs. If two input requests apply to an attempted access, the first request is used. If no default statement is used, no access is permitted to users and programs not listed in the guardfile.

<access specification>

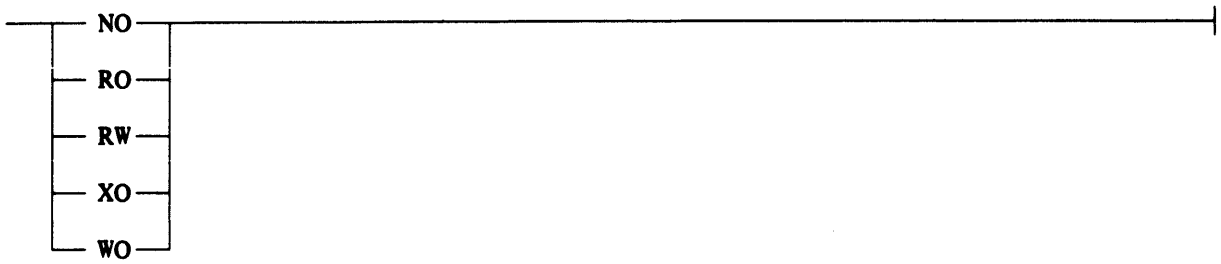
Syntax

<access specification>



MV1780

<access right>



Semantics

The access specification consists of two basic entities: access rights and dmverbs. The following describe access rights:

<empty>	Assumes DEFAULT value
NO	No access permitted
RO	Read-only access permitted
RW	Read-write access permitted
XO	Execute-only access permitted for code files
WO	Write-only access permitted

Examples

```
USERCODE A = RO;
USERCODE B = RW;
```

User A may read but not write the file.

User B may read and write the file.

Other users may not access the file.

```
USERCODE A = RO;
PROGRAM X = RW;
```

All users other than A may both read and write the file if and only if they are executing program X.

Usercode A may only read the file even when running X; the first card prevails.

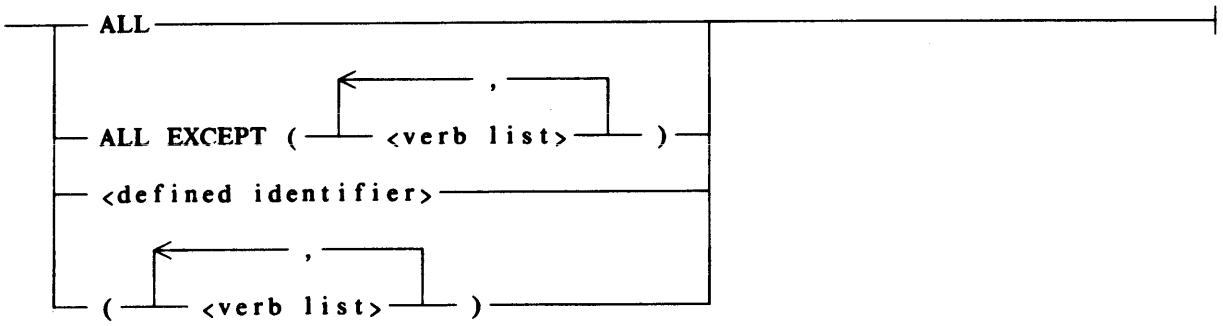
```
USERCODE A = RO;
```

User A may only read the file.

<dmverb specification>

Syntax

<dmverb specification>



<verb list>

Semantics

The following apply only to DMS II DATABASEs, not to ordinary files:

<empty>	Assumes DEFAULT values
ALL	All verbs permitted
ALL EXCEPT (<verb list>)	All verbs permitted except those in <verb list>
<defined identifier>	Uses specified values
<verb list>	Specified verbs allowed or not allowed

For a database, the data management operations to be allowed any user or program may be specified. Table 12-2-1 illustrates the dmverbs that may be specified along with certain default values associated with dmverb specifications and access rights. Any item in the first column of Table 12-2-1 can be used for the <verb> in <dmverb specification>.

Certain combinations of access rights and dmverb specifications are not permitted. This limitation is because an access right of read-only implies that the database may be opened for inquiry only (no modification of the database is permitted). Thus, some verbs (such as STORE, REMOVE, and so forth) are not permitted. Therefore, if an access right of read-only is assigned to an entry and a dmverb specification that includes verbs not in the standard read-only verb list (see Table 12-2-1) is requested, an error is given.

For data management usage, the access rights execute-only and write-only have no meaning. Access rights of execute-only and write-only are interpreted to mean no access.

Table 12-2-1.

Dmverbs and their Default Value Lists

Dmverb	Verb List 1 Default for Read-Only	Verb List 2 Default for Read-Write	Verb List 3 List for ALL
FIND	X	X	X
LOCK	X	X	X
OPENINQUIRY	X	X	X
OPENUPDATE		X	X
ASSIGN		X	X
DELETE		X	X
GENERATE		X	X
INSERT		X	X
REMOVE		X	X
CREATESTORE		X	X
LOCKSTORE		X	X
OPENINITIALIZE			X

NOTES

When an access right of read-only is specified for an entry, the database may only be opened INQUIRY.

Dmverbs not shown in this table are not available for security purposes. Verbs such as FREE, CREATE, and so forth, are not secured since their use does not alter the database.

Examples

```
USERCODE A = RW, DMVERBS = ALL;
```

User A may use all data management operations.

```
USERCODE A = RO, DMVERBS = (OPENINQUIRY,FIND);
```

User A may open the database for inquiry only, and after it has been opened, may only execute a FIND (all other DMVERBS are disallowed).

```
USERCODE A = NO, DMVERBS = (FIND);
```

Since the access right NO does not apply to data management, a syntax error is given.

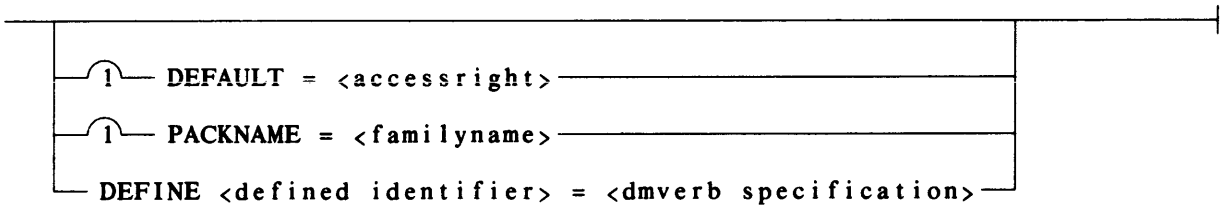
```
USERCODE A = RO, DMVERBS = (DELETE);
```

Since the DELETE operation is not consistent with the access right RO, a syntax error is given.

Default and Define Specifications

Syntax

<default statement>



Semantics

The SYSTEM/GUARDFILE program accepts default values for access rights, packnames, and dmverb specifications. The defaults and definitions, if present, must precede any program or usercode input specifications.

The DEFAULT access right applies to any program or usercode not named in the guardfile as well as to individual entries for which no access specification is given. If the MCP examines the guardfile and does not find an entry corresponding to the program and/or usercode attempting to access the guarded file, the DEFAULT access specified is assumed.

The PACKNAME default specifies the family name to be used for every program named in the input without an ON <familyname> clause.

The DEFINED dmverb specifications may be used in conjunction with the program and/or usercode entries that follow them. These are a shorthand method of spelling the data management verb lists. After the list is given once (in the DEFINE statement), the "definition" can be used to represent those verbs thereafter.

If no applicable <default statement> is presented to the SYSTEM/GUARDFILE program, then the following default values are assumed:

The default access right is no access.

The default dmverb specification is either verb list 1 or verb list 2, depending on the access right assigned to the entry.

The default pack name is DISK.

Using Clause

Frequently, combinations of programs and usercodes must be specified when specifying file security. For example, certain programs may be coded such that one user may use them for reading a file or searching a database, while another user might be allowed to update the file or database with the program. Also, a particular usercode may be allowed access only via a particular program. The USING clause is used to indicate these combinations.

In all cases where conflicting <input request>s are given, the first is the one used.

File Examples

```
USERCODE A = RO;
USERCODE B USING PROGRAM SAFEPROGRAM = RW;
```

No user other than A or B may access the file in any way.

User A may only read the file.

User B may read or write the file, but only using SAFEPROGRAM.

```
DEFAULT = XO;
```

Any user may execute the program (this is operationally equivalent to having the security of the file PUBLIC SECURED).

```
PROGRAM X = RO, USING USERCODE A = RW;
```

No user may access the file except through program X.

Only A may use X to both read and write the files; other users are restricted to reading.

```
DEFAULT = NO;
USERCODE C USING ACCESSCODE D = RW;
```

Only users running under usercode C, who know the accesscode D (and corresponding password), have read/write access to the CONTROLLED file. All other users are denied access.

Database Examples

```
DEFAULT = RO;
DEFINE STANDARD = ALL EXCEPT (OPENINITIALIZE,GENERATE,CLOSELOCK);
PROGRAM DB/SEARCH=RO USING USERCODE USR = RW,DMVERBS=STANDARD;
```

Any user with a usercode other than USR (or running without a usercode) can only access the database via program DB/SEARCH (or any other program) on a read-only basis. Since no dmverb specification was associated with the program, verb list 1 (of Table 12-2-1) is assumed by default.

Program DB/SEARCH, when executed under usercode USR may access the database on a read-write basis using the dmverb list specified as STANDARD in the SYSTEM/GUARDFILE default statement.

```
PROGRAM DB/SEARCH=RW,DMVERBS=ALL USING USERCODE USR=RO;
```

Program DB/SEARCH may access the database on a read-write basis, using any and all dmverbs, unless the program is being executed by user USR; in that case, only read access is permitted. Since no dmverb specification appears in the modifier, the default read-only verb list is assumed for user USR.

```
USERCODE USR=NO USING PROGRAM DB/SEARCH=RO;
```

User USR is not permitted access to the database unless it is via the program DB/SEARCH.

When program DB/SEARCH is executed by user USR, the data base may be accessed on a read-only basis (open inquiry). Use of any verbs other than those specified in verb list 1 of Table 12-2-1 is not permitted.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

3. MULTIPLE PROGRAM NAMES AND/OR USERCODES

Programs may be grouped into classes by function. For example, a group of programs whose sole function is to search a file or database and provide reports based on current information might be considered as "inquiry" programs. Other programs may periodically update low security items within the file or database, and still others may manipulate restricted information.

The SYSTEM/GUARDFILE program permits programs and/or usercodes to be grouped together to provide a shorthand method of specifying security.

The programs may be listed either with or without brackets. The access specification applies to all preceding items in the list, whether bracketed or not, which have no access specification of their own. The USING clause applies only to the immediately-preceding item or bracketed list of items.

Each name in the list is followed by a comma. An access specification may be specified for each name in the list or may be omitted.

If no access specification is supplied for any name in the list, the DEFAULT values of access rights and dmverbs are used.

If access specifications are stated for some of the names in the list, then one of the following applies:

1. The access specification stated for the current name being processed is applied to all previous names in the list for which no access specification was stated. The list is processed backwards until a name for which an access specification was stated is found or until the beginning of the list is encountered.

2. If any portion of the access specification is stated, the access specification is treated as complete for the purpose of insertion into previous elements of the name list. Therefore, if either the access right or the dmverb specification is stated for an item in the name list, subsequent access specifications are not applied to that item.

Examples

```
USERCODE [A,B,C = XO];
```

A,B and C all have execute-only access rights.

```
USERCODE [A = RO, B,C = RW];
```

A has read-only rights; B and C have read and write access.

```
USERCODE [U,V] USING PROGRAM [X,Y = RO, Z = RW];
```

U and V have the same access rights. They may have read-only access via programs X and Y or read-write access via program Z. Otherwise, they have no access.

```
PROGRAM [ A/B,A/C=RO,A/D,A/E,A/F=RW,DMVERBS=ALL ];
```

In the above example, programs A/B and A/C are assigned an access right of read-only. A/B gets a read-only access right because it precedes A/C for which an access right is specified, and because no access right is specified for A/B. Programs A/B and A/C have default dmverb specifications corresponding to verb list 1 of Table 12-2-1 because no dmverb specification is stated.

Similarly, programs A/D, A/E, and A/F have read-write access rights and have dmverb specifications of ALL.

Program A/B is not reassigned an access right of read-write because of the intervening specification for program A/C.

Also, program A/C is not assigned a dmverb specification of ALL even though the dmverb option is omitted from the access specification of A/C.

```
PROGRAM [A/B,A/C,A/D=RO] USING USERCODE
[USR1,USR2=RW,DMVERBS=STANDARD];
```

This example shows a bracketed name list used for usercodes and program names. Such a list may appear in either the primary entry or the USING clause or both. Programs A/B, A/C, and A/D may access the database on a read-only basis unless they are executed with users USR1 or USR2, in which case they may access the database on a read-write basis.

4. PACKNAMES

A family name may be used when specifying the names of program entries in the guardfile. For example:

```
PROGRAM A/B ON PCK = RO;
```

includes the pack name PCK as part of the program name entry. The MCP employs the following rules governing pack names:

1. A family name of DISK (A/B ON DISK) is equivalent to having no family name at all. Thus A/B ON DISK is interpreted as simply A/B.

2. If the program entry in the guardfile has a pack name other than DISK, then any programs that initiate the guardfile search must reside on the specified pack. Thus, if the guardfile entry specifies program A/B ON PCK and the actual program running is A/B ON OTHERPCK, no access is permitted.

3. If the program entry in the guardfile does not have a pack name (or the pack name is DISK), then the program that initiates the guardfile search may reside on any media. Thus, if the guardfile entry specifies program A/B, and the actual program running is A/B ON OTHERPCK, the access permitted is that specified in the guardfile.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

5. USERCODES

The SYSTEM/GUARDFILE program automatically appends the usercode under which it is executed to any program names which are specified in the input request. The user may, however, specify a system file or another usercode in the input request when necessary.

For example, the input requests:

```
PROGRAM (USR1)A/B = RO;
PROGRAM *A/B = RW;
```

may be specified.

At run time, the MCP compares the security information in the guardfile with the program name and ensures that they are the same before granting access. This procedure means that:

1. A guardfile entry without a usercode, when compared with a program name with a usercode, results in access being denied.

2. A guardfile entry with a usercode, when compared with a program name without a usercode, also causes access to be denied.

For example:

```
PROGRAM *A/B = RO;
```

creates this entry in the guardfile. When program *A/B is executed with or without a usercode, an access right of read-only is returned. When program (USR)A/B is executed, no access is permitted. Running program *A/B under usercode USR does not change the name of the program (the task attribute) to (USR)A/B;

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

6. RUNNING THE SYSTEM/GUARDFILE PROGRAM

SYSTEM/GUARDFILE has three files that may be label-equated:

CARD	The input file.
LINE	The printer file.
GUARD	The output guardfile. This file should be label-equated to have the desired title.

Example

```
?RUN SYSTEM/GUARDFILE
?FILE GUARD (TITLE=MY/GUARD ON XX)
?DATA CARD
<input cards>
?END
```

LISTING AN EXISTING GUARDFILE

The SYSTEM/GUARDFILE program may be used to list the contents of an existing guardfile. When the SYSTEM/GUARDFILE program is executed with a task value greater than 0, the LIST-ONLY option is invoked. The required guardfile should be label-equated to the desired file.

Example

```
?RUN SYSTEM/GUARDFILE;FILE GUARD=MYGUARDFILE ON MYPACK;VALUE=1;
```

DEBUGGING OPTIONS

A DEBUGGING toggle may be set in the SYSTEM/GUARDFILE program by executing the program with a taskvalue less than zero. Under the DEBUGGING option, the line number (in the SYSTEM/GUARDFILE program) at which an error is detected is shown along with the error message.

DEFINE IDENTIFIER

A DEFINE identifier may be any alphanumeric character string beginning with a non-digit which is not a reserved word in the SYSTEM/GUARDFILE program.

Reserved words are:

ACCESSCODE
ALL
DEFAULT
DEFINE
DMVERBS
EXCEPT
ON
NO
PACKNAME
PROGRAM
RO
RW
USERCODE
USING
WO
XO

DCSTATUS

TABLE OF CONTENTS

1. INTRODUCTION

SYSTEM/DCSTATUS is a DCALGOL program that makes use of the DCSYSTEMTABLES installation intrinsic to produce run-time "snapshots" of the Data Comm tables maintained by the MCP and the DCP.

No attempt will be made to interpret the results generated by the DCSTATUS program because understanding these results requires an understanding of NDL as well as at least a casual understanding of the DCP.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES.

2. GENERAL INFORMATION

DCSTATUS must be supplied with an option list which specifies those elements of the datacom subsystem which are to be analyzed. The options are in the following hierarchy: ALL, DCP, CLUSTER, LINE, STATION. Each higher order item is inclusive of all lower order items. For example, if CLUSTER is specified, the analysis is performed on all lines and stations on that cluster. The options TERMINAL, TABLES, MODEM, NETWORK, GRAPH, and FILE do not fit into this hierarchy.

Output from DCSTATUS is normally sent to the line printer (internal file name is LINE). The program will, however, detect if the file LINE has been label equated to KIND=REMOTE, as is the case for the DCSTATUS command in CANDE and the DP command in DIAGNOSTICMCS. In this case the output format is modified to fit a 72-character line width.

The DCSYSTEMTABLES intrinsic does not lock the various tables that it accesses. It is therefore possible that the contents of the tables may change while the intrinsic is accessing them. In addition, more than one call on the intrinsic is used to obtain the contents of all the various tables. Although infrequent, the influence of timing may occasionally cause the results produced by DCSTATUS to appear conflicting.

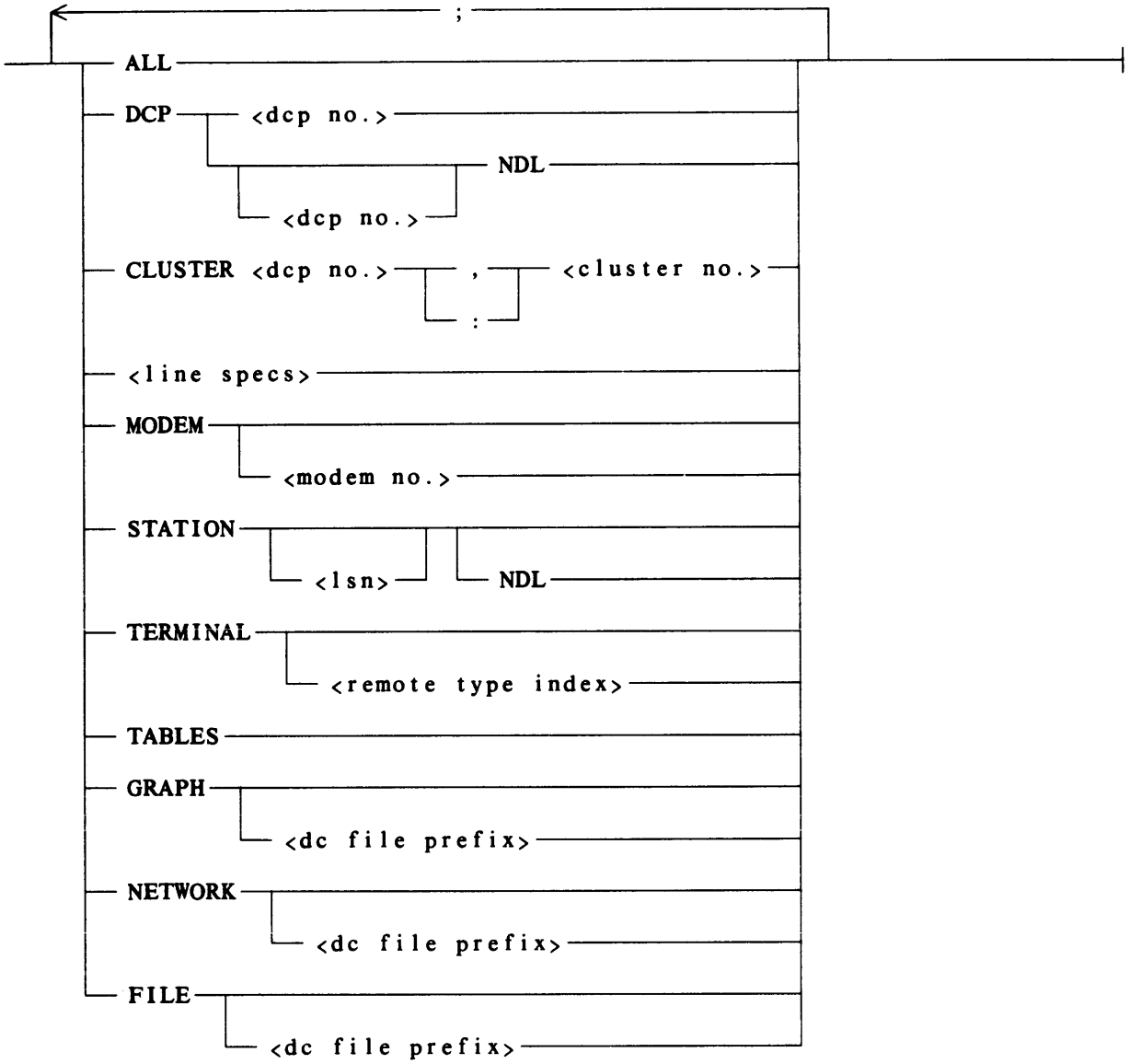
By use of the FILE statement, analysis can be performed on network definition files other than the currently active network files. Note that analysis of non-active network files precludes reporting information requiring active network files; that is, the following are the only allowable options for non-active network files:

DCP <dcp number> NDL
STATION <station number> NDL
TERMINAL
MODEM
GRAPH
NETWORK

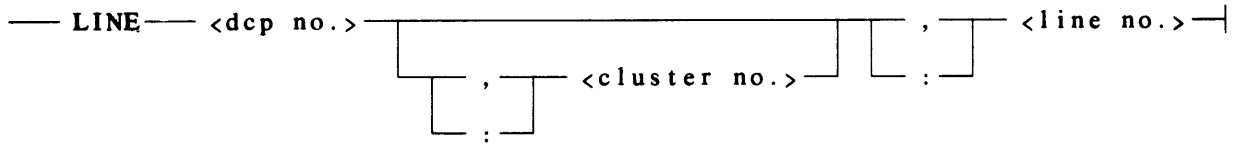
DCSTATUS

Syntax

<dcstatus options>

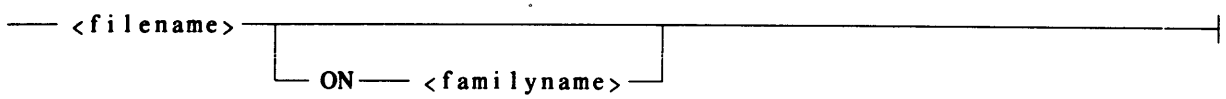


<line specs>



DCSTATUS

<dc file prefix>



13900

Semantics

Several options separated by semicolons may be specified for a single execution. The meaning of each option is as follows:

TABLES	Produces a raw hexadecimal dump of the DCC tables and the DCP line and station tables.
ALL	Produces a complete analysis of the datacom network. Analysis of the line and station tables together with an analysis of each remote type is performed. All other options are a subset of this option.
DCP	Produces an analysis of cluster, lines, and stations on all DCPs or a specific DCP. Use of the NDL option causes reporting to be based upon the information in the NIF and DCPCODE files instead of the current datacom tables.
CLUSTER	Produces an analysis of the lines and stations on the designated cluster.
LINE	Produces an analysis of the designated line and its stations.
MODEM	Produces an anlysis of modem information for a specific modem or for all modems defined in the network.
STATION	Produces a station analysis. If no LSN is specified, all stations will be analyzed. The normal sources of information for this option are the datacom tables in main memory or in DCP local memory. If the NDL option is specified then the sources of information are the NIF and DCPCODE files.
TERMINAL	Produces a listing of the NDL specifications of the terminals. The <remote type index> is the index used by the MCP to index into a table which describes each terminal specified in the NDL. Terminals are numbered in the sequence in which they appear in the NDL terminal definitions.

GRAPH	Produces a graph of the datacom network showing the relationship between the DCP'S, clusters, lines (names, addresses, and phone numbers for dial-in lines) and stations (names and LSNs). Since the graph information is obtained from the NIF and DCPCODE files, this option may be used whether datacom is running or not. If a <dc file prefix> is not specified, then the one currently being used by the system will be GRAPHED. ON <familyname> may be used in the <dc file prefix> specification.
NETWORK	Produces a brief tabular network configuration report. Information in the report includes: DCP, cluster, line, station, terminal, and MCS data.
FILE	Can be used to direct analysis at a non-active NIF and DCPCODE files. Since the graph information is obtained from the NIF and DCPCODE files, this option may be used whether datacom is running or not. If a <dc file prefix> is not specified then the one currently being used by the system will be used. ON <familyname> may be used in the <dc file prefix> specification.

3. EXECUTION

A typical WFL statement for execution of DCSTATUS may look like:

```
RUN SYSTEM/DCSTATUS(" <dcstatus options> ");
```

DCSTATUS may be executed remotely via the CANDE DCSTATUS command and the DIAGNOSTICMCS DP command. Refer to the documentation on these programs for syntax.

THIS PAGE IS INTENTIONALLY LEFT BLANK FOR FORMATTING PURPOSES

Documentation Evaluation Form

Title: B 7000/B 6000 Series System Software Operational Guide, Volume 1

Form No: 5011661

Date: June 1980

Burroughs Corporation is interested in receiving your comments and suggestions regarding this manual. Comments will be utilized in ensuing revisions to improve this manual.

Please check type of Suggestion:

☐ Addition ☐ Deletion ☐ Revision ☐ Error

Comments:

From:

Name _____

Title _____

Company _____

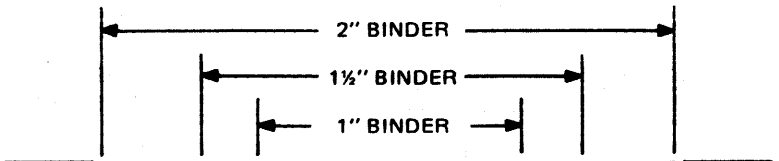
Address _____

_____

Phone Number _____ Date _____

Remove form and mail to:

Burroughs Corporation
Documentation Dept., TIO - West
P.O. Box 4040
El Monte, CA 91734



B 6000/B 7000 Series

SYSTEM SOFTWARE OPERATIONAL GUIDE

VOLUME 1

5011661

Printed in U.S.A.