PERKIN-ELMER

# OS/32 FASTCHEK

Reference Manual

48-064 F00 R00

The Perkin-Elmer Corporation, Data Systems Group, 2 Crescent Place, Oceanport, New Jersey 07757

TABLE OF CONTENTS

CHAPTERS (Continued)

# PREFACE

This manual describes the functions and features of the OS/32 FASTCHEK program, Part Number 03-344. OS/32 FASTCHEK is used to check the integrity of disc packs, and to initialize and rename packs in a fast, efficient and convenient manner.

## User Prerequisites

Users of OS/32 FASTCHEK should be familiar with the operation of OS/32.

## Synopsis of Chapters

Chapter 1 provides a general overview of the program's capabilities. Chapter 2 describes when to run FASTCHEK, how to load and start it.

Chapter 3 describes command entry, and Chapter 4 contains a description of the operation of FASTCHEK and also contains tuning information.

Chapter 5 discusses error handling and documents all the messages which can be generated. Internal failure conditions are discussed in Chapter 6.

Appendixes provide: a command summary; End of Task codes; device characteristics; format of the Pack Administration file; Link procedure; logical unit usage; a comparison with OS/32 DISCHECK and OS/32 DISCINIT; notes on the journal feature; notes on the compatibility with other products.

## System Requirements

OS/32 FASTCHEK executes as a segmented user task under OS/32 R06.2, or higher. R06.2 supports two new file types: nonbuffered indexed and extendable contiguous files.

OS/32 FASTCHEK functionally replaces the OS/32 DISCHECK and OS/32 DISCINIT programs. Current users of these products will find a comparison of FASTCHEK with DISCHECK and DISCINIT commands in Appendix G.

For information on the contents of all Perkin-Elmer 32-bit manuals, see the 32-Bit Systems User Documentation Summary.

# CHAPTER 1
# OS/32 FASTCHEK OVERVIEW

## 1.1 INTRODUCTION

In order to reliably store and retrieve information on a disc system certain procedures are necessary. These can be divided into two classes, those necessary to prepare the disc pack for use by the system, and those necessary to check the integrity of the pack and its contents.

Preparing a disc pack for use by the system involves two steps, 'formatting' and 'initializing'. The first of these is required so that the disc controller hardware can directly address each sector on the pack, and the second so that the operating system can allocate, write, and read data files on the pack. Thus 'formatting' is a hardware requirement, whereas 'initializing' is a software requirement.

When a pack is received from a manufacturer it is essentially 'blank' and has no information recorded on it and is referred to as unformatted. A hardware diagnostic/test program is then used to write sector headers on the pack and to check the surface of the pack. This process is termed 'formatting' and is often performed by an engineer.

The pack must then be 'initialized' by writing certain control information on the pack. One of the functions of FASTCHEK is to perform this initialization. The control information consists of the Volume Descriptor, the Directory and the Bit Map. The Volume Descriptor contains the name of the pack (the Volume Name), the addresses of the Bit Map and Directory, and certain other control information. It is always placed in the first sector on the pack (sector 0) so that the operating system can locate it. The Directory contains an entry for each data file currently on the pack. Each entry holds the name of the file, information on its type, its location on the pack, its status, and certain other control data. The Directory consists of a number of blocks linked together. As part of the initialization process, the directory can be 'pre-allocated' so as to optimise reading of the directory by the operating system. The Bit Map is used by the operating system to control the allocation of space on the pack. It contains one bit per sector on the pack with each bit being set to zero if the associated sector is free, and to one if the sector is either in use or defective.

During the intialization process, a check can be made to see if any of the sectors on the pack are defective. This operation is known as a 'surface check' or 'read check'. A sector is said to be defective if it either has been previously flagged as defective by the format program or cannot be sucessfully read by FASTCHEK. All defective sectors are flagged as being 'in use' in the Bit Map so as to prevent their use by the operating system.

Once the pack has been initialized it can be 'marked on' and made available for use.

If a software or hardware failure occurs, the integrity of the data on the pack must be assumed to be in doubt and FASTCHEK must be run before the pack can be returned to normal use. This operation is necessary for a number of reasons.

Firstly, it should be apparent that a hardware failure may result in erroneous data being written on the pack or may cause valid data to be written to the wrong place on the pack. Secondly, since the operating system buffers certain data in memory (to avoid the inefficiency of rewriting it to the pack whenever it is updated), it is highly probable that, following a system failure, the pack may contain inconsistent data. The data buffered in memory is always flushed when the pack is marked off. Thus, as a check that a pack being marked on has previously been marked off, the operating system maintains a control bit in the Volume Descriptor which is set whenever the pack is marked on as unprotected and reset when it is marked off. If this bit is found to be set when an attempt is made to mark the pack on as unprotected, it is taken as an indication that the integrity of the data on the pack is in doubt and the pack will not be marked on. FASTCHEK is used to perform the required checking and then, if the operation is successful, reset the bit so that the pack can be marked on for normal use. Specifically, FASTCHEK closes any files that were open (i.e. being accessed) at the time of the failure and warns of a possible data loss if the file was being written to. FASTCHEK also checks (where necessary) all the data linkage structures and when possible, repairs any that have been corrupted. Checks are also made for allocation conflicts. That is cases where, according to the existing control information, two or more files share the same sectors.

It is important to realise that when a system failure occurs and FASTCHEK is used to check and restore the integrity of the pack, FASTCHEK does nothing to fix the underlying cause of the failure, it simply attempts to restore the integrity of the pack. Thus for example, if the operating system detects a hardware error while accessing the Bit Map, it will request that the pack be marked off and checked. Running FASTCHEK in this case may restore the integrity of the pack (probably by relocating and rebuilding the Bit Map) but obviously cannot correct the original hardware fault.

## 1.2 GENERAL DESCRIPTION

FASTCHEK can be used to perform three different functions, initialize a formatted disc pack, rename a pack, and check the integrity of a pack.

The initialisation function can be performed in three different modes:

- with surface check
- with prefill and surface check
- without surface check

In FASTCHEK's terminology these modes are known as READCHECK, FILL, and NOREADCHECK respectively.

If the pack has just been formatted then one of the first two modes (READCHECK or FILL) must be used in order to check for and record the defective sectors. It should be noted that the prefill mode can be used either to prefill the pack with a recognizable 'never yet used' signature or (more usefully) to ensure that any previous data is overwritten thus performing a 'security erase' operation.

If the pack has been previously initialized then the NOREADCHECK mode can be used to save time. The surface check can be safely omitted since FASTCHEK records the defective sector information on the pack (in a file called the Pack Administration File) whenever a surface check is performed. If this file cannot be found on the pack then FASTCHEK automatically performs a surface check. Thus initialising a pack without a surface check is a fast and secure method for clearing all data files from the pack.

The rename function is used to change the name of a pack. The pack must have been previously initialized.

The check function is used to check the integrity of the pack after a system failure. The integrity of the pack is assured provided that all defective sectors are known, the directory contains valid information, the bit map accurately refects all unused sectors, all data files can be accessed, and no files share common sectors. Thus the integrity check can be divided into three phases, a basic check of the volume descriptor and the directory, a check of the file accessing paths and the file allocation, and a surface check of the pack. It will be apparent that whereas the directory check only takes a short time, a file check requires more time and a surface check even longer.

The integrity checking hierarchy is thus as follows:

- surface check then directory and file check
- directory and file check
- directory check

Since it is obviously time wasting to perform more checking than is actually required FASTCHEK allows the user to specify a given level of checking but will automatically perform more if found necessary. Thus the check function can be performed in four modes as follows:

- surface check then directory and file check
- directory and file check without prior surface check but surface check if found necessary
- directory check withcut file check but file check (and surface check) if necesary
- directory check only.

In FASTCHEK's terminology these modes are known as READCHECK, NOREADCHECK, CLOSE, and CLOSEONLY respectively.

Hence in normal circumstances it is most efficient to use the third mode, CLOSE, and rely on FASTCHEK to perform more extensive checking only if it is required. The fourth mode, CLOSEONLY, is used only in special applications where the user wishes to restore the integrity of the pack in a very short time if this is possible, but if it is not (because a file or surface check is required), he will use an alternate recovery stategy by, for example, switching to another system.

## 1.3 GENERAL FEATURES

FASTCHEK supports all disc devices currently supported by the operating system. It uses a highly efficient surface checking algorithm and an optimised file allocation checking technique. A full check of a 256 MB disc requires only approximately 7 minutes and only some two minutes if no surface check is required (provided sufficent memory is available). Smaller discs require correspondingly smaller times.

FASTCHEK is simple to use due to the provision of extensive defaulting of parameters and the automatic fallback to more extensive checking modes when required.

Commands can be entered either interactively (in response to a series of prompts), from a command file, or as start options. The interactive method is most convenient for inexperienced or occasional users.

FASTCHEK also provides the following options:
- an EXTENDALLOWED opticn to recover as much data as possible when performing the check function
- a WRITERECOVERY option to recover sectors which appear defective because of an erroneous Cyclic Redundancy Check
- a REPORTONLY option to prevent any updating during the check operation and only report on the integrity of the pack
- a KEEPSPOOL option to retain all spool files during a check operation

FASTCHEK creates and maintains a Pack Administration File named PACKINFO.DIR/0. This file contains a list of defective sectors on the pack and a record of the administrative history of the pack (i.e. when it was last checked, renamed, etc).

## 1.4 REQUIREMENTS

FASTCHEK requires a minimum of 40KB of user memory (32KB Pure, 8KB Impure) but will execute faster if more is available.

## 1.5 FILE TYPES

This revision of FASTCHEK introduces two new file types:

- Nonbuffered indexed (NB) files
- Extendable contiguous (EC) files

Both new file types are supported on all devices supported by OS/32 disc drivers. The attributes supported for both file types are: read, write, test and set, binary, wait, random, formatted, ASCII, unconditional proceed, image, rewind, backspace record, write filemark, forward space file, and backspace filemark. For details of these file types, see the OS/32 Application Level Programmer Reference Manual.

# CHAPTER 2
# GETTING STARTED

## 2.1 WHEN TO RUN FASTCHEK

FASTCHEK must be run:

- to initialize a newly formatted pack before attempting to use it

- to check the integrity of a pack after a system failure which occurred while the pack was marked on as unprotected

- to check the integrity of a pack on which the operating system has detected an error or logical inconsistency and has thus output one of the following messages:

```
BIT MAP ERROR ON voln:  MARK OFF AND CHECK

I/O ERROR ON voln:  MARK OFF AND CHECK
```

- to check the integrity of a pack which, when marked on, results in one of the following messages:

```
NOFF-ERR
PRI-DIR READ ERR
```

- NOTE: Drive must not be marked on in order to RUN Fastcheck - when system goes down with drives marked on.

FASTCHEK should be run:

- to check the integrity of a pack which, when marked on, results in one of the following messages:

```
SEC-DIR READ ERR
SEC-DIR WRIT ERR
```

- to check the integrity of a pack when a task detects one of the following errors while accessing a file on the pack:

```
Unrecoverable error (status code X'84')
End of Medium error (status code X'90') when reading
    from an Indexed file
```

FASTCHEK can be used:

- to rename (i.e. change the name of) a pack

- to clear all existing files from a pack

Users who do not have the time or inclination to study this manual can safely run FASTCHEK using the following START commands:

```
    START ,INIT=disc:,VOL=name

          to initialize the pack on drive "disc:" and
          set its volume name to "name"


    START ,CHECK=disc:

          to check the integrity of the pack on drive "disc:"


    START ,RENAME=disc:,VOL=name

          to rename the pack on drive "disc:" to "name"
```

However, it is strongly recommended that this manual be read so that the user understands the full range of options available.

**NOTE**

If the disc drive or controller hardware is suspected to be faulty, do not run FASTCHEK to check the integrity of the pack unless the REPORTONLY option is used. FASTCHEK might delete valid data from the pack because of errors due to faulty hardware.

## 2.2 LOADING FASTCHEK

### 2.2.1 Loading FASTCHEK from the System Console

This command loads FASTCHEK from the system console.

Format:

```
    LOAD  taskid , fd , segsize increment
```

Parameters:

| | |
|---|---|
| taskid | is a 1- to 8-character alphanumeric string specifying the name of the task after it is loaded into a foreground segment in main storage. |

| | |
|---|---|
| fd | is the file descriptor of the device or file containing the task image load module of FASTCHEK. If this parameter is omitted, the default is "taskid.TSK". |
| segsize increment | is a decimal number in kb specifying the additional main memory workspace to be added to the module's impure segment. This value, if given, overrides the "WORK=n" option which may have been specified when the task was linked.<br><br>NOTE<br><br>FASTCHEK will generally execute faster if it is allocated more memory. For information on the choice of optimum segment size increment see Chapter 4. |

## 2.2.2 Loading FASTCHEK from an MTM Terminal

This command loads FASTCHEK from an MTM terminal. (Note that because packs can only be marked on and off from the system console, FASTCHEK is not normally run from an MTM terminal although there is no operational reason why it cannot be.)

**Format:**

```
    LOAD  fd  ,segsize increment
```

**Parameters:**

| | |
|---|---|
| fd | is the file descriptor of the device or file which contains the task image load module of FASTCHEK. |
| segsize increment | is a decimal number in kb specifying the additional main memory workspace to be added to the module's impure segment. This value, if given, overrides the "WORK=n" option which may have been specified when the task was linked.<br><br>NOTE<br><br>FASTCHEK will generally execute faster if it is allccated more memory. For information on the choice of optimum segment size increment see Chapter 4. |

## 2.3 STARTING FASTCHEK

After FASTCHEK has been loaded, the START command may be used to begin execution of the program. The disk must be marked off, and Fastchek must be made the current task. The format of the START command is identical in both the OS/32 System Console and MTM environments.

The START command which should be used for FASTCHEK, has the general format:

```
    START [,param1 [,param2 [...[,paramn]...]]]
```

Depending on the parameters used in the START command, FASTCHEK will commence operation in one of three possible command entry modes: interactive, batch or immediate.

Note once FASTCHEK has validated all commands and is about to commence the requested operation, then, if the commands have been entered other than via the system console, the following message is logged to the console:

```
    OS/32 FASTCHEK Rnn-nn <function> disc: STARTING
```

where <function> is one of INITIALIZE, CHECK, or RENAME.

### 2.3.1 Starting FASTCHEK in Interactive Mode

This command is used to start FASTCHEK in Interactive Mode so that the FASTCHEK commands can be entered in a conversational manner.

**Format:**

START [COMMAND [=] [idev:] [,LIST [=] [fd]]

**Parameters:**

| | |
|---|---|
| COMMAND= | idev: specifies the input device from which commands are to be entered and must be an interactive device. If this parameter is omitted or idev: is omitted, the default is the command entry device CON:. Note that throughout this manual all references to the device name CON: in fact refer to the name of the system console if this is other than CON:. |

| | |
|---|---|
| LIST= | fd specifies the output device or file to which all listing information is to be output. If this parameter is omitted or fd is omitted, the list device may be specified during the interactive command input phase. |

**Functional Details:**

When started in interactive mode, FASTCHEK carries on a question (i.e. a prompt) and answer dialogue with the user. Interactive mode is most suited to inexperienced and/or infrequent users of FASTCHEK. A detailed explanation of the interactive mode prompts, and their possible answers, may be found in Chapter 3 of this manual.

After FASTCHEK is started, this message is displayed:

```
    OS/32 FASTCHEK Rnn-nn
```

where nn-nn gives the revision and update level of FASTCHEK. The first prompt message for the dialogue is then displayed.

**Examples:**

```
    START

        (implies that the system console or MTM terminal
        is to be used as the command device)

    ST ,LI=PR:, C=CRT1:

    ST ,LIST FASTCHEK.LST
```

## 2.3.2 Starting FASTCHEK in Batch Mode

This command is used to start FASTCHEK in Batch Mode so that the FASTCHEK commands are read from a file or device. These commands are fully described in Chapter 3.

**Format:**

```
    START ,COMMAND [=] fd1 [,LIST [=] [fd2]]
```

**Parameters:**

| | |
|---|---|
| COMMAND= | fd1 specifies the input file or non-interactive device from which commands are to be read. |
| LIST= | fd2 specifies the output device or file to which all listing information is to be output. If this parameter is omitted or fd2 is omitted, the list device may be specified (using the LIST command) by one of the commands read from the batch file. If this parameter is omitted from both the START command and the Batch command file then PR: is used as the default list device. If the assignment to PR: fails then CON: is used as the secondary default. |

**Functional Details:**

When started in batch mode, FASTCHEK reads commmands from the specified command file or device. These commands are documented in Chapter 3.

After FASTCHEK is started, this message is displayed:

```
    OS/32 FASTCHEK Rnn-nn
```

where nn-nn gives the revision and update level of FASTCHEK. Commands are then read from the batch file or device.

**Examples:**

```
    START ,COMMAND=SCRT:FASTCHEK.CMD

    ST ,LI=PR:, C=CR:

         (where CR: is a card reader)
```

## 2.3.3 Starting FASTCHEK in Immediate Mode

This command starts FASTCHEK in Immediate Mode. This mode is identical to batch mode (in that the commands to FASTCHEK are identical), except that no command device is specified. All the commands required by FASTCHEK must be passed to FASTCHEK as arguments of the START command.

**Format:**

```
    START ,cmd1, cmd2 [, cmd3 [, cmd 4 ..... ]]
```

**Parameters:**

cmd1, cmd2, etc are the FASTCHECK commands. These are discussed in Chapter 3.

**Functional Details:**

In Immediate Mode aLL commands to FASTCHEK must be passed via the START command. It is not possible to pass some commands to FASTCHEK via the START command and the remainder from a command file. Thus for example, the following START command will be rejected:

```
    START ,CHECK DSC2:,COMMAND=SCRT:FASTCHEK.CMD
```

Immediate mode is best used when the number of commands to FASTCHEK can be fitted into the START command. If the required commands cannot be fitted into the START command then batch mode must be used.

After FASTCHEK is started, this message is displayed:

```
    CS/32 FASTCHEK Rnn-nn
```

where nn-nn gives the revision and update level of FASTCHEK. The commands passed in the START command are then processed.

**Examples:**

```
    START ,INIT DSC1:,VOL=SCRT,READCHECK,WRITEREC

    ST ,CHECK=DSC4:,LIST PRIN:

    ST ,RENAME DSC2:,V=GA04
```

## 2.4 STOPPING FASTCHEK

Once FASTCHEK has processed all its initial commands, it commences operation. If the user desires to halt FASTCHEK after it has begun execution, he may use the SEND command of OS/32 to direct a message to FASTCHEK. The two messages which FASTCHEK will recognize are STOP and PAUSE.

Note that if FASTCHEK is being executed outside the MTM environment then it must be selected as the current task by the OS/32 TASK command before the SEND command is entered.

### 2.4.1 SEND STOP

This command instructs FASTCHEK to terminate in an orderly manner. Note that if some operation has been started then the SEND STOP command will not leave the pack in its original state. Thus if an Initialize operation is mistakenly started on the wrong pack, then using SEND STOP will not prevent the destruction of the data on the pack.

Format:

```
    SEND STOP
```

Functional Details:

This command, when received by FASTCHEK instructs the program to make an "orderly" shutdown. To effect its shutdown, FASTCHEK will wait for any outstanding I/O operations to complete, close any open logical units, and delete any workfiles currently being used.

Regardless of the initial command mode (i.e., interactive, batch or immediate), reception of the SEND STOP command will cause FASTCHEK to terminate.

Note that the use of this command is preferred to simply cancelling FASTCHEK because it results in a more orderly shutdown.

### 2.4.2 SEND PAUSE

This command is used to instruct FASTCHEK to pause so that the operator can perform some required action.

**Format:**

```
    SEND PAUSE
```

**Functional Details:**

When this command is received by FASTCHEK, the program will pause. FASTCHEK can then be continued by using the OS/32 CONTINUE command.

The SEND PAUSE facility is normally used in conjunction with the Journal feature (see Appendix H).

# CHAPTER 3
# COMMAND ENTRY

## 3.1 INTRODUCTION

Certain commands must be given to FASTCHEK to specify the function to be performed, the mode of that function, and any required options. As discussed in Chapter 2, these commands can be entered either interactively in response to a series of prompts, or from a file or input device, or as part of the Operating System START command.

In this chapter each command is described. Section 3.2 discusses the commands used in the batch and immediate modes of command entry, and section 3.3 describes the prompts and responses for interactive mode.

## 3.2 BATCH AND IMMEDIATE MODE COMMAND ENTRY

These two methods of command entry are provided for experienced users who are familiar with the command syntax. Batch mode entry is used either when the number of options to be selected is too great to be entered as part of the Operating System START command, or when it is convenient to set up a fixed command file which will be used repeatedly.

When commands are entered as arguments to the Operating System START command each command is separated by a comma. (Note however that some commands may have multiple parameters separated by commas but these are enclosed in parentheses.) When commands are entered from a batch file each line (or more correctly record) read from the file can contain multiple commands and these can be separated either by commas or semi-colons.

Null commands are allowed and are ignored. That is a START command of the form

```
    ST ,arg1,arg2,,,,arg6
```

is valid. Similarly a line read from the batch file can validly have the format

```
    cmd1,cmd2;;;cmd3,,cmd4;,cmd5
```

In general blanks are ignored. That is a command may be preceded or followed by blanks and the following are all equivalent:

```
    cmd1,cmd2,cmd3

    cmd1    ,    cmd2    ; cmd3

         cmd1,   cmd2,   cmd3
```

In addition many of the commands have the form

```
    KEYWORD = value
```

In these cases any blanks preceding or following the equal sign are ignored and the equal sign is not necessary if at least one blank separates the keyword from the value. Thus the following formats are all valid and equivalent:

```
    KEYWORD value

    KEYWORD=value

    KEYWORD =value

    KEYWORD   = value

    KEYWORD=   value
```

In addition, where the value itself has some default setting, the following formats are all valid and equivalent:

```
    KEYWORD

    KEYWORD=

    KEYWORD =

    KEYWORD=default
```

Comments can be entered by preceding the comment by an asterisk. The comment is taken to extend to the end of command line. Thus for example in the line

```
    cmd1,cmd2,* SET OPTIONS 1, 2 AS REQUIRED
```

all characters to the right of the asterisk are taken to be a comment even though the comment may contain commas or semi-colons. This means that on a given line no command will be recognized if it follows a comment since it will be taken to be part of the comment.

A line read from a batch file may be terminated by a carriage return character.

The batch file is read until one of the following conditions is encountered:

- End of File status (X'88') is returned
- End of Medium status (X'90') is returned
- the command 'END' is recognized
  (Note that any commands following the END command on the line containing it will not be processed.)
- an erroneous command is detected

The individual commands are described in the following sections. Note that they are ordered logically rather than alphabetically.

The commands can be divided into three classes: those which specify the function to be performed; those which set the mode of the function; and those which set options and values. Only certain combinations of commands are allowed - for example it is not valid to specify more than one function or mode. When all commands have been entered and individually validated a consistency check is performed before initiating the requested operation.

The following table shows which commands are allowed for each possible function and mode. An "M" indicates that the command is mandatory, an "O" that it is optional, and a blank that it is invalid.

| COMMANDS | INIT READ CHECK | INIT NORD CHECK | INIT FILL | CHECK READ CHECK | CHECK NORD CHECK | CHECK CLOSE | CHECK CLOSE ONLY | RE-NAME |
|---|---|---|---|---|---|---|---|---|
| INITIALIZE | M | M | M | | | | | |
| CHECK | | | | M | M | M | M | |
| RENAME | | | | | | | | M |
| READCHECK | M | | | M | | | | |
| NOREADCHECK | | M | | | M | | | |
| FILL | | | M | | | | | |
| CLOSE | | | | | | M | | |
| CLOSEONLY | | | | | | | M | |
| EXTENDALLOWED | | | | O | O | O | | |
| KEEPSPOOL | | | | O | O | O | | |
| REPORTONLY | | | | O | O | O | O | |
| WRITERECOVERY | O | O | | O | O | O | | |
| NOWRITERECOVERY | O | O | | O | O | O | | |
| BLOCKS | O | O | O | | | | | |
| DIRECTORY | O | O | O | | | | | |
| VOLUME | M | M | M | | | | | M |

Note also that the commands WRITERECOVERY and NOWRITERECOVERY are mutually exclusive as are BLOCKS and DIRECTORY.

Note that any error messages generated while the commands are being processed are logged to the system console (or MTM terminal) and not to the list device. These messages are fully documented in Chapter 5 but in general consist of a plain language message followed by the erroneous command line with an indication of the error position.

## 3.2.1 INITIALIZE

The INITIALIZE command is used to select the Initialize function and also to specify the disc device containing the pack to be initialized.

**Format:**

```
    INITIALIZE [=] devn:

    or

    INITIALISE [=] devn:
```

**Parameters:**

| | |
|---|---|
| devn: | is the device name of the disc drive containing the pack to be initialized. |

**Functional Details:**

The pack on the specified drive will be initialized in the selected mode (FILL, READCHECK, or NOREADCHECK). Note that if no mode is specified then NOREADCHECK is used as the default. The Volume Descriptor, Bit Map, and Directory will be initialized and written. The Pack Admisistration file will be created in the FILL or READCHECK modes and will be updated in the NOREADCHECK mode. Automatic mode switching from NOREADCHECK to READCHECK will occur if the Pack Administration file does not exist or is invalid. For further details see Chapter 4.

The specified device must be currently ready, not write protected, and marked off.

**NOTE**

It should be noted that if a pack is to be initialized in order to clear all currently existing files from the pack, it is most efficient to use the NOREADCHECK mode. This mode is also preferred because it preserves the Pack Administration file. Moreover, if the user wishes to both initialize a pack and also to perform a surface check, and the pack currently contains a Pack Administration file, then this is best done by first initialising the pack in NOREADCHECK mode and then checking it in READCHECK mode since this will preserve the Pack Administration file.

**Examples:**

```
    INIT DSC1:

    INITIALIZE FLP3:

    INIT=D5FX:
```

## 3.2.2 CHECK

The CHECK command is used tc select the Check function and also to specify the disc device containing the pack to be checked.

**Format:**

```
    CHECK [=] devn:
```

**Parameters:**

| | |
|---|---|
| devn: | is the device name of the disc drive containing the pack to be checked. |

**Functional Details:**

An integrity check will be performed on the the pack mounted on the specified drive. The extent of the checking performed is determined by the mode selected (READCHECK, NOREADCHECK, CLOSE, or CLOSEONLY). Note that if no mode is specified then the default is CLOSE. The Volume Descriptor and Directory are always checked. NOREADCHECK forces a check of the file allocation and access paths and READCHECK causes a surface check to be performed as well as the file check. Mode switching from CLOSE to NOREADCHECK to READCHECK will occur automatically if found necessary but no mode switch will occur if CLOSEONLY is selected.

Further operational details can be found in Chapter 4.

The specified device must be currently ready and marked off. The drive must not be write protected unless the REPORTONLY option is used in which case it may be write protected if desired.

**Examples:**

```
    CH DSC1:

    CHECK FLP3:

    CHE=D5FX:
```

## 3.2.3 RENAME

The RENAME command is used to select the Rename function and also to specify the disc device containing the pack to be renamed.

**Format:**

```
    RENAME [=] devn:
```

**Parameters:**

| | |
|---|---|
| devn: | is the device name of the disc drive containing the pack to be renamed. |

**Functional Details:**

The pack on the specified drive will be renamed by changing the volume name in the Volume Descriptor to the specified name. Note that the Volume Descriptor is checked to be valid (see section 4.1.3) before effecting the rename. Note also that a warning message is output if an attempt is made to rename a pack to its current name.

The specified device must be currently ready, not write protected, and marked off.

**Examples:**

```
    REN DSC1:

    RENAME FLP3:

    RENA=D5FX:
```

## 3.2.4 READCHECK

The READCHECK command selects the mode in which either an Initialize or Check function is to be performed and specifies that a surface check of the pack is to be performed.

**Format:**

```
    READCHECK
```

**Parameters:**

None.

**Functional Details:**

In an Initialize/Readcheck operation a surface check is performed and then the Bit Map is allocated and initialized, the Directory is allocated and initialized, and the Volume Descriptor is written.

Similarly, in a Check/Readcheck operation, a surface check is performed before checking the Directory and the file allocation and access paths.

During the surface check an attempt is made to read every sector on the pack. If a given sector cannot be read it is taken to be defective and marked as allocated in the Bit Map. (See section 3.2.12 for the effect of the WRITERECOVERY option.)

If the pack is being initialized then the Pack Administration file, PACKINFO.DIR will be created and initialized and used to record the defective sector addresses.

If the pack is being checked and a valid Pack Administration file is present on the pack then the defective sector address data in this file is updated.

**Examples:**

```
    READCHECK

    REA

    READCH
```

## 3.2.5 NOREADCHECK

The NOREADCHECK command selects the mode in which either an Initialize or Check function is to be performed and specifies that no surface check is to be performed unless found to be required.

**Format:**

```
    NOREADCHECK
```

**Parameters:**

None.

**Functional Details:**

In an Initialize/Noreadcheck operation no surface check of the pack is performed provided that a valid Pack Administration file currently exists on the pack, otherwise the mode automatically switches to READCHECK and a surface check will be done.

In a Check/Noreadcheck operation a full Directory and file integrity check is performed but no surface check is done provided that a valid Pack Administration file exists on the pack, otherwise the mode automatically switches to READCHECK and a surface check will be performed.

**Examples:**

```
    NOREADCHECK

    NOREA

    NOREADCH
```

## 3.2.6 FILL

The FILL command selects the mode in which an Initialize function is to be performed. In an Initialize/Fill operation the specified data pattern is written to every sector on the pack before carrying out a surface check and then initialising the pack.

**Format:**

```
FILL [=] [xxxxxxxx]
```

**Parameters:**

| | |
|---|---|
| xxxxxxxx | is a string of up to 8 hexadecimal digits to be used as the fill data pattern for each fullword in every sector on the pack. If less than 8 digits are specified then leading zeroes are assumed. If no data pattern is entered then a fill pattern of 00000000 is used. |

**Functional Details:**

In an Initialize/Fill operation the pack is first filled with the specified data pattern. The operation then proceeds as for Initialize/Readcheck; that is, a surface check is performed, the Pack Administration file is created and used to record the defective sector addresses, the Bit Map is allocated and initialized, the Directory is allocated and initialized, and the Volume Descriptor is written.

**Examples:**

```
FILL                (data pattern set to 00000000)

FIL = BDBDBDBD

F = 5555            (data pattern set to 00005555)
```

## 3.2.7 CLOSE

The CLOSE command selects the mode in which the Check function is to be performed. In a Check/Close operation the integrity of the Directory is checked and any open files which can be safely closed are closed.

Note that if no mode is specified for a Check operation then Check/Close is selected.

**Format:**

```
    CLOSE
```

**Parameters:**

None.

**Functional Details:**

In a Check/Close operation an integrity check of the Directory is performed. Any Contiguous file found to be open is closed and any Indexed file open only for read is closed. If any Indexed file is found to be open for write or the integrity check of the Directory fails then the NOREADCHECK mode is automatically selected and a Check/Noreadcheck operation is performed.

The CLOSE mode is the preferred mode for the Check operation since only the minimum required checking is performed. If however, the user knows that the pack contains Indexed files open for write, and thus that a mode switch to NOREADCHECK will occur, it is slightly more efficient to initiate the operation in NOREADCHECK mode.

**Examples:**

```
    CLOSE

    CLO
```

## 3.2.8 CLOSEONLY

The CLOSEONLY command selects the mode in which the Check function is to be performed. In a Check/Closeonly operation the integrity of the Directory is checked and any open files which can be safely closed are closed.

Note that in contrast with CLOSE mode, no mode switch will occur if any files cannot be closed or the integrity check fails.

**Format:**

```
    CLOSEONLY
```

**Parameters:**

None.

**Functional Details:**

In a Check/Closeonly operation an integrity check of the Directory is performed. Any Contiguous file found to be open is closed and any Indexed file open only for read is closed.

If any Indexed file is found to be open for write or the integrity check of the Directory fails then the operation is terminated leaving the pack in a state in which it cannot be Marked On as unprotected. In this case a Check/Noreadcheck or Check/Readcheck operation must be performed before the pack can be returned to normal use. Note that if Temporary files are found, these will be deleted (irrespective of file type). However, the space allocated to these files is not released, and a warning message to this effect will be output if any Temporary files are deleted. This space may be reclaimed at a later time by using a Check/Noreadcheck operation.

The CLOSEONLY mode should only be used in cases where it is imperative that the integrity of the pack be determined (and if possible restored) in a very short time. Since, if further checking is required, the user cannot afford the necessary time but must perform some alternate recovery action, for example, by switching to another system.

**Examples:**

```
    CLOSEONLY

    CLOSEO
```

## 3.2.9 EXTENDALLOWED

The EXTENDALLOWED command sets the option which allows Indexed files open for write to be extended as far as possible beyond their last known checkpoint when a Check/Readcheck or Check/Noreadcheck operation is being performed.

Format:

```
EXTENDALLOWED
```

Parameters:

None.

Functional Details:

| If an indexed, nonbuffered indexed, or extendable contiguous file open for write is found during a Check/Readcheck or Check/Noreadcheck operation, it will be closed. If the EXTENDALLOWED option has been set and the file is found to extend beyond the last checkpoint (i.e., the file has been written to since it was previously closed or checkpointed), then, instead of being closed at the last checkpoint, it will be closed and checkpointed to include as many extra records as can safely be recovered. This means that the file is extended up to and including the last record completely contained by the second last nonzero data block pointer in the last checkpointed index block. In practice, this means that no extension is possible for a file which has never been checkpointed or closed.

The EXTENDALLOWED option is only valid if a Check function is to be performed. The option can be set for the READCHECK, NOREADCHECK, and CLOSE modes, but not for the CLOSEONLY mode. Note that specifying EXTENDALLOWED for a Check/Close operation has no effect unless a mode switch to NOREADCHECK or READCHECK occurs. Then any indexed files open for write will be extended.

Examples:

```
EXT

EXTENDALLOWED

EXTEND
```

## 3.2.10 KEEPSPOOL

The KEEPSPOOL command sets the option which allows aged Spool files to be retained and not deleted when a Check operation is being performed.

**Format:**

```
    KEEPSPOOL
```

**Parameters:**

None.

**Functional Details:**

When a Check operation is being performed, any spool file whose Date Last Written is such that the file is older than 24 hours is normally deleted. The KEEPSPOOL option specifies that all (closed) spool files are to be retained irrespective of age.

The KEEPSPOOL option is only valid if a Check function is to be performed. The option can be set for the READCHECK, NOREADCHECK, and CLOSE modes, but not for the CLOSEONLY mode. Note that specifying KEEPSPOOL for a Check/Close operation has no effect unless a mode switch to NOREADCHECK or READCHECK occurs since aged spool files are not deleted in CLOSE mode.

**Examples:**

```
    K

    KEEPSPOOL

    KEEP
```

### 3.2.11 REPORTONLY

The REPORTONLY command sets the option which prevents any writing to a pack on which a Check function is being performed. That is, the integrity of the pack is checked and all problems are reported but no attempt is made to correct them.

Format:

REPORTONLY

Parameters:

None.

Functional Details:

If the REPORTONLY option is specified for a Check function (in any mode) the processing normally carried out is performed except that the pack is never written to. Thus any corrective action which would normally be taken is not performed, but a full report of any problems is given.

The REPORTONLY option is intended for a user who wishes (and has the necessary skill) to attempt to correct a problem by himself (and thus possibly recover some data which would otherwise be lost). As an aid to this type of user, FASTCHEK generates more extensive diagnostic information (than produced when REPORTONLY is not specified) when a problem is found (see section 4.1.2.5).

The REPORTONLY option should also be used if a hardware failure is suspected in order to prevent files from being deleted purely because of errors associated with the hardware failure.

Since the pack is never written to if REPORTONLY is in effect, FASTCHEK can be run with the disc drive write protected.

It should be noted that, if NOREADCHECK or READCHECK mode is specified, the REPORTONLY option requires that a duplicate copy of the Bit Map be constructed (because the one on the pack cannot be updated). The copy can be built either in memory or on a temporary file. Thus the REPORTONLY option can only be used if either sufficient memory or a temporary file of the required size is available. (See also section 4.1.2.5).

If the REPORTONLY option is used in order to check which files will be deleted (when the program is run without REPORTONLY specified), then it is critical that the same segment size increment is used when loading the program. This is because the order of performing certain operations may change as a function of the available memory.

**Examples:**

```
    REP

    REPORTONLY

    REPORT
```

## 3.2.12 WRITERECOVERY and NOWRITERECOVERY

The WRITERECOVERY and NOWRITERECOVERY commands control the option setting that allows the attempted recovery of sectors (by rewriting them) which, during the surface check, are found to have a Cyclic Redundancy Check (CRC) error.

Note that NOWRITERECOVERY must be explicitly specified if this option is not required since WRITERECOVERY is the default.

Format:

    WRITERECOVERY

    NOWRITERECOVERY

Parameters:

None.

Functional Details:

If the NOWRITERECOVERY command is entered then any sectors found to have a CRC error during the surface check operation will be counted as defective. If WRITERECOVERY is specified (or defaulted) then these sectors are written and then read again. If the read error persists, the sector is counted as defective, otherwise it is assumed to be good. Note that the data rewritten to the sector is the same as that initially read from it. This allows the data content of the sector to be preserved, if at all possible.

It should be noted that most CRC errors are caused by a fault (e.g., power failure) occurring while data ia actually being written to the sector.

The WRITERECOVERY option may only be specified for a Check function in any mode other than CLOSEONLY, or an Initialize function in any mode other than FILL. It should be noted that if the mode is other than READCHECK, setting the WRITERECOVERY option will have no effect unless a mode switch to READCHECK occurs. In addition, the WRITERECOVERY option is not allowed if Initialize/Fill is specified because the prefill action corrects any CRC errors which would be recoverable using the WRITERECOVERY option.

**Examples:**

```
WR

WRITERECOVERY

WRITEREC

NOWR

NOWRIT

NOWRITERECOVERY
```

## 3.2.13 BLOCKS and DIRECTORY

The BLOCKS and DIRECTORY commands are used to set the size and position of the Directory allocated during the Initialize operation. The two commands are equivalent except that they set the Directory size in terms of blocks and files respectively. Only one of the two may be used and if neither is entered default values are used.

**Format:**

```
    BLOCKS [=] [bbb] [/[ccc]]

    DIRECTORY [=] [fff] [/[ccc]]
```

**Parameters:**

| | |
|---|---|
| bbb | gives the required size of the Directory in terms of the decimal number of directory blocks. |
| ccc | gives the decimal number of the cylinder on which the Directory is to start (counting the first cylinder as zero). |
| fff | gives the required size of the Directory in terms of the decimal number of files it can contain. |

**Functional Details:**

Each directory block occupies 1 sector and can contain up to 5 file entries. Thus the commands

```
    BLOCKS = 100/1    and    DIRECTORY = 500/1
```

are equivalent. If the size of the Directory is not specified then a default value appropriate to the type of pack being initialized is used. Selected examples are given in the table below. A full list is given in Appendix C.

| Disc Type | Default Blocks | Default Files |
|---|---|---|
| 256 MB | 320 | 1600 |
| 67 MB | 128 | 640 |
| 5 MB | 24 | 120 |
| Floppy | 1 | 5 |

If the starting cylinder number is not specified or is specified as zero, the Directory is located starting on cylinder 1 (i.e., the second cylinder). Cylinder zero is already partly allocated for the volume descriptor sector.

If the area required by the Directory is found to contain defective sectors, the Directory is relocated to the next available error free area (of the required size) that starts on a track boundary.

When the Directory is allocated, the successive directory blocks are not on successive sectors but are on each Nth sector where the value varies depending on the type of disc. Selected examples are given in the table below (see also Appendix C).

| Disc Type | Block Sequencing |
|---|---|
| 256 MB | every 32nd sector |
| 67 MB | every 32nd sector |
| 5 MB | every 6th sector |
| Floppy | every 2nd sector |

Thus, on a 256 MB pack, (which has 64 sectors per track) successive blocks are allocated on sectors 0, 32, 1, 33, 2, 34, ..... 29, 61, 30, 62, 31, 63 and then on the same sectors on the next track. This allocation technique optimizes the time required by the Operating System to scan the Directory.

It should be noted that since during initialization, the Pack Administration file is always created if it did not previously exist, the minimum Directory size is one block. If zero blocks are requested, one will be allocated.

It is important to realize that although it is valid to request the allocation of a Directory which is smaller than is required for the number of files which will be allocated on the pack (since the Operating System will automatically extend the Directory), the Check function of FASTCHEK will execute considerably faster if the Directory is initialized to be large enough to hold all files to be placed on the pack.

It will be apparent that, for a given type of pack, there is a 'reasonable' upper limit to the Directory size. The maximum allowed size for the Directory is taken to be one eighth (1/8) of the total pack size. This represents over 500,000 files on a 256 MB pack.

**Examples:**

```
    B = 300/100
    DIR = 1500/100

    BLOCK                  (implies default values)
    DIRECTORY

    BLOC = /               (implies default values)
    DIRECT = /

    BLOCKS 100             (implies default start cylinder)
    DIRECTORY 500

    BL /20                 (implies default number of blocks)
    DIR /20

    BLO = 0                (minumum Directory allocation)
    DIRECT = 0
```

## 3.2.14 VOLUME

The VOLUME command specifies the volume name of the pack and is mandatory for the Initialize or Rename function (and invalid for the Check function).

**Format:**

```
VOLUME [=] voln
```

**Parameters:**

| | |
|---|---|
| voln | is the volume name. |

**Functional Details:**

The specified volume name can be any valid volume name. That is, it cannot be longer than 4 characters, cannot contain any imbedded blanks, the first character must be alphabetic, and the remaining characters, either numeric or alphabetic. Note that lower case letters are allowed but will be translated to upper case.

It should be noted that it is unwise to use a volume name which is the same as one of the device names in the system (i.e., naming a pack DSC1 if one of the disc drives is called DSC1:) since the Operating System will not allow this pack to be Marked On. It is also unwise to use names which are the same as the keywords used in the Display Devices command (e.g., OFF) since this can lead to confusion.

**Examples:**

```
V SCRT

VOL = work

VOLUME=OS32
```

## 3.2.15 LIST

The LIST command is used to specify the file or device to which messages are output. If this command is omitted then PR: is used as the default list device unless it cannot be assigned in which case CON: (or rather the device name of the system console) is assigned as the list device.

**Format:**

```
LIST [=] fd
```

**Parameters:**

| | |
|---|---|
| fd | is the file descriptor of the file or device to be used as the list device. |

**Functional Details:**

The LIST command may be entered either as part of the Operating System START command or (if batch command entry is used) as one of the commands read from the batch file. However if batch command entry is used then the LIST command cannot be specified in both the START command and as one of the batch commands.

If the file descriptor specifies a file and FASTCHEK is being executed in the MTM environment then the extension can be entered as either P, G, or S (or omitted in which case P is assumed). If FASTCHEK is being executed outside the MTM environment then the account number can be entered as a number (between 0 and 255 inclusive) or omitted (indicating account 0). If P, G, or S is used then this will be taken to mean account 0.

If a file is specified as the list device then it must currently exist. Output to this file will be appended after any existing data.

Note that nothing is output to the list device until all commands have been processed and validated. Thus if for example errors occurred while reading commands from a batch file, then the resulting error messages would not be output to the list device but to the system console (or MTM terminal).

**Examples:**

```
    L PR:

    LIST = D300:FBACK.LST/123

    LIS FBACK.LST/P
```

## 3.2.16 END

The END command is used to indicate the end of the commands being read from the batch command file or device.

**Format:**

```
    END
```

**Parameters:**

None.

**Functional Details:**

When this command is recognized no attempt is made to decode any remaining commands in the current command line and no further records are read from the batch command file or device. The program then performs a consistency check on the commands entered and if no errors are found, commences the execution of the requested function.

Note that the END command is not valid if entered as part of the START command.

**Examples:**

```
    END

    EN
```

## 3.3 INTERACTIVE COMMAND ENTRY

This method of command entry is provided for inexperienced and occasional users who are not familiar with the various command keywords and parameters.

The Interactive Command Entry mode is invoked when the program is started with an interactive device (i.e. a terminal) as the command device. That is, the START command has one of the following formats:

```
    START ,COMMAND=idev: [,LIST=fd]

    START [,LIST=fd]
```

where idev: is the device name of an interactive device and the second format results in CON: being used as the interactive command device.

When started in this mode FASTCHEK outputs a series of prompts requesting various parameters, values, or Yes/No responses. If the response to a given prompt is not valid an error message is output and then the prompt is re-displayed so that a valid response can be entered.

After all the questions have been answered a message is output giving the selected function, its mode, and any options. The user is then asked to comfirm that this data is satisfactory. A negative response causes the complete dialogue to begin again.

The majority of the prompts will accept a carriage return as indicating that a default value is to be used. In these cases the default response is indicated in the prompt message by a number sign character (#). For example

```
    Mode (#NOReadcheck, REAdcheck, or Fill=xxxxxxxx) ?
```

where NOREADCHECK is the default mode. Note that the default value can be explicitly selected if desired, that is, the response "NOREADCHECK" is valid in the above case. However, "#NOREADCHECK" or "#" are not.

The prompt also shows the minimum abbreviations of the allowed keyword responses in upper case with the remainder of the keyword in lower case (provided that the terminal being used supports lower case).

The dialogue is conducted in such an order that mandatory (non-defaultable) parameters are requested first. Once these have been input the user can elect to default the remaining parameters by using the response

```
    !GO
```

In this case the dialogue is terminated and the requested operation commences immediately thus bypassing the remaining questions and also the confirmation step. Note that the !GO response is not allowed until all the mandatory parameters have been obtained but once this has been done the !GO response can be given to any prompt thus defaulting the remainder.

If the user wishes to change his response to a previous prompt he can cause the complete dialogue to be restarted by entering the response

    !RESTART

to any prompt.

Two other special responses are recognized and can be input for any prompt. These are

    !PAUSE

and

    !STOP

The !PAUSE response causes the program to be paused. When it is continued the current prompt message is redisplayed. The !STOP response causes the program to be terminated in an orderly fashion with end of task code 250.

The following sections give the prompt messages used and the allowable responses. The first prompt requests the function to be performed. Since the dialogue varies depending on the selected function, separate sections are used to describe the conversation for each possible function.

Note that in these sections the functional details of each response are not documented since they have been given in sections 3.2.1 through 3.2.15 and these should be consulted if any clarification is required.

## 3.3.1 Dialogue for the Initialize function

The first prompt is

**Function (INITialize, CHeck, or REName) Devn: ?**

to which the response must be

    INITIALIZE [=] devn:
or

    INITIALISE [=] devn:

to select the Initialize function and devn: which is the device name of the drive containing the pack to be initialized.

If the Initialize function is requested, the dialogue continues with the prompt

**Volume Name ?**

to which the response must have the form

    voln

where voln is a valid volume name.

The next prompt requests the mode in which the Initialization is to be performed. It is as follows:

**Mode (#NOReadcheck, REAdcheck or Fill=xxxxxxxx) ?**

If the default mode of NOREADCHECK is not to be used, then the response must be one of the following:

    NOREADCHECK

    READCHECK

    FILL [=] [xxxxxxxx]

where xxxxxxxx is a hexadecimal number of up to 8 digits. Note that default responses can be made to both the above and all remaining prompts. Thus, the special response !GO can be used to terminate the dialogue and commence execution.

The Directory allocation information is then requested using the prompt

**Directory (#nnn Files / Cylinder #m) ?**

where nnn and m indicate the default allocation for the type of disc previously specified. If the default values are not to be used then the response should have the form

    [fff] [/ [ccc]]

where fff is the decimal number of files which the Directory is to contain and ccc is the decimal cylinder number on which the Directory is to start.

If the NOREADCHECK or READCHECK mode was requested earlier then the prompt

**Attempt Write Recovery (#Yes or No) ?**

is output. A default (null) or YES response will enable the WRITERECOVERY option. A NO response will disable it.

If the List device was not specified in the START command the following prompt is displayed:

**List Device (#PR:, @=idev: or FD) ?**

where idev: is the device name of the interactive terminal being used. The default (null) response will select PR: as the list device. A response of "@" will select the terminal being used as the list device. Alternatively any required file or device can be selected by entering its file descriptor. Note that if a file is specified it must currently exist and that the listing information will be appended to any existing data in the file.

At this point all required data has been entered and a message of the following form is output:

```
                        {Fill with xxxxxxxx             }
                        {                               }
Initialize devn:  Mode ={Readcheck [with Writerecovery] }
                        {                               }
                        {Noreadcheck [with Writerecovery]}

    Volume voln   Directory for nnn Files at Cylinder m Requested
```

and this is followed by the prompt

**OK to Run (Yes or No) ?**

If the response is NO then the complete dialogue is restarted. If the response is YES (or !GO) then execution commences. Note that no default response is allowed to this prompt.

## 3.3.2 Dialogue for the Check function

The first prompt is

**Function (INITialize, CHeck, or REName) Devn: ?**

to which the response must be

CHECK [=] devn:

to select the Check function and devn: is the device name of the drive containing the pack to be checked.

If the Check function is requested the dialogue continues with the prompt

**Mode (#CLOse, CLOSEOnly, NOReadcheck or READcheck) ?**

If the default mode of CLOSE is not to be used then the response must be one of the following:

CLOSE

CLOSEONLY

NOREADCHECK

READCHECK

Note that default responses can be made to both the above and all remaining prompts. Thus the special response !GO can be used to terminate the dialogue and commence execution.

If the requested mode was other than CLOSEONLY then the required settings of the EXTENDALLOWED, WRITERECOVERY, and KEEPSPOOL options are solicited using the following prompts:

**Extend Indexed Files (#No or Yes) ?**

**Attempt Write Recovery (#Yes or No) ?**

**Keep Aged Spool Files (#No or Yes) ?**

In all cases a YES response will enable the option and a NO response will disable it. Default (null) responses will disable EXTENDALLOWED and KEEPSPOOL but enable WRITERECOVERY.

Then, irrespective of the selected mode, the prompt

**Report Only (#No or Yes) ?**

is output. A default (null) or NO response will disable this option whereas a YES response will enable the REPORTONLY option thus preventing any modification of the current state of the pack.

If the List device was not specified in the START command the following prompt is displayed:

```
  List Device (#PR:, @=idev: or FD) ?
```

where idev: is the device name of the interactive terminal being used. The default (null) response will select PR: as the list device. A response of "@" will select the terminal being used as the list device. Alternatively any required file or device can be selected by entering its file descriptor. Note that if a file is specified it must currently exist and that the listing information will be appended to any existing data in the file.

At this point all required data has been entered and a message of the following form is output:

```
                    {Close       }
                    {            }
Check devn:   Mode ={Closeonly   }
                    {            }
                    {Noreadcheck}
                    {            }
                    {Readcheck   }

    with [Writerecovery] [Extendallowed] [Keepspool] [Reportonly]
```

and this is followed by the prompt

```
  OK to Run (Yes or No) ?
```

If the response is <u>N</u>O then the complete dialogue is restarted. If the response is <u>Y</u>ES (or !GO) then execution commences. Note that no default response is allowed to this prompt.

## 3.3.3 Dialogue for the Rename function

The first prompt is

**Function (INITialize, CHeck, or REName) Devn: ?**

to which the response must be

```
    RENAME [=] devn:
```

to select the Rename function and devn: is the device name of the drive containing the pack to be renamed.

If the Rename function is requested the dialogue continues with the prompt

**Volume Name ?**

to which the response must have the form

```
    voln
```

where voln is a valid volume name.

If the List device was not specified in the START command the following prompt is displayed:

**List Device (#PR:, @=idev: or FD) ?**

where idev: is the device name of the interactive terminal being used. The default (null) response will select PR: as the list device. (Note that the !GO response to this prompt will also select PR: as the list device.) A response of "@" will select the terminal being used as the list device. Alternatively any required file or device can be selected by entering its file descriptor. Note that if a file is specified it must currently exist and that the listing information will be appended to any existing data in the file.

If the !GO response was not made to the above prompt a message of the following form is output:

```
  Rename devn: as voln
```

and this is followed by the prompt

**OK to Run (Yes or No) ?**

If the response is NO then the complete dialogue is restarted. If the response is YES (or !GO) then execution commences. Note that no default response is allowed to this prompt.

# CHAPTER 4
# FASTCHEK OPERATION

## 4.1 DESCRIPTICN OF OPERATION

This section describes the operation of FASTCHEK. For the sake of simplicity each function (Initialize, Check, and Rename) is treated separately. Note that the Pack Administration file is discussed in detail in section 4.4.

### 4.1.1 Initialize Function

Certain actions are common to all (Fill, Readcheck, and Noreadcheck) modes of Initialization.

The first operation performed during an Initialization (irrespective of the mode) is a validity check of the Volume Descriptor. This check involves reading the Volume Descriptor; and then rewriting it with a volume name of "NULL", the Bit Map and Directory pointers set to zero, and the Volume On-line Attributes bit set. The sector is then re-read and the data read compared with that which was written. (Note that initially setting up the Volume Descriptor in this way ensures that the pack cannot be marked on if the Initialize function terminates abnormally, since a "DUPL-ERR" error will occur because the name of the pack conflicts with that of the Null device.) If an error occurs while writing or re-reading the Volume Descriptor the standard I/O error message (see Chapter 5) is output followed by

```
  WHILE ACCESSING VOLUME DESCRIPTOR
```

and then the program terminates with end of task code 10. If the data read back does not match that written the program will terminate with end of task code 31 after printing the message

```
  VOLUME DESCRIPTOR DATA VALIDATION ERROR
      IN FULLWORD AT xx EXPECTED yyyyyyyy FOUND zzzzzzzz
```

Any defective sectors are then located (either by a surface check or from the information in the Pack Administration file) and space for the Pack Administration file is either allocated (in the Fill and Readcheck modes) or determined (in Noreadcheck mode).

The Directory is then allocated. Initially an attempt is made to allocate a directory starting on the first track of the requested cylinder. If this is not possible because the required area contains defective sectors, then the starting address is incremented by one track at a time and further attempts are made. If the directory cannot be allocated the program terminates with end of task code 20 after printing the message

```
  INSUFFICIENT ERROR-FREE SPACE FOR DIRECTORY
```

Note that no attempt is made to reposition the Directory at a location before the start cylinder specified by the user. Thus if the Directory cannot be allocated, the user should rerun the program specifying either a smaller start cylinder number or a smaller directory.

Once the Directory has been allocated it is initialized. The Bit Map is then allocated in the first error free area of the required size immediately following the Directory. If this cannot be done the program terminates with end of task code 21 after printing the message

```
  INSUFFICIENT SPACE FOR BIT MAP
```

Note that if this occurs and a relatively high start cylinder number was specified for the Directory, then the user should rerun the program specifying either a smaller starting cylinder number or a smaller directory.

In general, if the program is unable to allocate either the Bit Map or the Directory, and the user has specified a low start cylinder number for the Directory, a hardware failure is indicated because there will be a large number of defective sectors.

Once the Bit Map has been allocated it is initialized to reflect the allocation of the Directory, the Volume Descriptor, the Pack Administration file, the Bit Map itself, and any defective sectors. The Bit Map initialization is done by first clearing the complete Bit Map and then setting the required bits. Note that a check is made to ensure that the Bit Map can be read and actually contains all zeros. If a defective sector is detected then a mode switch to Readcheck will occur. However, if the data is successfully read but is not all zeros the program terminates with end of task code 30 after printing the message

```
  DATA VALIDATION ERROR IN BIT MAP AT SECTOR xxxxxx
      IN FULLWORD xx EXPECTING 00000000 FOUND zzzzzzzz
```

If this occurs a hardware failure is indicated.

A similar check is made while initializing the Directory. The actions are as for the Bit Map check except that the message

```
  DATA VALIDATION ERROR IN DIRECTORY AT SECTOR xxxxxx
      IN FULLWORD yy EXPECTING 00000000 FOUND zzzzzzzz
```

is printed.

The final step in the initialization is to write the Volume Descriptor containing the specified volume name and the addresses of the Directory and Bit Map. A copy of the Volume Descriptor is also written in the last sector of the pack. This might be useful when attempting to recover a pack whose Volume Descriptor was overwritten.

The message

```
 PACK INITIALIZED - PREALLOCATED DIRECTORY AT xxxxxx,
                                BIT MAP AT yyyyyy
```

is then output to show that the Initialize operation is complete and to give the locations of the start of the Dircetory and Bit Map.

#### 4.1.1.1 Initialize/Fill Operation

If the Fill mode is specified, then once the initial check of the Volume Descriptor has been performed, all other sectors on the pack are filled with the specified data pattern. This operation is optimized by using the largest possible buffer up to a maximum of one cylinder and ensuring that no write crosses a cylinder boundary.

If a defective sector is encountered during the Fill operation, it will be included in the defective sector list even if it is not found during the surface check. (It is possible to have a sector that can be read but not written, in particular on an MSM type disc, because the read retry logic is more extensive.)

When the Fill phase is complete, a surface check is performed. This is described in the next section.

#### 4.1.1.2 Initialize/Readcheck Operation

If the Readcheck mode is specified, then once the initial check of the Volume Descriptor has been performed, a surface check is made on the remainder of the pack. This operation is optimized by using the largest possible buffer up to a maximum of one cylinder and ensuring that no read crosses a cylinder boundary. Note that if the pack being checked is a 2.5 MB or 5 MB fixed or removable pack, (i.e., device codes 48 through 51 decimal) because of the characteristics of the controller used for these types of discs, it is necessary to read all sectors individually in order to locate all defective sectors. A leapfrog algorithm is used to optimize this process. Note that the same technique is required and is used for the Fill operation.

Users who wish to monitor the progress of the surface check operation can do so by examining register A, which will contain the current sector address.

If an error occurs while reading a sector and the WRITERECOVERY option is set then the sector is written to in an attempt to correct a possible Cyclic Redundancy Check error. The sector is then re-read.

All sectors found to be defective are recorded in a defective sector list in memory. If a large number of defectives are found this list will be expanded at the expense of the read buffer. If so many defectives are found that no read buffer remains, the program will terminate with end of task code 22 after printing the message

```
  DEFECTIVE SECTOR LIST OVERFLOW AFTER nnnnnn FOUND
```

Note that the defective sector addresses are printed as they are found and thus in the above case all those found will be displayed. The defective sector addresses are displayed using the message format

```
  DEFECTIVE SECTOR AT xxxxxx  (CHS=ccc/hh/ss)   <text>
```

where xxxxxx is the hexadecimal sector address, ccc/hh/ss gives the equivalent hexadecimal cylinder, head and sector numbers, and <text> will be one of the following explanatory messages

```
        RECOVERED

        STATUS zzzz READING

        STATUS zzzz WRITING
```

where the first indicates that the sector was recovered through the WRITERECOVERY option, and the others that the defective sector was detected either during a read or write operation.

If the program terminates because the defective sector list overflowed (or the program terminates normally but with an exceptionally large number of defective sectors), then there is almost certainly an underlying hardware fault. Alternatively, the pack may have been formatted on a drive differently aligned to that being currently used.

When the surface check is complete the number of defective sectors found is logged. Then, if a valid Pack Administration file previously existed on the pack, the newly found defective sectors are compared with those previously recorded. Any discrepancies are logged using messages of the form

```
  CURRENT/PREVIOUS DEFECTIVE SECTOR DISCREPANCIES
      xxxxxx  (CHS=ccc/hh/ss) NOW GOOD
      xxxxxx  (CHS=ccc/hh/ss) NOW DEFECTIVE
```

A large number of discrepancies should be taken as an indication of hardware failure. If there are no discrepancies the message

```
      **** NCNE ****
```

is displayed after the heading.

The Pack Administration file is then created (since the required size is known from the number of defectives) and used to record the defective sector addresses.

Initialization then continues as discussed earlier in section 4.1.1.

## 4.1.1.3 Initalize/Noreadcheck Operation

If the Noreadcheck mode is specified, then once the initial check of the Volume Descriptor has been performed, a validity check of the Pack Administration file is made. This check is discussed in section 4.4.

If the Pack Administration file is found to be valid then the defective sector addresses are read from the file. The defective sector data is then reported as follows

```
  nnnn DEFECTIVE SECTORS RECCRDED
      xxxxxx  (CHS=ccc/hh/ss)
      xxxxxx  (CHS=ccc/hh/ss)
```

The Initialization then proceeds as discussed in section 4.1.1. If the Pack Administration file does not exist or is invalid, an automatic switch to Readcheck mode occurs and the operation proceeds as discussed in section 4.1.1.2.

## 4.1.2 Check Function

All modes (Closeonly, Close, Noreadcheck, and Readcheck) of the Check function start with a validity check of the Volume Descriptor. This involves reading the Volume Descriptor, rewriting it with the Volume Attributes On-line bit set, and then reading it back and comparing the data read and written. If an error occurs on any of these operations, the standard I/O error message (see Chapter 5) is output followed by the message

```
  WHILE ACCESSING VOLUME DESCRIPTOR
```

and the program then terminates with end of task code 10. If the data read back does not match that written the program will terminate with end of task code 31 after printing the message

```
  VOLUME DESCRIPTOR DATA VALIDATICN ERROR
      IN FULLWCRD xx EXPECTEC yyyyyyyy FOUND zzzzzzzz
```

The contents of the Volume Descriptor are then checked and if found to be invalid the program will terminate with end of task code 8 after printing

```
VOLUME DESCRIPTOR ERROR
```

followed by one of the messages below

```
INVALID VOLUME NAME vvoollnn

INVALID DIRECTORY POINTER xxxxxxxx

INVALID BIT MAP POINTER xxxxxxxx

BIT MAP (AT xxxxxx TO xxxxxx) OVERLAPS DIRECTORY (AT xxxxxx)
```

where vvoollnn is the hexadecimal representation of the volume name and xxxxxx are hexadecimal addresses. Note that the Directory and Bit Map pointers are invalid if they are greater than the maximum sector address on the pack. In addition, the Bit Map pointer cannot be zero.

The validity of the Pack Administration file is then checked (see section 4.4). If the file is invalid or does not exist, then if the initially selected mode is Noreadcheck or the program later switches from Close to Noreadcheck, then Readcheck mode will be automatically selected since the defective sector information cannot be read from the file but must be obtained from a surface check.

After these initial checks are made, the Check function proceeds as discussed in the sections below. When the operation is complete the program will terminate with end of task code 0 if the pack can then be marked on as unprotected and used normally and in this case the message

```
CHECK COMPLETE - VOLUME voln READY TO BE MARKED ON
```

is logged. Note that in this case, if the last sector on the pack is not in use, then a duplicate copy of the Volume Descriptor is written to this sector. This may be used when attempting to recover a pack whose Volume Descriptor has been overwritten.

If Closeonly mode was used and further checking is required then the program will terminate with end of task code 1 after printing the message

```
**** CHECK/NOREADCHECK REQUIRED ****
```

and in this case the Volume Attributes On-line bit will be left set so that the pack can only be marked on as protected and it will be necessary to run a Check/Noreadcheck or Check/Readcheck operation before the pack can be returned to normal use.

If the Reportonly option was set and the program would otherwise terminate with end of task code 0, then it will terminate with end of task code 1 after printing the message

```
  **** OPTION REPORTONLY SET ****
  **** ACTION MESSAGES ARE ADVISORY ONLY ****
```

#### 4.1.2.1 Check/Readcheck Operation

Once the initial checking described in section 4.1.2 has been carried out, a surface check is performed in the same manner as described in section 4.1.1.2.

The operation then proceeds as described in the following section except that if an unexpected defective sector is encountered, the program will terminate (with end of task code 10 after printing the appropriate I/O error message) instead of switching to the Readcheck mode (since the program is already in this mode).

#### 4.1.2.2 Check/Noreadcheck Operation

If no valid Pack Administration file exists, the mode automatically switches to Readcheck and the actions described in section 4.1.2.1 are performed. If the Pack Administration file is valid, the defective sector data read from it is reported as discussed in section 4.1.1.3.

The Bit Map is then cleared and checked as described in section 4.4.1 and the bits corresponding to the Volume Descriptor and the Bit Map itself are set. If a defective sector was found within the area occupied by the Bit Map, the message

```
  BIT MAP CONTAINS DEFECTIVE SECTOR AT xxxxxx
```

is printed. If a new Bit Map can be built either in memory or on a Temporary file, this will be done. Otherwise, the program terminates with end of task code 5 after printing the message

```
  INSUFFICIENT WORKSPACE FOR DUPLICATE BIT MAP
```

If this happens, the user has the option of rerunning the program with either sufficient memory to build an in memory Bit Map (see Appendix C) or sufficient contiguous file space available on the Temporary Volume. Alternatively, the pack can be backed up, re-initialized and then restored.

If workspace exists for the duplicate Bit Map when the file allocation check is complete, the new Bit Map will be scanned for a free area of the required size and the Bit Map will be copied into this area. If the required space cannot be found, the program will terminate with end of task code 21 after printing the message

```
INSUFFICIENT SPACE FOR BIT MAP
```

At this point, the user must backup the pack, re-initialize it, and then restore it.

The Directory entries are then checked. The Directory blocks are read on a track basis (thus obtaining multiple blocks per read) until the end of the preallocated portion of the Directory is detected, at which point the blocks are individually read. If an error occurs while reading a directory block (i.e., the Directory contains a defective sector) then the Directory is truncated at the previous block and the standard I/O error message (see Chapter 5) is printed followed by

```
WHILE ACCESSING DIRECTORY
DIRECTORY TRUNCATED TO nnnn BLOCKS AT SECTOR xxxxxx
```

where nnnnn is the decimal number of what is now the last block, and xxxxxx is its hexadecimal sector address. If this occurs, all files which were contained in the truncated portion of the Directory will be lost.

Each block is checked as follows: first, the five entries in the block are checked to ensure that each active entry is correct. That is, the file name is valid, the file type is valid (i.e., the file is either an indexed, nonbuffered indexed, contiguous or extendable contiguous file). Then if it is a contiguous file, the first and last sector addresses are valid; and if an Indexed file, the index and data block sizes are nonzero, and the first and last index block addresses are valid. Failure of these checks will result in the following messages:

```
INVALID FILE NAME ffffffffffffff.eeeeee/act

filename.ext/act - INVALID FILE TYPE x

filename.ext/act - INVALID FIRST SECTOR ADDRESS xxxxxxxx

filename.ext/act - INVALID LAST SECTOR ADDRESS xxxxxxxx

filename.ext/act - LAST SECTOR ADDRESS xxxxxx LESS THAN
                                        FIRST xxxxxx

filename.ext/act - INVALID INDEX BLOCK SIZE OF ZERO

filename.ext/act - INVALID DATA BLOCK SIZE OF ZERO

filename.ext/act - INVALID FIRST INDEX BLOCK ADDRESS xxxxxxxx

filename.ext/act - INVALID LAST INDEX BLOCK ADDRESS xxxxxxxx

filename.ext/act - FIRST INDEX BLOCK ADDRESS xxxxxxxx
                            EQUALS LAST - SHOULD NOT

filename.ext/act - FIRST INDEX BLOCK ADDRESS xxxxxxxx NOT
                                     EQUAL TO LAST xxxxxxxx
```

```
  filename.ext/act - INVALID NUMBER OF LOGICAL RECORDS xxxxxxxx

  filename.ext/act - INVALID RECORD LENGTH nnnn OR NUMBER
                                              OF RECORDS nnnnnnn
```

The message

```
  FILE filename.ext/act DELETED
```

will then be printed. Note that when an invalid filename is encountered the hexadecimal equivalent of the name is given by

```
  ffffffffffffffff.eeeeee/act
```

and in the file deleted message, unprintable characters are replaced by # characters.

Once this check of the five entries has been made, the Forward Pointer to the next directory block is checked. If it is invalid or points to an already allocated sector, one of the following messages will be printed:

```
  DIRECTORY BLOCK AT xxxxxx HAS INVALID FORWARD POINTER xxxxxxxx

  DIRECTORY SECTOR AT xxxxxx CHAINS TO ALLOCATED SECTOR xxxxxx
```

and the Directory is then truncated. If the current directory block was preallocated, the Directory is truncated at this block, otherwise it is truncated at the last non-preallocated block containing active entries. In either case the message

```
  DIRECTORY TRUNCATED TO nnnn BLOCKS AT SECTOR xxxxxx
```

is printed. When the Check operation is completed, the Directory will be truncated, if necessary, to remove any trailing empty directory blocks from the non-preallocated portion of the Directory. This message is not displayed unless the REPORTONLY option is selected.

The allocation and state of each of the files in the block is then checked.

Contiguous files are processed as follows: the area occupied by the file (indicated by the first and last sector addresses) is checked to be used and, if so, is flagged as used in the Bit Map. If an allocation conflict is detected, one of the following messages will be printed:

```
  filename.ext/act - ALLOCATION CONFLICT AT SECTOR xxxxxx

  filename.ext/act CONTAINS DEFECTIVE SECTOR AT xxxxxx

  filename.ext/act CONTAINS RECOVERED DEFECTIVE SECTOR AT xxxxxx
                     IN SECTOR nnnnnn OF FILE
```

The first two messages are followed by

```
  FILE filename.ext/act DELETED
```

and the third by

```
  FILE filename.ext/act MAY CONTAIN ERRONEOUS DATA
```

If the file is open for write, it will be closed and the message

```
  filename.ext/act OPEN FOR WRITE (COUNT xxxx)
  FILE filename.ext/act CLOSED - POSSIBLE LOST DATA
```

is printed. If the file is open for read, it will be closed but no message will be generated. Finally, the current sector address within the directory entry is checked to be less than or equal to the size of the file. If not, it is reset to zero, and the message

```
  filename.ext/act - INVALID CURRENT SECTOR ADDRESS xxxxxxx RESET
```

is printed.

| Nonbuffered indexed, extendable contiguous, and indexed files are processed as follows: if the file is open for write, it will be closed and the message

```
  filename.ext/act OPEN FOR WRITE (COUNT xxxx)
```

is printed. After the file allocation is validated (as discussed below), this message will be followed by

```
  FILE filename.ext/act CLOSED - POSSIBLE LOST DATA
```

Note that if the file is found to extend beyond its last checkpointed extent, the message

```
  filename.ext/act TRUNCATED TO CHECKPOINT AT RECORD nnnn
```

or

```
  filename.ext/act EXTENDED FROM nnnn TO nnnn RECORDS
```

will be printed depending on whether the EXTENDALLOWED option has been set or not. Note that the file can be validly truncated to record 0 since this implies that a Directory entry still exists for the file, but that it contains no data.

If the file was open for read, its read count will be reset but no message will be generated.

The validity of the file allocation is checked by reading forward through the index blocks, checking their linkages, checking the data block pointers, and checking for allocation conflicts for all index and data blocks. Failure of these checks will result in one of the following messages:

```
filename.ext/act - INVALID NEXT INDEX BLOCK ADDRESS xxxxxxxx

filename.ext/act - INVALID PREVIOUS INDEX BLOCK ADDRESS xxxxxxx

filename.ext/act - INVALID DATA BLOCK ADDRESS xxxxxxxx
```

```
  filename.ext/act - NEXT INDEX BLOCK ADDRESS xxxxxxxx SHOULD
                                                        BE ZERO

  filename.ext/act - LAST INDEX BLOCK ADDRESS xxxxxxxx
                                        SHOULD EQUAL xxxxxxxx

  filename.ext/act - ALLOCATION CONFLICT AT SECTOR xxxxxx

  filename.ext/act CONTAINS DEFECTIVE SECTOR AT xxxxxx
```

If any of the above errors occur, the action taken depends on the current state of the file and the position of the error within the file. If the file was not open for write then any of the above errors result in the file being deleted. If the file was open for write and the error is detected within the previously checkpointed extent of the file (i.e. within the Index and Data blocks containing the current number of logical records as indicated by the Directory entry) then the file is deleted. If the file was open for write and the EXTENDALLOWED option is set then if the error is in the extended portion of the file, the file is not extended but truncated at the last checkpoint. Thus if the file is deleted the error message will be followed by

```
  FILE filename.ext/act DELETED
```

but if it is truncated the error message will be followed by

```
  FILE filename.ext/act TRUNCATED TO CHECKPOINT AT RECORD nnnnn
```

If one of the sectors occupied by the file is a recovered defective sector then the message

```
  filename.ext/act CONTAINS RECOVERED DEFECTIVE SECTOR AT xxxxxx
```

If this sector is within the previously checkpointed extent of the file then the file is deleted if the sector lies within an Index block, otherwise (if the sector is part of a Data block) then the file is not deleted and the message

```
    DATA POSSIBLY CORRUPTED BEGINNING AT RECORD nnnn
```

is printed.

If the file was not open for write then the unused data block addresses in the last index block are checked to be zero. If one is found to be nonzero then the message

```
  filename.ext/act - INVALID DATA BLOCK ADDRESS xxxxxxxx
                                           - SHOULD BE ZERO
```

is printed. Note however, that the file will not be deleted since this is not considered to be a fatal error. The data block address will be set to zero and all following data block pointers will be forced to zero. This cleanup operation is also performed for any files which are truncated or extended.

In the above discussion, no explicit reference has been made to either Spool or Temporary files. These are treated in a slightly different fashion from standard data files.

Temporary files are always deleted. No messages are output as the individual files are deleted, but at the end of the Check operation the message

```
  nnnn TEMPORARY FILES DELETED
```

is output giving the total number deleted.

Spool files are treated as follows: first, the file type must be indexed. If not, the message

```
  filename.ext/act - INVALID SPOOL FILE TYPE x
  FILE filename.ext/act DELETED
```

is output. If the file is closed, then it is deleted if it is more than 24 hours old (as determined by its Date Last Written) unless the KEEPSPOOL option is set. If the file is open for write, it is treated as a standard user data file. Once the file has been properly closed (or was not open for write), it is deleted if it currently contains no data. When all checking is complete, the following summary messages are output if the associated counts are nonzero.

```
  nnnn AGED SPOOL FILES DELETED

  nnnn EMPTY SPOOL FILES DELETED

  nnnn SPOOL FILES PRESENT
```

If the last message is printed, the user should, once the pack has been marked on and the Spooler is active, check the Spool Queue to ensure that the remaining files are, in fact, on the Queue. If not, add them to the Queue.

When the checking of Directory blocks and files is complete, the Volume Descriptor is rewritten with the Volume Attributes Online bit reset.

Two final points should be noted. First, if during the Check/Noreadcheck operation, a defective sector is encountered within the Bit Map, Directory, or the file index blocks, then an automatic mode switch to Readcheck will occur. This will be indicated by the standard I/O error message followed by

```
  **** SWITCHING TO READCHECK MODE ****
```

A surface check will then be performed before recommencing the checking discussed above.

Second, the allocation and linkage checking of indexed, | nonbuffered indexed and extendable contiguous files is done in parallel for a number of files. As stated earlier, the preallocated Directory is read in, one track at a time. Once all the entries in the blocks in that track are checked, all the indexed and nonbuffered type files found are checked together by using a tree based sweeping algorithm. This technique optimizes the checking of indexed, nonbuffered indexed, and extendable contiguous files but can only be used for the preallocated portion of the Directory. When initializing a pack, request a Directory big enough to contain all files to be allocated on the pack. Note that because the file is not completely checked until all its index blocks have been checked, it is normal to have messages relating to the one file interspersed with messages relating to other files. For example, the "OPEN FOR WRITE" message is commonly separated from the "..CLOSED - POSSIBLE LOST DATA" message.

### 4.1.2.3 Check/Close Operation

Once the initial check of the Volume Descriptor (discussed in section 4.1.2) has been made, the Directory is checked in the same manner discussed in section 4.1.2.2. Each Directory entry is validated but no allocation checks are made. If any files are found to be open for read, their read counts will be reset, but no message is generated.

If any contiguous file is found to be open for write, it will be closed and the following messages printed:

```
  filename.ext/act OPEN FOR WRITE (COUNT xxxx)
  FILE filename.ext/act CLOSED - POSSIBLE LOST DATA
```

If any indexed, nonbuffered indexed, or extendable contiguous file is found to be open for write, the message

```
  filename.ext/act OPEN FOR WRITE (COUNT xxxx)
```

followed by

```
  **** SWITCHING TO NOREADCHECK MODE ****
```

indicating that a mode switch is occuring. Note that the mode switch occurs when the first indexed, nonbuffered indexed, or extendable contiguous file open for write is found. In addition, a mode switch will occur if any of the Directory blocks or entries are found to be invalid or a Temporary file is encountered.

If no errors are found, and there are no files open for write (except possibly contiguous files), and no Temporary files are encountered, the Volume Descriptor is rewritten with the the Volume Attributes Online bit reset, and the program terminates with end of task code 0.

### 4.1.2.4 Check/Closeonly Operation

This operation is exactly the same as the Check/Close operation except that, if any errors or indexed, nonbuffered indexed, or extendable contiguous files open for write are detected, no mode switch occurs and the program terminates immediately with end of task code 1. If this happens, the Volume Attributes Online bit in the Volume Descriptor will be left set, and it will be necessary to run a full check before the pack can be returned to normal use.

Note however, that if any Temporary files are encountered, these will be deleted. Since this action leaves space allocated in the Bit Map which is in fact free, the message

```
WARNING: SPACE NOT RELEASED IN BIT MAP
```

is printed just before the program terminates with end of task code 0. In this case, the pack can safely be Marked On and used normally. However, at some future time, a Check/Noreadcheck operation should be run to release the unused space.

### 4.1.2.5 Reportonly Operation

If the REPORTONLY option is set for a Check operation, the pack is never written to and any messages generated will, in general, contain extra diagnostic information.

Since the pack cannot be written to, the write/read-back/compare checks cannot be made on the Volume Descriptor and the Bit Map. In addition, a duplicate copy of the Bit Map must be constructed either in memory or in a Contiguous Temporary file. In both cases, an area equal to the size of the Bit Map is required (see Appendix C). If the required workspace cannot be obtained, the program will terminate with end of task code 5 after printing the message

```
INSUFFICIENT WORKSPACE FOR DUPLICATE BIT MAP
```

When all checking is complete, this duplicate bit map is compared with that currently existing on the pack. If there are any differences, the message

```
CURRENT BIT MAP DIFFERS FROM EXPECTED
```

is printed. If the Journal feature (see Appendix H) is currently enabled, then the differences are reported as follows:

```
nnnn BIT(S) STARTING AT xxxxxx SET - EXPECTED RESET
nnnn BIT(S) STARTING AT xxxxxx RESET - EXPECTED SET
```

Note that if the pack being checked was previously marked off, any differences (and any reported errors other than defective sectors) indicate a possible failure in the operating system.

As indicated above, most of the errors reported will include additional information. For example, where an error is detected in a Directory entry, the location of the entry is reported by

```
  IN DIRECTORY BLOCK AT xxxxxx OFFSET yy
```

where xxxxxx is the sector address of the block in hexadecimal, and yy is the offset in hexadecimal of the entry within the block.

Similarly, errors within the index blocks of an indexed, nonbuffered indexed, or extendable contiguous file are located by the message

```
  IN INDEX BLOCK nnnn AT xxxxxx
```

where nnnn is the block number (base 1) and xxxxxx is the hexadecimal sector address. Errors in the data blocks are located by the message

```
  IN DATA BLOCK nnnn AT xxxxxx OFFSET yy
  IN INDEX BLOCK mmmm AT zzzzzz
```

Errors in the data block pointers are located by the message

```
  IN INDEX BLOCK nnnn AT xxxxxx OFFSET yy
```

In addition, certain actions not normally reported, will be. Thus, files found open for read are reported as

```
  filename.ext/act OPEN FOR READ (COUNT xxxx)
  IN DIRECTORY BLOCK AT xxxxxx OFFSET yy
  FILE filename.ext/act READ COUNT RESET
```

Similarly, the deletion of Temporary and Spool files is reported by

```
  TEMPORARY FILE filename.ext/act DELETED

  AGED SPOOL FILE filename.ext/act DELETED

  EMPTY SPOOL FILE filename.ext/act DELETED
```

Remember that any action message (such as "FILE ... DELETED") is advisory only in REPORTONLY mode, and the user is reminded of this by the note

```
  **** OPTION REPORTONLY SET ****
  **** ACTION MESSAGES ARE ADVISORY ONLY ****
```

printed both at the head and the tail of the listing.

When the program finishes, it terminates with end of task code 1.

## 4.1.3 Rename Function

The Volume Descriptor is first read and then checked as discussed in section 4.1.2. That is, the current volume name is checked to be valid, and the Directory and Bit Map pointers are checked. In addition the Volume Attributes On-line bit is checked to be reset and if not the message

```
  VOLUME ATTRIBUTES ON-LINE BIT SET - CHECK REQUIRED
```

is printed and the program then terminates with end of task code 1.

The Volume Descriptor is then rewritten with the new volume name. If this is the same as the previous name then the message

```
  WARNING: VOLUME NAME IS ALREADY voln
```

is printed.

The program then terminates with end of task code 0 after printing the message

```
  PACK ON devn: RENAMED FROM nold TO voln
```

where nold is the old volume name and voln the new.

## 4.2 TIMING INFORMATION

Note: all timings given in this section assume that there is no other concurrent activity.

The time required to execute a given function depends on the function itself, the selected mode, the type of disc, the contents of the pack, and the amount of memory available.

Rename operations require very little time and will always execute in under 10 seconds.

Initialize operations are relatively fast if no surface check is required. However, if a surface check is performed, the execution time will depend on both the available memory (as this determines the buffer size) and the type of pack. The time required is given by the formula

$$T = R * C * ( H + N )$$

| | | |
|---|---|---|
| where | T | is the total time in seconds |
| | R | is rotation time of the disc in seconds |
| | C | is the number of cylinders on the pack |
| | H | is the number of heads on the disc |
| | N | is the number of reads required to process one cylinder and is given by INT((S+B-1)/B) where S is the cylinder size and B the buffer size except for the 2.5 and 5 MB fixed and removable packs where (because of the different algorithm used) N has the fixed values 5 and 7 respectively. |

The values of the device dependent parameters in the above expressions can be found in Appendix C. The times for various types of disc and various buffer sizes are given in the following table. (Figures are given for Bit Map plus track and a half sized buffers because this size is the optimum for Check/Noreadcheck operations.)

| Disc Type | Buffer Size (KB) | Equivalent To | Surface Check Time (sec) |
|---|---|---|---|
| 256 MB | 304 | cylinder | 274 |
| 256 MB | 147 | Bit Map + 1.5 trks | 302 |
| 256 MB | 32 | 2 tracks | 398 |
| 256 MB | 16 | track | 521 |
| 67 MB | 80 | cylinder | 82 |
| 67 MB | 57 | Bit Map + 1.5 trks | 96 |
| 67 MB | 16 | track | 137 |
| 5 MB | independent of buffer size | | 92 |
| FLOPPY | 4 | cylinder | 26 |

In the above table, the maximum buffer size given for each disc type is the optimum for that disc. That is, further increasing the buffer size will not decrease the required time. Note also that the time is independent of buffer size for 2.5 and 5 MB discs because of the different algorithm used which requires only a 1-sector buffer. It should be apparent both from the table and from the formula, that significant reductions can be made in the buffer size without greatly affecting the performance. In particular, if a track sized buffer is used instead of (the optimum) cylinder buffer, then the required time will always be less than double the minimum possible. Thus, for a 256 MB disc, the buffer size can be reduced from 304 to 16 KB (i.e., by 94%) with less than a doubling of the required time.

The time required for a Fill operation is equal to that required for a surface check. That is, an Initialize/Fill will require twice the time given in the table.

The time required for a Check operation depends critically on the specified mode (or the mode used if a mode switch occurs). A full Check will require a Directory check, a File check, and a surface check. The Directory check is essentially very fast, and a Check/Closeonly or a Check/Close in which no mode switch occurs will process in excess of 2000 files per second in the preallocated portion of the Directory, and approximately 150 files per second in the non-preallocated portion. These figures assume that sufficient memory is available to hold one complete track of the preallocated portion of the Directory. (Note that these figures apply to the 256 and 67 MB discs; for the 5 MB discs the figures are approximately 1000 and 75, respectively; and 100 and 5, respectively, for a Floppy.)

The File check requires more time than a Directory check and is optimized by specifying sufficient memory to hold the entire Bit Map in core together with a track sized buffer for the Directory and a half track buffer for the Index file check tree. Assuming that the average Contiguous file is 75 KB in size, and the average Indexed file contains 1.5 index blocks (equivalent to a source file of about 1500 records) then on 256 and 67 MB discs, the File checking logic will process approximately 100 Contiguous files per second and 20 Indexed files per second within the pre-allocated part of the Directory. In the non pre-allocated portion, these figures drop to approximately 80 and 15 respectively. These figures assume that sufficient memory is available to hold the entire Bit Map in memory. If only one quarter of the Bit Map can be held in memory at one time, there will be approximately a 10% degradation in the the above figures. Timings for the other types of discs can be estimated by reducing these figures by the performance ratios evident from the Directory check estimates.

## 4.3 TUNING INFORMATION

If only a single disc has to be processed, then optimum performance is achieved by using the maximum available amount of memory (up to the limit useable by the program). The following table gives the segment size increments required for optimum performance for each function/mode for selected types of discs. The figures given are calculated by using a cylinder sized buffer for surface check operations, a Bit Map plus a track and a half sized buffer for file check operations, and a track buffer for Directory check operations.

| DISC TYPE | INIT READ CHECK | INIT NORD CHECK | INIT FILL | CHECK READ CHECK | CHECK NORD CHECK | CHECK CLOSE | CHECK CLOSE ONLY | RE-NAME |
|---|---|---|---|---|---|---|---|---|
| 256 MB | 304 | 139 | 304 | 304 | 147 | 16 | 16 | 0 |
| 67 MB | 80 | 49 | 80 | 80 | 57 | 16 | 16 | 0 |
| 5 MB | 12 | 3 | 12 | 12 | 12 | 6 | 6 | 0 |
| FLOPPY | 4 | 4 | 4 | 4 | 5 | 4 | 4 | 0 |

Note that FASTCHEK is supplied with a default segment size increment of 16 KB.

It should be noted that (as discussed in section 4.2) the performance penalty of using less than the optimum segment size increment is not severe. Thus if multiple discs have to be processed, then it is advantageous to run multiple copies of FASTCHEK in parallel especially if the discs are on independent channels. In particular, if there are a large number of discs to be checked after a system failure, multiple copies of FASTCHEK should be used to simultaneously check discs on independent channels, and then the second (and any subsequent) discs on each channel should be checked in further parallel runs. Note that since FASTCHEK is a segmented task, multiple copies will share one copy of the code (i.e. pure segment). The fixed and removable packs in a 10 MB disc system should not be processed in parallel since these share a common head arm.

It is important to realise that when a pack is Initialized, specifying a pre-allocated Directory of sufficient size to contain all files to be allccated is critical in achieving high performance during Check functions. It should also be noted that because the Directory check phase is extremely fast, there is almost no penalty in specifying Close mode for a Check function, since if there are no Indexed files open for write the operation will complete almost immediately, but very little time is lost if a mode switch to NOREADCHECK is required.

## 4.4 PACK ADMINISTRATION FILE

The Pack Administration file, PACKINFO.DIR/0, contains both a list of the defective sectors on the pack and a record of the administrative history of the pack. Its primary function is to provide the defective sector information so that when a pack is Initialized or Checked and the Bit Map has to be rebuilt, this can be done without performing a surface check to find the defective sectors.

The file is a Contiguous file of some 9 sectors for hard discs (and only one sector on Floppy discs since no administration history is maintained on these discs). The file is protected against deletion and update by normal application tasks by maintaining the Directory entry with a Write Count of -1. The file is organised as a set of 64 byte records packed 4 to a sector. The first record is a control record containing global information and pointers to the data records which are either history records or Defective sector records.

The history records record the following events:
- pack initialization and mode of initialization
- pack name set by rename or initializing (last four times)
- surface check performed (last four times)
- Check/Close or Check/Clcseonly performed
- File integrity check performed

Note that additional types of history records are supported by other utilities.

The Pack Administration file is created when FASTCHEK is used to Initialize a pack in the FILL or READCHECK mode. It will also be created if the NOREADCHECK mode is specified since a mode switch to READCHECK will occur if the file does not exist. Hence the administration history records only this and subsequent events. Thus packs should be Initialized in NOREADCHECK mode so as to preserve any prior history.

Since the file is created at Initialization time, it is always the first file in the Directory and always occupies the first error free area of the required size (which will, in general, directly follow the Volume Descriptor).

All Check and Rename operations performed by FASTCHEK will be recorded in the Pack Administration file provided that it exists. The existence and validity checking are performed as follows: First, an existence check is made by checking whether PACKINFO.DIR exists as the first entry in the Directory and is a Contiguous file. If not, the program assumes that no Pack Administration file exists on the pack, and the message

```
  WARNING: PACK ADMINISTRATION FILE PACKINFO.DIR NOT FOUND
```

is printed.

The validity of the file is then checked by first checking that the data pointers in the control record are in nondecreasing order and that the last data pointer is equal to (filesize*4-1). The validity of the contents of the Defective Sector record(s) are then checked as follows: the addresses of the defective sectors are checked to be in ascending order and to be greater than zero and less than or equal to the maximum sector address for the given type of pack. The first address found to be zero is assumed to flag the end of the list, and subsequent addresses are checked to be zero. If the Defective Sector record(s) are full, then the number of defective sectors as held in the Control record and the latest Surface Check History record are checked to be equal and greater than or equal to the number in the Defective Sector record(s). If the Defective Sector record(s) are not full, these three values must be equal. If any of these checks fail, a warning message is output as follows:

```
  WARNING: PACK ADMINISTRATION FILE PACKINFO.DIR CORRUPTED
```

The message

```
  WARNING: PACK ADMINISTRATION FILE PACKINFO.DIR OVERFLOWED
```

will be output if the Defective sector records are full and the number of addresses recorded is less than the count of defective sectors held in the Control record.

If an Unrecoverable I/O error (status X'84') occurs while the file is being accessed, the message

```
  WARNING: PACK ADMINISTRATION FILE PACKINFO.DIR UNUSABLE
```

is output.

If the file exists and is valid, the current system date/time is checked to be later than the 'last updated date/time' held in the Control record. If this check fails, the utility pauses after issuing the message

```
PACK ADMINISTRATION FILE LAST UPDATED ON mm/dd/yy hh:mm:ss
ADJUST SYSTEM DATE/TIME IF REQUIRED, THEN CONTINUE
```

On being continued, the program will use the current system date/time (thus allowing the operator the correct the date/time if it is incorrectly set).

If the user desires to examine the Pack Administration file, it can be dumped using the DISPLAY command of OS/32 COPY. The dump can then be interpreted using the record layouts given in Appendix D.

# CHAPTER 5
# ERROR HANDLING AND MESSAGES

## 5.1 COMMAND ERROR HANDLING

The action taken when a command error is detected depends on the mode of command entry being used.

If either the immediate or batch command entry mode is being used, the erroneous command line is displayed on the system console (or MTM terminal) together with an error message and an indication of where in the line the error occurred. For example, if the START command

```
  ST ,CHECK=DSC1:,ROADCHECK
```

is used, it will result in the following messages

```
  Unrecognizable Keyword
  CHECK=DSC1:,ROADCHECK
              ^
```

and the program will then terminate with end of task code 2.

Similarly, if the program is started specifying a file as the command device and that file contains the following commands

```
  CHECK DSC1:
  READCHECK
  WRITEREC; EXTEND; KEEPSPADE
  END
```

then the following messages will be displayed

```
  Unrecognizable Keyword
  WRITEREC; EXTEND; KEEPSPADE
                    ^
```

and the program will terminate with end of task code 3.

If the interactive command entry mode is used, an error in response to a prompt will cause an error message to be displayed on the interactive terminal followed by the erroneous response and an indication of the position of the error. For example, if in response to the prompt

```
  Mode (#NOReadcheck, READcheck, or Fill=xxxxxxxx) ?
```

the user replies with

```
  REEDCHECK
```

then the messages

```
  Unrecognisable Keyword
  REEDCHECK
  ^
  Mode (#NOReadcheck, REAdcheck, or Fill=xxxxxxxx) ?
```

will be displayed and the user should enter the correct response.

If batch or immediate command entry is used, then once all the individual commands have been checked, a final consistency check of the commands is made. If this fails then the first detected error will be reported on the system console (or MTM terminal) and the program will terminate with end of task code 4. For example, if the following START command is used

```
  ST ,CLOSE,LI=PR2:,EXTEND
```

this will result in the following message

```
  Function (Check/Initialize/Rename) not Specified
```

Two other types of errors can occur during the command processing phase. First, the program may be unable assign the specified command device, and second, an I/O error may occur on the command device.

If the command device specified in the START command cannot be assigned, the message

```
  ASSIGN ERROR xxlu FOR voln:filename.ext/act
    <error description>
```

is displayed on the system console (or MTM terminal) and the program then terminates with end of task code 2.

If an I/O error occurs on the command device, the message

```
  I/O ERROR ON LU nn voln:filename.ext/act FUNC=xx
      STATUS=xxxx   <error description>
```

is displayed on the system console (or MTM terminal). If the interactive command entry mode is being used, the program will pause, and on being continued will retry the errored I/O. If batch command entry is being used, the program will terminate with end of task code 3.

## 5.2 LIST OUTPUT AND ERROR HANDLING

No output to the list device occurs until all commands have been processed and validated. In addition, if a file is specified as the list device, all output will be appended after any existing data in the file.

If the specified list device is a printer (i.e. has device code X'7x'), then the program will output the heading

```
  PERKIN-ELMER OS/32 FASTCHEK Rnn-nn      mm/dd/yy
                                               hh:mm:ss  PAGE nnn
```

in one line at the top of each page. Note that a maximum of 55 lines of data is output to each page.

If the list device is not a printer and is not the same device as the command device, then no pagination is performed and the heading

```
  PERKIN-ELMER OS/32 FASTCHEK Rnn-nn          mm/dd/yy  hh:mm:ss
```

is only output once.

If the list device is assigned to the same interactive device as the command device, then no pagination is performed and no heading is output.

After the heading is printed, a message is output giving the operation to be performed and any options. These messages are identical to the confimatory messages used in the interactive command entry mode and are shown in sections 3.2.1, 3.2.2, and 3.2.3.

If an I/O error occurs on the list device, the message

```
  I/O ERROR ON LU nn voln:filename.ext/act FUNC=xx
      STATUS=xxxx   <error description>
```

is displayed on the system console (or MTM terminal), or on the interactive command device (if this is not the same as the list device). The program then pauses, and on being continued, retries the errored I/O.

## 5.3 MESSAGE SUMMARY

This section documents all messages generated by the program. The messages are given in alphabetical order. Note that no explicit operator action is required unless specifically stated.

For the sake of clarity, the messages are shown in upper case with any parameters shown in lower case although the actual messages are displayed in upper and lower case. Thus for example, the message displayed as

```
  33 Temporary files deleted
```

is documented as

```
  nnnn TEMPORARY FILES DELETED
```

In addition, in order to avcid repetitious definitions, certain conventions are used in specifying the parameters. These are as follows:

- parameters shown as xxxx, yyyy, or zzzz represent hexadecimal values

- parameters shown as llll, mmmm, or nnnn represent decimal values

- the parameter "voln:filename.ext/act" or "filename.ext/act" represents the name of a file.

The messages are documented in alphabetical order in the following pages. Note that any message starting with a filename or a value is ordered by the first word of the message.

Note also that on the actual listing the action messages are prefixed by an asterisk. In contrast, the diagnostic messages which describe the problem are indented. Thus a typical listing will have the following appearance:

```
    SYSTEM.DIR/000 Open for Write (Count FFFF)
  * File SYSTEM.DIR/000 Closed - Possible Lost Data
    MTMATF./255 Open for Write (Count FFFF)
  * File MTMATF./255 Extended from 0 to 190 Records
    MTMACCT./255 Open for Write (Count 0001)
    BATCH.QUE/000 Open for Write (Count FFFF)
    SPL.QUE/000 Open for Write (Count FFFF)
  * File MTMACCT./255 Closed - Possible Lost Data
  * File BATCH.QUE/000 Closed - Possible Lost Data
  * File SPL.QUE/000 Closed - Possible Lost Data
```

!GO NOT ALLOWED YET

Meaning:

The special !GO response has been entered during the interactive command entry but is not yet valid because all the mandatory data has not been entered.

```
**** ABNORMAL TERMINATION - CODE nnn ****
```

where nnn is the end of task code.

Meaning:

This message is appended to the end of the listing (provided that the list and command devices are not the same) when the program terminates with an end of task code greater than 1.

```
**** CHECK/NOREADCHECK IS REQUIRED ****
```

Meaning:

This message is output when a Check/Closeonly operation terminates because an Indexed, Nonbuffered Indexed, or Extendable Contiguous file is found which is open for write, or some other error condition (indicated in the previous message) is detected which requires a Check/Noreadcheck operation to restore the integrity of the pack.

```
**** OPTION REPORTONLY SET ****
**** ACTION MESSAGES ARE ADVISORY ONLY ****
```

Meaning:

This message is output at the head and tail of the listing when the REPORTONLY option is set for a Check operation to remind the user that any indicated actions (e.g., FILE ... DELETED) are advisory only.

```
**** SWITCHING TO <newmode> MODE ****
```

where <newmode> gives the new mode of operation and will be either NOREADCHECK or READCHECK.

Meaning:

An error (indicated by the previous message) has forced a switch to a more extensive checking mode.

```
**** VOLUME ATTRIBUTES ON-LINE BIT SET - CHECK REQUIRED ****
```

Meaning:

A Rename function was attempted on a pack whose Volume Descriptor indicated that the pack had not previously been properly marked off. The pack must be checked by FASTCHEK before it can be renamed.

Program Action:

The program terminates with end of task code 1.

```
AGED SPOOL FILE filename.ext/act DELETED
```

where filename.ext/act is the name of the file.

Meaning:

The aged Spool file was deleted. This message is only output if the REPORTONLY option is set.

nnnn AGED SPOOL FILES DELETED

where nnnn — is the number of files deleted.

Meaning:

The indicated number of Spool files older than 24 hours have been deleted.

filename.ext/act - ALLOCATION CONFLICT AT SECTOR xxxxxx

where xxxxxx — is the sector address.

Meaning:

The file occupies a sector which is already in use by either the Bit Map, the Directory, or another data file.

```
ASSIGN ERROR xxll FOR voln:filename.ext/act
  <error description>
```

where xx is the SVC 7 error code

ll is the logical unit number

voln:filename.ext/act is the file descriptor of the device being assigned.

<error description> will be one of the following:

| Error Code | Description |
|---|---|
| 1 | ILLEGAL FUNCTION |
| 2 | ILLEGAL LU |
| 3 | VOLUME/DEVICE NOT PRESENT |
| 4 | FILE DOES NOT EXIST |
| 6 | PROTECTED BY KEYS |
| 7 | PRIVILEGE CANNOT BE GRANTED |
| 8 | INSUFFICIENT SYSTEM SPACE |
| 9 | LU ALREADY ASSIGNED OR DEVICE OFFLINE |
| B | INVALID FILE NAME |
| C | TRAP GENERATING DEVICE |
| D | ACCOUNT VIOLATION |
| 80-FF | I/O ERROR |
| E-7F | UNKNOWN ERROR |

Meaning:

The specified device cannot be assigned for the indicated reason.

Program Action:

If the command device is being assigned, the program will terminate with end cf task code 2. If the device is either the list or disc device, then if the interactive command entry mode is being used, the appropriate prompt will be reissued; otherwise the program will terminate.

BIT MAP RELOCATED TO xxxxxx THROUGH yyyyyy

| | | |
|---|---|---|
| where | xxxxxx | is the start address of the Bit Map |
| | yyyyyy | is the end address of the Bit Map. |

Meaning:

The Bit Map has been relocated to the indicated sector addresses. This message only occurs if the original Bit Map was found to contain a defective sector and had to be relocated.

BIT MAP CONTAINS DEFECTIVE SECTOR AT xxxxxx
IN SECTOR nnnn OF BIT MAP

| | | |
|---|---|---|
| where | xxxxxx | is the sector address |
| | nnnn | is the number of the sector (base 0) within the Bit Map. |

Meaning:

The surface check performed during a Check/Readcheck operation has located a defective sector within the area of the Bit Map. See also section 4.1.2.2.

```
BIT MAP CONTAINS RECOVERED DEFECTIVE SECTOR AT xxxxxx
    IN SECTOR nnnn OF BIT MAP
```

where xxxxxx    is the sector address

nnnn is the number of the sector (base 0) within the Bit Map.

Meaning:

The surface check performed during a Check/Readcheck operation has located a defective sector which was recovered within the area of the Bit Map. This is only an informatory message since Bit Map is about to be completely rebuilt.

```
                   {CLOSE      }
                   {           }
 CHECK devn:  MODE={CLOSEONLY  }
                   {           }
                   {NOREADCHECK}
                   {           }
                   {READCHECK  }

    [EXTENDALLOWED] [WRITERECOVERY] [KEEPSPOOL] [REPORTONLY]
```

where devn:     is the device mnemonic of the disc device

Meaning:

This message is output when all commands have been validated to indicate the function about to be performed.

CHECK COMPLETE - VOLUME voln READY TO BE MARKED ON

where voln — is the name of the pack being checked.

Meaning:

This message is output when a Check/Close, Check/Noreadcheck, or Check/Readcheck operation (without the REPORTONLY option set) terminates sucessfully. The pack is then ready for normal use.

filename.ext/act CONTAINS DEFECTIVE SECTOR AT xxxxxx

where xxxxxx — is the sector address.

Meaning:

The file contains a sector found to be defective.

filename.ext/act CONTAINS RECOVERED DEFECTIVE SECTOR AT xxxxxx

where xxxxxx — is the sector address.

Meaning:

The file contains a sector found to be defective but which was recovered.

```
CURRENT BIT MAP DIFFERS FROM EXPECTED
    nnn BITS STARTING AT xxxxxx <message>
```

| | |
|---|---|
| where xxxxxx | gives the equivalent sector address at which a string of nnn bits all differ in the same sense. |
| &lt;message&gt; | is either "SET - EXPECTED RESET" or "RESET - EXPECTED SET" |

Meaning:

The new Bit Map built during a Check operation with the REPORTONLY option set is not the same as that currently on the pack. See also section 4.1.2.5.

```
CURRENT/PREVIOUS DEFECTIVE SECTOR DISCREPANCIES
    xxxxxx  (CHS=ccc/hh/ss)  NOW {GOOD     }
                                 {DEFECTIVE}
```

| | |
|---|---|
| where xxxxxx | is the sector address |
| ccc | is the hexadecimal cylinder number |
| hh | is the hexadecimal head number |
| ss | is the hexadecimal sector number. |

Meaning:

The surface check performed during a Check/Readcheck operation has located different defective sectors to those recorded in the Pack Administration file. The second message is repeated for each discrepancy. If there are no discrepancies then this second line is replaced by "**** NONE ****".

DATA POSSIBLY CORRUPTED BEGINNING AT RECORD nnnn

| | |
|---|---|
| where nnnn | is the record number base 0. |

Meaning:

As indicated by the previous message, an Indexed, Nonbuffered Indexed, or Extendable Contiguous file has been found to contain a recovered defective sector. This message gives the number of the first record that might contain corrupted data.

DATA VALIDATION ERROR IN BIT MAP AT SECTOR xxxxxx

| | |
|---|---|
| where xxxxxx | is the address of the Bit Map sector in which the error occurred. |

Meaning:

A hardware failure is indicated.

Program Action:

The program will terminate with end of task code 30.

DEFECTIVE SECTOR AT xxxxxx (CHS=ccc/hh/ss) <message>

| | |
|---|---|
| where xxxxxx | is the sector address |
| ccc | is the hexadecimal cylinder number |
| hh | is the hexadecimal head number |
| ss | is the heaxdecimal sector number |
| <message> | will be one of:<br>RECOVERED - indicating that the sector was recovered through the WRITERECOVERY option,<br>STATUS yyyy READING indicating that status yyyy was returned by a read operation<br>STATUS yyyy WRITING indicating that status yyyy was returned by a write operation. |

Meaning:

A defective sector has been found at the indicated address. The word RECOVERED, if present, indicates that the sector has been recovered through the WRITERECOVERY option.

nnnn DEFECTIVE SECTORS FOUND

| | |
|---|---|
| where nnnn | is the total number of defective sectors. |

Meaning:

The surface check operation has located the indicated number of defective sectors on the pack.

```
nnnn DEFECTIVE SECTORS RECORDED
    xxxxxx  (CHS=ccc/hh/ss)
```

| | |
|---|---|
| where nnnn | is the number of defective sectors recorded in the Pack Administration file |
| xxxxxx | is the sector address |
| ccc | is the hexadecimal cylinder number |
| hh | is the hexadecimal head number |
| ss | is the hexadecimal sector number. |

Meaning:

The pack being checked or initialised has, according to the information read from the Pack Administration file, the indicated defective sectors. The second message is repeated for each defective sector. If the pack has no defective sectors then only the first line is displayed and the count is shown as zero. Note that these messages are displayed only if NOREADCHECK mode is specified. In READCHECK mode, the recorded defective sector data is not explicitly displayed since it can be inferred from the comparison of the current and previous defective sectors.

```
nnnn DEFECTIVE SECTORS RECOVERED
```

| | |
|---|---|
| where nnnn | is the total number of defective sectors recovered by the WRITERECOVERY option. |

Meaning:

The surface check operation has recovered the indicated number of defective sectors on the pack.

DEFECTIVE SECTOR LIST OVERFLOW AFTER nnnnnn FOUND

| | |
|---|---|
| where nnnnnn | is the number of defective sectors found. |

Meaning:

The in memory defective sector list has overflowed after the indicated number has been found. See also sector 4.1.1.2.

Program Action:

The program will terminate with end of task code 22.

DEVICE devn: IS A DISC - INVALID AS COMMAND OR LIST DEVICE

| | |
|---|---|
| where devn: | is the device name specified as the command or list device. |

Meaning:

The device shown was specified (as indicated in the following message) as the command or list device and it is a disc. This is not valid.

DEVICE devn: IS NOT A DISC

| | |
|---|---|
| where devn: | is the device specified to be Checked, Initialized, or Renamed. |

Meaning:

The indicated device is not a disc.

DIRECTORY AS SPECIFIED (nnn FILES / CYLINDER mmm) OVERFLOWS PACK

| | |
|---|---|
| where nnn | is the requested size of the Directory in terms of the number of files |
| mmm | is the requested start cylinder for the Directory. |

Meaning:

The Directory as requested will not fit on the pack.

DIRECTORY TRUNCATED TO nnnn BLOCKS AT SECTOR xxxxxx

| | |
|---|---|
| where nnnn | is the number of remaining directory blocks (base 1) |
| xxxxxx | is the sector address of the block |

Meaning:

The Directory has been truncated at the indicated point and this is now the last Directory block.

DIRECTORY BLOCK AT xxxxxx CHAINS TO ALLOCATED SECTOR yyyyyy

| | | |
|---|---|---|
| where | xxxxxx | is the sector address of the Directory block |
| | yyyyyy | is the address of the next Directory block |

Meaning:

The forward pointer in the Directory block (which gives the address of the next Directory block) contains a sector address which is allocated either to the Bit Map, a previous Directory block, or a data file.

Program Action:

The Directory will be truncated as indicated by the message following. See also section 4.1.2.2.

DIRECTORY BLOCK AT xxxxxx CHAINS TO DEFECTIVE SECTOR yyyyyy

| | | |
|---|---|---|
| where | xxxxxx | is the sector address of the Directory block |
| | yyyyyy | is the address of the next Directory block |

Meaning:

The forward pointer in the Directory block (which gives the address of the next Directory block) contains a sector address which is known to be a defective sector.

Program Action:

The Directory will be truncated as indicated by the message following. See also section 4.1.2.2.

DIRECTORY BLOCK AT xxxxxx HAS INVALID FORWARD POINTER yyyyyyyy

where xxxxxx is the sector address of the Directory block

yyyyyyyy is the address of the next Directory block

Meaning:

The forward pointer in the Directory block (which contains the address of the next Directory block) is greater than the maximum sector address on the pack.

Program Action:

The Directory will be truncated as indicated by the message following. See also section 4.1.2.2.

EMPTY SPOOL FILE filename.ext/act DELETED

where filename.ext/act is the name of the file.

Meaning:

The empty Spool file was deleted. This message is only output if the REPORTONLY option is set.

nnnn EMPTY SPOOL FILES DELETED

where nnnn is the number of files deleted.

Meaning:

The indicated number of empty Spool files have been deleted.

FILE filename.ext/act EXTENDED FROM nnnn TO mmmm RECORDS

| | |
|---|---|
| where nnnn | is the number of previously checkpointed records. |
| mmmm | is the number of records now in the file. |

Meaning:

An Indexed, Nonbuffered Indexed, or Extendable Contiguous file open for write was found to extend beyond its previously checkpointed extent and was extended (through the EXTENDALLOWED option) to the point indicated.

FILE filename.ext/act CLOSED - POSSIBLE LOST DATA

where filename.ext/act is the name of the file.

Meaning:

The file was open for write and has been closed. Note that the words "Possible Lost Data" will be omitted if the file, in fact, contains no data.

FILE filename.ext/act DELETED

where filename.ext/act is the name of the file.

Meaning:

The file has been deleted for the reason given in the previous message.

FILE filename.ext/act MAY CONTAIN ERRONEOUS DATA

where filename.ext/act is the name of the file.

Meaning:

The file may contain erroneous data because (as indicated by the previous message) the file contained a recovered defective sector.

FILE filename.ext/act READ COUNT RESET

where filename.ext/act is the name of the file.

Meaning:

The file was open for read and has been closed. This message is only output if the REPORTONLY option is set.

nnnn FILES CLOSED

where nnnn is the number of files closed.

Meaning:

The indicated number of files was closed during the Check operation.

nnnn FILES DELETED

where nnnn    is the number of files deleted.

Meaning:

The indicated number of files was deleted during the Check operation. Note that this does not include the count of Temporary and aged Spool files deleted which is given separately.

nnnn FILES EXTENDED

where nnnn    is the number of files extended.

Meaning:

The indicated number of Indexed, Nonbuffered Indexed, and Extendable Contiguous files open for write has been extended beyond their previously checkpointed extent.

nnnn FILES TRUNCATED

where nnnn    is the number of files truncated.

Meaning:

The indicated number of Indexed, Nonbuffered Indexed, and Extendable Contiguous files open for write were truncated to their previous checkpoint.

```
filename.ext/act - FIRST INDEX BLOCK ADDRESS xxxxxxxx
                            EQUALS LAST - SHOULD NOT
```

where xxxxxxxx is the address of the first index block.

Meaning:

The first and last index block addresses of an Indexed, Nonbuffered Indexed, or Extendable Contiguous file are | equal, but they should not be because the file contains more data than can be mapped by one index block.

```
filename.ext/act - FIRST INDEX BLOCK ADDRESS xxxxxxxx NOT
                                         EQUAL TO LAST yyyyyyyy
```

where xxxxxxxx is the first index block address

yyyyyyyy is the last index block address.

Meaning:

The first and last index block addresses of an Indexed, Nonbuffered Indexed, or Extendable Contiguous file are | not equal but they should be because the number of data records in the file can be mapped by only one index block.

```
FUNCTION (CHECK/INITIALIZE/RENAME) NOT SPECIFIED
```

Meaning:

This message is generated during the command consistency check and indicates that no function was specified.

IN DIRECTORY BLOCK AT xxxxxx OFFSET yy

| | |
|---|---|
| where xxxxxx | is the sector address of the Directory block |
| yy | is the offset of the entry within the block. |

Meaning:

If the REPORTONLY option is set, and an error is detected in a Directory entry, this message will follow the error message to give the location of the invalid Directory entry.

IN FULLWORD AT xx EXPECTED yyyyyyyy FOUND zzzzzzzz

| | |
|---|---|
| where xx | is the address offset within the sector |
| yyyyyyyy | is the expected value of the fullword |
| zzzzzzzz | is the value actually found. |

Meaning:

This message is output after the data validation error message in order to locate the error.

IN INDEX BLOCK nnnnnn AT xxxxxx

| | |
|---|---|
| where nnnnnn | is the index block number (base 1) |
| xxxxxx | is the sector address of the index block |

Meaning:

If the REPORTONLY option is set, then when an error is detected in an Indexed, Nonbuffered Indexed, or Extendable Contiguous file, this message will follow the error message to give the location of the associated index block.

```
IN INDEX BLOCK nnnnnn AT xxxxxx OFFSET yy
```

where nnnnnn    is the index block number (base 1)

xxxxxx    is the sector address of the index block

yy    is the hexadecimal offset within the block.

Meaning:

If the REPORTONLY option is set then, when an error is detected in an index block, this message will follow the error message so as to give the location of the invalid index block.

```
                      {FILL WITH xxxxxxxx                 }
                      {                                   }
INITIALIZE devn:  MODE={READCHECK [WITH WRITERECOVERY]    }
                      {                                   }
                      {NOREADCHECK [WITH WRITERECOVERY]   }

  VOLUME voln     DIRECTORY FOR nnnn FILES AT CYLINDER mmm
                                                REQUESTED
```

where devn:    is the device mnemonic of the disc device xxxxxxxx is the data pattern for the Fill operation

voln    is the new volume name for the pack

nnnn    gives the capacity of the Directory to be allocated

mmm    is the start cylinder number of the Directory.

Meaning:

This message is output when all commands have been validated to indicate the function about to be performed.

IN SECTOR nnnn OF FILE

where nnnn is the sector number within the file.

Meaning:

If the REPORTONLY option is set and an allocation conflict or defective sector is found in a Contiguous file, this message will follow the error message to give the location within the file.

INSUFFICIENT ERROR-FREE SPACE FOR DIRECTORY

Meaning:

Insufficient error free Contiguous space is available to allocate the Directory. See also section 4.1.1.

Program Action:

The program will terminate with end of task code 20.

INSUFFICIENT SPACE FOR BIT MAP

Meaning:

Insufficient free Contiguous space is available to allocate the Bit Map. See also sections 4.1.1 and 4.1.2.2.

Program Action:

The program will terminate with end of task code 21.

INSUFFICIENT WORKSPACE FOR DUPLICATE BIT MAP

Meaning:

Insufficient workspace, either in memory or on a Contiguous Temporary file, is available to build a duplicate copy of the Bit Map. See also section 4.1.2.2.

Program Action:

The program will terminate with end of task code 5.

```
INTERNAL FAILURE rrnn AT xxxxxx  STACK PTR ssssssss
  yyyyyyyy  zzzzzzzz
  yyyyyyyy  zzzzzzzz
```

|  |  |
|---|---|
| where rr | is the major internal failure code |
| nn | is the minor internal failure code |
| xxxxxx | is the address at which the failure occured |
| ssssssss | is the current value of the workspace stack pointer |
| yyyyyyyy | and zzzzzzzz give the walkback information. |

Meaning:

An internal failure has occurred (see Chapter 6).

Program Action:

The program will pause, and then on being continued, will terminate with end of task code 251.

Required Operator Action:

See Chapter 6.

```
I/O ERROR ON LU nn voln:filename.ext/act FUNC=xx RAND=yyyyyyyy
    STATUS=zzzz  <error description>
```

where nn is the logical unit number

voln:filename.ext/act is the file descriptor of the device assigned to the logical unit

xx is the SVC 1 function code

yyyyyyyy is the SVC 1 random address (in hexadecimal)

zzzz is the SVC 1 status code

<error description> will be one of the following:

| Status Code | Description |
|---|---|
| COxx | ILLEGAL FUNCTION |
| AOXX | DEVICE UNAVAILABLE |
| 90XX | END OF MEDIUM |
| 88XX | END OF FILE |
| 84XX | UNRECOVERABLE ERROR |
| 8281 | I/O HALTED |
| 8282 | TIMED OUT |
| 8283 | DEVICE WRITE PROTECTED |
| 8291 | REQUEST PURGED |
| 82XX | PARITY OR RECOVERABLE ERROR |
| 8100 | ILLEGAL OR UNASSIGNED LU |
| 80XX | UNKNOWN ERROR |

Meaning:

An I/O error occurred on the specified logical unit performing the specified function. Note that the "RAND=yyyyyyyy" is suppressed if random access was not being used in the function with the error.

Program Action:

If the error occurred on the command or list devices, then the actions discussed in sections 5.1 and 5.2 respectively take place.

If the error occurs on the disc device, then the action taken depends on type of error and the current phase of the program. If an Unrecoverable Error (code X'84xx') occurs, then a mode switch will occur if allowed (see Chapter 4). In all other cases the program will terminate with end of task code 10. Note that during the surface check phase, I/O errors may occur (because of defective sectors) but these will not result in the above message being logged.

Required Operator Action:

If the program pauses, remedy the fault, and continue the program.

filename.ext/act - INVALID CURRENT SECTOR ADDRESS xxxxxxxx RESET

where xxxxxxxx is the current sector address of the file.

Meaning:

A Contiguous file was found whose Directory entry contained a current sector address greater than the number of sectors in the file. The current sector address was reset to zero.

filename.ext/act - INVALID DATA BLOCK ADDRESS xxxxxxxx

where xxxxxxxx is the address of the data block.

Meaning:

The current index block contains a data block address which is invalid because it is either greater than the maximum sector address for the pack or is zero when not expected to be so.

filename.ext/act - INVALID DATA BLOCK ADDRESS xxxxxxxx
- SHOULD BE ZERO

where xxxxxxxx is the address of the data block.

Meaning:

The data block address should be zero because it is not in use because it follows the last active data block in the file. Therefore, all trailing data block addresses in the last index block of the file should be zero. This is an advisory message only and is not a reason to delete the file.

filename.ext/act - INVALID DATA BLOCK SIZE OF ZERO

Meaning:

The data block size of an Indexed, Nonbuffered Indexed, or Extendable Contiguous file is invalid because it is zero.

INVALID DEVICE NAME

Meaning:

The device name (as indicated in the message following) is invalid. That is, the program is expecting a device name (rather than a file descriptor) and the one specified is not valid.

INVALID FILE DESCRIPTOR

Meaning:

The file descriptor (as indicated in the message following) is invalid.

INVALID FILENAME ffffffffffffffff.eeeeee/act

where ffffffffffffffff.eeeeee/act is the hexadecimal representation of the filename.

Meaning:

The filename found in the Directory entry does not conform to the allowed format. The file will be deleted as indicated by the FILE ... DELETED message. In this message, any nonprintable characters will be replaced by # characters.

filename.ext/act - INVALID FILE TYPE x

where x is the hexadecimal file type code.

Meaning:

The file type code is not 0, 1, 2, or 3 (indicating Contiguous, Extendable Contiguous, Indexed, or Nonbuffered Indexed, respectively).

filename.ext/act - INVALID FIRST INDEX BLOCK ADDRESS xxxxxxxx

where xxxxxxxx is the address of the first index block of the file.

Meaning:

The address of the first index block of an Indexed, Nonbuffered Indexed, or Extendable Contiguous file is invalid because it is greater than the maximum sector address on the pack, or it is zero, but the file contains data.

filename.ext/act - INVALID FIRST SECTOR ADDRESS xxxxxxxx

where xxxxxxxx is the starting sector address of the file.

Meaning:

The starting sector address of a Contiguous file is invalid because it is either zero or greater than the maximum sector address on the pack.

filename.ext/act - INVALID INDEX BLOCK SIZE OF ZERO

Meaning:

The index block size of an Indexed file is invalid because it is zero.

filename.ext/act - INVALID LAST INDEX BLOCK ADDRESS xxxxxxxx

where xxxxxxxx is the address of the last index block of the file as given in the file's Directory entry.

Meaning:

The address of the last index block of an Indexed, Nonbuffered Indexed, or Extendable Contiguous file is invalid because it is greater than the maximum sector address on the pack, or it is zero, but the file contains data.

filename.ext/act - INVALID LAST SECTOR ADDRESS xxxxxxxx

where xxxxxxxx is the ending sector address of the file.

Meaning:

The ending sector address of a Contiguous file is invalid because it is either zero or greater than the maximum sector address on the pack.

filename.ext/act - INVALID NEXT INDEX BLOCK ADDRESS xxxxxxxx

where xxxxxxxx is the address of the next index block of the file as given by the forward pointer in the current index block.

Meaning:

The address of the next index block of an Indexed, Nonbuffered Indexed, or Extendable Contiguous file is | invalid because it is greater than the maximum sector address on the pack.

filename.ext/act - INVALID NUMBER OF LOGICAL RECORDS xxxxxxxx

where xxxxxxxx is the number of records in the file as given by the Directory entry.

Meaning:

The number of logical records is invalid because it is greater than hexadecimal 7FFFFFFF.

filename.ext/act - INVALID PREVIOUS INDEX BLOCK ADDRESS xxxxxxxx

| | |
|---|---|
| where xxxxxxxx | is the address of the previous index block of the file as given by the backward pointer in the current index block. |

Meaning:

The address of the previous index block of an Indexed, Nonbuffered Indexed, or Extendable Contuguous file is invalid because it is not equal to the actual address of the previous index block.

filename.ext/act - INVALID RECORD LENGTH nnn OR NUMBER OF RECORDS mmmmmm

| | |
|---|---|
| where nnn | is the record length of the file |
| mmmmm | is the number of logical records in the file as indicated by the Directory entry. |

Meaning:

One or both of the logical record length and the number of logical records is invalid because their product exceeds FFFFFFFF hexadecimal.

INVALID SEND MESSAGE IGNORED

Meaning:

A message sent to the task cannot be recognized.

Program Action:

The message will be ignored.

filename.ext/act - INVALID SPOOL FILE TYPE x

| | |
|---|---|
| where x | is the hexadecimal file type code. |

Meaning:

A Spool file was found whose type code was not 2 (indicating Indexed).

KEYWORD CONFLICTS WITH PREVIOUS ENTRY

Meaning:

The keyword (indicated in the following message) conflicts with one previously entered. That is, the commands being entered are not consistent.

```
filename.ext/act - LAST INDEX BLOCK ADDRESS xxxxxx
                                     SHOULD EQUAL yyyyyy
```

| | |
|---|---|
| where xxxxxx | is the address of the last index block in file as obtained from the Directory entry |
| yyyyyy | is the address of the last index block in the file as found by following down the chain of index blocks. |

Meaning:

The address of the last index block as recorded in the Directory entry does not match that calculated by following down the file.

```
filename.ext/act - LAST SECTOR ADDRESS xxxxxx LESS THAN
                                                FIRST yyyyyy
```

| | |
|---|---|
| where xxxxxx | is the ending sector address of the file |
| yyyyyy | is the starting sector address of the file. |

Meaning:

The ending sector address of a Contiguous file is invalid because it is less than the starting sector address of the file.

```
LIST DEVICE ASSIGNED TO devn:
```

| | |
|---|---|
| where devn: | is the name of the System Console (or MTM Terminal). |

Meaning:

No list device was specified and an attempt was made to assign to the device PR:. However, this failed, and the list device was assigned to the console.

```
filename.ext/act - NEXT INDEX BLOCK ADDRESS xxxxxxxx
                                               SHOULD BE ZERO
```

| | |
|---|---|
| where xxxxxxxx | is the address of the next index block of the file as given by the forward pointer in the current index block. |

Meaning:

The current index block should be the last in the file. Thus, its forward pointer should be zero, but has the value indicated.

NO DEFAULT ALLOWED

Meaning:

A default (null) response was given to an interactive mode prompt to which no default is allowed.

filename.ext/act OPEN FOR READ (COUNT xxxx)

where xxxx is the read count for the file.

Meaning:

The file is open for read. It will be closed as indicated by the next message. Note that this message is output only if the REPORTONLY option is set.

filename.ext/act OPEN FOR WRITE (COUNT xxxx)

where xxxx is the write count for the file.

Meaning:

The file is open for write. It may be closed or deleted as indicated by the next message.

```
OPERATING SYSTEM IS Rnn-nn - Rmm-mm REQUIRED
```

|  |  |
|---|---|
| where nn-nn | gives the revision and update level of the operating system |
| mm-mm | gives the revision and update level of the required operating system. |

Meaning:

The version of FASTCHEK being used is not compatible with the operating system being used.

Program Action:

The program will terminate with end of task code 2.

```
OS/32 FASTCHEK Rnn-nn <function> devn: STARTING
```

|  |  |
|---|---|
| where nn-nn | gives the revision and update level of the program |
| <function> | will be either INITIALIZE, CHECK, or RENAME |
| devn: | is the device name of the disc |

Meaning:

This message is logged on the system console (if it is not being used as the interactive command device) after all commands have been validated and indicates that the specified function is starting.

```
PACK ADMINISTRATION FILE LAST UPDATED ON mm/dd/yy hh:mm:ss
ADJUST SYSTEM DATE/TIME IF REQUIRED, THEN CONTINUE
```

| | |
|---|---|
| mm/dd/yy hh:mm:ss | is the date/time on which the Pack Administration file was last updated. Note that the date is given in the form dd/mm/yy if the operating system is sysgened with European date format. |

Meaning:

The current system date/time is earlier than the date/time at which the Pack Administration file (PACKINFO.DIR) was last updated.

Program Action:

The program will pause to allow the operator to correct the system date/time if required.

Required Operator Action:

Continue the program after updating the system date/time if necessary.

```
PACK INITIALISED - PREALLOCATED DIRECTORY AT xxxxxx
                                 BIT MAP AT yyyyyy
```

| | |
|---|---|
| where xxxxxx | is the sector address of the first Directory Block |
| yyyyyy | is the address of the sector at which the Bit Map starts. |

Meaning:

This message is output when the pack has been sucessfully initialised and gives the location of the Directory and Bit Map.

PACK ON devn: RENAMED FROM vold TO voln

| | |
|---|---|
| where devn: | is the device name of the disc drive |
| vold | is the previous volume name |
| voln | is the new volume name. |

Meaning:

The Rename operation has been successfully completed and the pack on the indicated drive has been renamed to the name shown. The task will then terminate with end of task code 0.

PARAMETER TOO LARGE

Meaning:

The parameter (indicated in the messgae following) is too large. But the START command also contained other used commands.

PERKIN-ELMER OS/32 FASTCHEK Rnn-nn

| | |
|---|---|
| where nn-nn | gives the revision and update level of the program |

Meaning:

This message is logged on the system console (or MTM terminal) immediately the program starts and before any attempt is made to process any start commands.

## POSSIBLE LOOP IN DIRECTORY CHAIN

Meaning:

The number of Directory blocks encountered during a Check/Closeonly or Check/Close operation has exceeded one eighth of the number of sectors on the pack. If the current mode is CLOSE a mode switch to NOREADCHECK will occur, otherwise the program will terminate with end of task code 1.

## RENAME devn: AS voln

|  |  |
|---|---|
| where devn: | is the device mnemonic of the disc device |
| voln | is the volume name to which the pack is to be renamed. |

Meaning:

This message is output once all commands have been validated to indicate the function about to be performed.

## nnnn SPOOL FILES PRESENT

|  |  |
|---|---|
| where nnnn | is the number of files remaining. |

Meaning:

The indicated number of Spool files remain on the pack.

TEMPORARY FILE filename.ext/act DELETED

where filename.ext/act is the name of the file.

Meaning:

The Temporary file was deleted. This message is only output if the REPORTONLY option is set.

TEMPORARY FILE ENCOUNTERED

Meaning:

A Temporary file was encountered during a Check/Close operation forcing a switch to NOREADCHECK mode (indicated by the following message).

nnnn TEMPORARY FILES DELETED

where nnnn is the number of files deleted.

Meaning:

The indicated number of Temporary files were deleted.

filename.ext/act TRUNCATED TO CHECKPOINT AT RECORD nnnn

where nnnn is the number of records remaining in the file.

Meaning:

| An Indexed, Nonbuffered Indexed, or Extendable Contiguous file open for write was found to extend beyond its previously checkpointed extent and was truncated back to its previous checkpoint.

UNEXPECTED CHARACTER

Meaning:

The character (indicated in the message following) was detected when a delimiter was expected.

UNEXPECTED TASK QUEUE ENTRY xxxxxxxx IGNORED

where xxxxxxxx is the unexpected task queue entry fullword.

Meaning:

The task queue entry is not one of those expected.

Program Action:

The task queue entry will be ignored.

UNEXPECTED TRAILING CHARACTER

Meaning:

A keyword or a parameter following a keyword (indicated in the message following) was recognized but was followed by an unexpected character.

UNEXPECTED TRAILING CHARACTER(S) IGNORED

Meaning:

The indicated trailing characters (following a semicolon separator) were nct expected and have been ignored. This message can only occur when commands are being entered interactively.

UNRECOGNISABLE KEYWORD

Meaning:

A keyword (indicated in the message following) cannot be recognized.

VOLUME DESCRIPTOR DATA VALIDATION ERROR

Meaning:

The data read back from the Volume Descriptor did not match the data previously written to it. A hardware failure is indicated.

Program Action:

The program will terminate with end of task code 31.

```
VOLUME DESCRIPTOR ERROR
      <message>
```

where <message> will be one of the following:

```
INVALID VOLUME NAME vvoollnn

INVALID DIRECTORY POINTER pppppppp

INVALID BIT MAP POINTER pppppppp

BIT MAP (AT xxxxxx TO yyyyyy) OVERLAPS
                       DIRECTORY (AT zzzzzz)
```

| | |
|---|---|
| where vvoollnn | is the hexadecimal representation of the volume name. |
| pppppppp | is the hexadecimal start sector number of the Directory or Bit Map. |
| xxxxxx | and yyyyyy give the start and end addresses of the Bit Map. |
| zzzzzz | gives the start address of the Directory. |

Meaning:

The Volume Descriptor is invalid for the reason given in the explanatory message. Note that the Directory and Bit Map pointers are invalid if they are greater than the maximum possible address on the pack. The Bit Map pointer is also invalid if zero.

Program Action:

The program terminates with end of task code 8.

VOLUME NAME INVALID

Meaning:

The specified volume name (indicated in the message following) was not valid.

VOLUME NAME IS voln

| | |
|---|---|
| where voln | is the volume name of the pack. |

Meaning:

This message is output at the beginning of a Check operation after the Volume Descriptor was checked to indicate the name of the pack being checked.

VOLUME NAME NOT SPECIFIED

Meaning:

This message is generated during the command consistency check and indicates that no vclume name was specified for a Rename or Initialize function.

WARNING: DIRECTORY CONTAINS RECOVERED DEFECTIVE SECTOR AT xxxxxx

|  |  |
|---|---|
| where xxxxxx | is the address of the recovered defective sector. |

Meaning:

The Directory contains at the indicated address, a sector which was found to be defective and then recovered (through the WRITERECOVERY option). This message is a warning to the user that the Directory Block may have been corrupted. However, since all entries in the Block will be completely checked, there is no need to truncate the Directory.

WARNING: PACK ADMINISTRATION FILE (PACKINFO.DIR) CORRUPTED

Meaning:

The Pack Information file does not contain vaild data (see section 4.4).

Program Action:

See section 4.1.

WARNING: PACK ADMINISTRATION FILE (PACKINFO.DIR) NOT FOUND

Meaning:

The Pack Administration file does not exist as the file in the Directory.

Program Action:

See section 4.1.

WARNING: PACK ADMINISTRATION FILE (PACKINFO.DIR) NOT UPDATED

Meaning:

An I/O error occurred while the PacK Administration file was being updated (as indicated by the previous message) and as a result the file has not been updated.

WARNING: PACK ADMINISTRATION FILE (PACKINFO.DIR) OVERFLOWED

Meaning:

More defective sectors exist than could be recorded into the Pack Administration file, that is, the Defective sector records are full.

Program Action:

See section 4.1.

WARNING: PACK ADMINISTRATION FILE (PACKINFO.DIR) UNUSABLE

Meaning:

This message follows the I/O Error message when an Unrecoverable I/O error occurs while accessing the Pack Information file while checking it.

Program Action:

See section 4.1.

WARNING: SPACE NOT RELEASED IN BIT MAP

Meaning:

A Check/Closeonly operation has been run and Temporary files were deleted but their allocated space was not released in the Bit Map. A Check/Noreadcheck operation will have to be run at some convenient time to release this space.

WARNING: VOLUME NAME IS ALREADY voln

where voln is the current name of the pack.

Meaning:

The volume name specified for a Rename function is the same as the current name of the pack.

WHILE ACCESSING BIT MAP

Meaning:

This message is output after the I/O Error Message when the I/O error occurred while accessing the Bit Map.

Program Action:

If I/O error is due to a defective sector a mode switch to READCHECK will occur if allowed in the current function/mode. Otherwise the program will terminate with end of task code 10.

WHILE ACCESSING DIRECTORY

Meaning:

This message is output after the I/O Error Message when the I/O error occurred while accessing the Directory.

Program Action:

If I/O error is due to a defective sector a mode switch to READCHECK will occur if allowed in the current function/mode. Otherwise the program will terminate with end of task code 10.

WHILE ACCESSING PACK ADMINISTRATION FILE

Meaning:

This message is output after the I/O Error Message when the I/O error occurred while accessing the Pack Administration file. It will be followed by the warning message that the file is unusable.

Program Action:

See section 4.1.

WHILE ACCESSING VOLUME DESCRIPTOR

Meaning:

This message is output after the I/O Error Message when the I/O error occurred while accessing the Volume Descriptor.

Program Action:

The program will terminate with end of task code 10.

# CHAPTER 6
# INTERNAL FAILURE

## 6.1 DESCRIPTION

In certain conditions FASTCHEK may detect some error or internal inconsistency from which it cannot recover. When this happens FASTCHEK will pause after printing the following message on the system console (or MTM terminal):

```
    INTERNAL FAILURE rrnn AT xxxxxxx  STACK PTR ssssssss
      yyyyyyyy  zzzzzzzz
      yyyyyyyy  zzzzzzzz
      ........  ........
```

| | |
|---|---|
| where rr | is the major internal failure code and identifies the area of failure |
| nn | is the minor internal failure code and identifies the specific failure |
| xxxxxxxx | is the address at which the failure occurred |
| ssssssss | is the current value of the workspace stack pointer |
| yyyyyyyy | and zzzzzzzz give the walkback information. |

If FASTCHEK is then continued, it will terminate with end of task code 251. The task pauses rather than terminating directly so as to enable the user to dump any task related information, for example by using the Display Registers or Examine commands.

If an Internal Failure should occur, the user is requested to contact the nearest Perkin-Elmer support office. The following information will assist the diagnosis of the fault:

- a copy of the system console log covering the period in which the failure occurred

- a copy of any listing output produced

- a memory map of the system

- a dump of the impure segment of the task
  (This is most simply produced by using the EXAMINE command to dump the required area. Alternatively a Panic Dump of all memory can be made.)

Interal Failure conditions are, by their nature, not due to any operator error and it is highly likely that rerunning the task with the same parameters will result in the same failure.

# APPENDIX A
# FASTCHEK COMMAND SUMMARY

```
BLCCKS [=] [bbb] [/[ccc]]

CHECK [=] devn:

CLOSE

CLOSEONLY

COMMAND [=] fd

DIRECTCRY [fff] [/[ccc]]

END

EXTENDALLOWED

FILL [=] [xxxxxxxx]

INITIALISE [=] devn:

INITIALIZE [=] devn:

KEEPSPOOL

LIST [=] fd

NOREADCHECK

NOWRITERECOVERY

READCHECK

RENAME [=] devn:

REPORTONLY

VOLUME [=] voln

WRITERECOVERY
```

# APPENDIX B
# END OF TASK CODES

The End of Task codes used by FASTCHEK are given below.

| EOT Code | Meaning |
|---|---|
| 0 | Normal completion - pack can now be marked on and used normally. |
| 1 | Pack requires further checking - will occur if:<br>- a Check/Closeonly operation cannot close all open files<br>- the REPORTONLY option is used<br>- a Rename operation detects that the pack has not been marked off. |
| 2 | Error in Start arguments or incompatible Operating System Revision Level. |
| 3 | Error in command read from Batch command file. |
| 4 | Inconsistency in specified commands. |
| 5 | Insufficient memory or workspace. |
| 8 | Invalid Volume Descriptor. |
| 10 | Fatal I/O error on disc. |
| 20 | Insufficient space for Directory. |
| 21 | Insufficient space for Bit Map. |
| 22 | Defective sector list overflow. |
| 30 | Data validation error in Bit Map or Directory. |
| 31 | Data validation error in Volume Descriptor. |
| 250 | STOP message received or !STOP response to interactive command mode prompt. |
| 251 | Internal Failure. |

# APPENDIX C
# DEVICE CHARACTERISTICS

The following tables give pertinent characteristics for the various types of discs. Note that the table on page C-3 contains discs, which although usable under the current Operating System release, are no longer current products.

| DISC TYPE | 5 MB FIXED | 5 MB REMOV | MSM80 REMOV | MSM300 REMOV | FLOPPY | MSM80F + HPT | MSM330 FIXED |
|---|---|---|---|---|---|---|---|
| Nominal Capacity | 5 MB | 5 MB | 67 MB | 256 MB | 250 KB | 68.5MB | 268 KB |
| Device Code (Hex) | 32 | 33 | 35 | 36 | 37 | 38 | 2C |
| (Decimal) | 50 | 51 | 53 | 54 | 55 | 56 | 44 |
| Rotation Time (sec) | 1/40 | 1/40 | 1/60 | 1/60 | 1/60 | 1/60 | 1/160 |
| Number of Cylinders | 408 | 408 | 823 | 823 | 77 | 842.2 | 1024 |
| Number of Heads (i.e. tracks\|cyl) | 2 | 2 | 5 | 19 | 1 | 5 | 16 |
| Cylinder size (KB) | 12 | 12 | 80 | 304 | 3.25 | 80 | 256 |
| (sectors) | 48 | 48 | 320 | 1216 | 13 | 320 | 1024 |
| Track size (KB) | 6 | 6 | 16 | 16 | 3.25 | 16 | 16 |
| (sectors) | 24 | 24 | 64 | 64 | 13 | 64 | 64 |
| Bit Map size (KB) | 2.50 | 2.50 | 32.35 | 122.25 | 0.25 | 33.00 | 122.25 |
| (sectors) | 10 | 10 | 129 | 489 | 1 | 132 | 489 |
| Total size (KB) | 4896 | 4896 | 65840 | 250192 | 250.25 | 67376 | 262144 |
| (sectors) | 19584 | 19584 | 263360 | 1000768 | 1001 | 269504 | 1048576 |
| Default Directory size (files) | 120 | 120 | 640 | 1600 | 5 | 640 | 1600 |
| (blocks) | 24 | 24 | 128 | 320 | 1 | 128 | 320 |
| Directory Alloc'n Interleaving Factor | 6 | 6 | 32 | 32 | 1 | 32 | 32 |

| DISC TYPE | HPT of MSM80F | MSM80 FIXED | MCCD32 REMOV | MCCD32 FIXED | MCCD64 FIXED | MCCD96 FIXED |
|---|---|---|---|---|---|---|
| Nominal Capacity | 1.5 MB | 67 MB | 13.5MB | 13.5MB | 40 MB | 67 MB |
| Device Code (Hex) | 39 | 3A | 3B | 3C | 3D | 3E |
| (Decimal) | 57 | 58 | 59 | 60 | 61 | 62 |
| Rotation Time (sec) | 1/60 | 1/60 | 1/60 | 1/60 | 1/60 | 1/60 |
| Number of Cylinders | 19.2 | 820 | 823 | 821 | 821 | 821 |
| Number of Heads (i.e. tracks\|cyl) | 5 | 5 | 1 | 1 | 3 | 5 |
| Cylinder size (KB) | 80 | 80 | 16 | 16 | 48 | 80 |
| (sectors) | 320 | 320 | 64 | 64 | 192 | 320 |
| Track size (KB) | 16 | 16 | 16 | 16 | 16 | 16 |
| (sectors) | 64 | 64 | 64 | 64 | 64 | 64 |
| Bit Map size (KB) | 0.75 | 32.25 | 6.5 | 6.5 | 19.25 | 32.25 |
| (sectors) | 3 | 129 | 26 | 26 | 77 | 129 |
| Total size (KB) | 1536 | 65600 | 13168 | 13136 | 39408 | 65680 |
| (sectors) | 6144 | 262400 | 52672 | 52544 | 157632 | 262720 |
| Default Directory size (files) | 20 | 640 | 320 | 320 | 320 | 640 |
| (blocks) | 4 | 128 | 64 | 64 | 64 | 128 |
| Directory Alloc'n Interleaving Factor | 32 | 32 | 32 | 32 | 32 | 32 |

| DISC TYPE | 2.5 MB FIXED | 2.5 MB REMOV | 40 MB REMOV |
|---|---|---|---|
| Nominal Capacity | 2.5 MB | 2.5 MB | 40 MB |
| Device Code (Hex) | 30 | 31 | 34 |
| (Decimal) | 48 | 49 | 52 |
| Rotation Time (sec) | 1/25 | 1/25 | 1/40 |
| Number of Cylinders | 203 | 203 | 406 |
| Number of Heads (i.e. tracks\|cyl) | 2 | 2 | 20 |
| Cylinder size (KB) | 12 | 12 | 100 |
| (sectors) | 48 | 48 | 400 |
| Track size (KB) | 6 | 6 | 5 |
| (sectors) | 24 | 24 | 20 |
| Bit Map size (KB) | 1.25 | 1.25 | 20.00 |
| (sectors) | 5 | 5 | 80 |
| Total size (KB) | 2436 | 2436 | 40600 |
| (sectors) | 9744 | 9744 | 162400 |
| Default Directory size (files) | 60 | 60 | 400 |
| (blocks) | 12 | 12 | 80 |
| Directory Alloc'n Interleaving Factor | 4 | 4 | 5 |

# APPENDIX D
# PACK ADMINISTRATION FILE FORMAT

## D.1 INTRODUCTION

This Appendix describes the format of each type of record in the Pack Administration file, PACKINFC.DIR/0. Note that this file is a contiguous file in which each sector contains four 64-byte records.

## D.2 Control Record

The Control record is the first record in the file and thus occupies bytes 0 through 63 of sector 0. It contains a time stamp which contains the date/time at which the file was last updated and a number of data record pointers.

The time stamp is used by the utilities which update PACKINFO.DIR to perform a partial validity check on the current system date/time so as to ensure that the time stamps on the history records are correct. The data record pointers (of which there is one for each type of data record) each consist of a halfword which contains the ending record number of the data record(s) of the given type.

The layout of the control record is as follows:

| Byte | Contents/Meaning |
|---|---|
| 0-1 | Record type indicator (always set to GP) |
| 2-7 | Date/Time last updated in form yymmddhhmmss, where each field (e.g. dd=day) occupies 1 byte |
| 8-9 | Halfword count of number of defective sectors on pack |
| 10-11 | reserved for future soft defective sector counts |
| 12-13 | reserved for future soft defective sector counts |
| 14-15 | reserved for future soft defective sector counts |
| 16-17 | dummy "zercth" data pointer (always 0) |
| 18-19 | data pointer for Initialisation (type=IN) history record |
| 20-21 | data pointer for Check/Close (type=CL) history record |
| 22-23 | data pointer for File Integrity Check (type=CF) history record |
| 24-25 | data pointer for Name (type=NA) history records |
| 26-27 | data pointer for Surface Check (type=SC) history records |

| | |
|---|---|
| 28-29 | data pointer for Incremental Backup (type=BI) history records |
| 30-31 | data pointer for Selective Backup (type=BS) history records |
| 32-33 | data pointer for Restore (type=RS) history records |
| 34-35 | data pointer for Full Backup (type=BF) history records |
| 36-61 | reserved for future use by data pointers (up to 13) |
| 62-63 | data pointer for defective sector list records |

Note that because the data pointers contain the ending record number of each set of data records, the data pointers for unused types of data records always contain the same value of the previous data record pointer. In addition, since on a floppy disc, there are no history records, all the data pointers except that for the defective sector records contain zero.

## D.3 DEFECTIVE SECTOR RECORD

Each Defective Sector Record contains up to 16 fullword defective sector addresses. Note that since sector 0 on the pack can never be defective, a defective sector address of zero in the Defective Sector records acts as a null entry.

The Defective Sector records are allocated by FASTCHEK when the pack is initialized using the INITIALIZE/FILL or INITIALIZE/READCHECK operations. On hard discs, FASTCHEK creates PACKINFO.DIR with sufficient Defective Sector records to allow a minimum expansion factor of 128 defective sector addresses. That is, if during the surface check made during pack initialisation, N defective sectors are found, then (N+128-1)/16+1 Defective Sector records are allocated in PACKINFO.DIR. Note that in the case of a floppy disc, three Defective Sector records are always allocated, and thus on a floppy disc the maximum number of defective sectors recorded by PACKINFO.DIR is 48.

The Defective Sector records are written whenever a surface check operation is performed. Thus, if FASTCHEK is used to perform a CHECK/READCHECK operation, the Defective Sector records will be effectively cleared and rewritten containing the addresses of the defective sectors found during the surface check.

The Defective Sector addresses are always held in ascending address order.

## D.4 HISTORY RECORDS

The history records record the administrative history of the pack and each record has a fixed format for the first 38 bytes of the record. Bytes 0 and 1 indicate the record type, bytes 2 through 37 contain an extended time stamp which includes the date and time of the activity, the device mnemonic of the disc drive, the

O/S licence number, the O/S version ID, and the revision and update number of the utility which made the entry.

Currently the types of history records are as follows:

| Type Code | Meaning |
|---|---|
| IN | pack initialized |
| NA | pack name set |
| SC | surface check performed |
| CL | Check/Close performed |
| CF | File integrity check performed |
| BF | full pack backup performed |
| BI | incremental pack backup performed (i.e. using SINCE or BEFORE option but with no other file selection) |
| BS | selective pack backup performed |
| RS | pack restore performed |

Note that whereas only one IN, CL, CF and BF record is ever present in the file, up to 4 NA, SC, BI, BS and RS type records are held. Note also that all Backup and Restore operations refer to those performed by OS/32 FASTBACK.

The layout of the extended date/time stamp is as follows:

| Byte | Contents/Meaning |
|---|---|
| 2-7 | Date/Time at which entry made in form yymmddhhmmss where each field (e.g., dd=day) occupies 1 byte |
| 8-11 | Device mnemonic of disc drive on which pack was mounted at the time at which the entry was made |
| 12-27 | Licence number of O/S |
| 28-35 | Version of O/S (as specified to OS/32 CUP or OS/32 SYSGEN) |
| 36-37 | Revision and update level of utility which made the entry in form rruu, with rr and uu occupying 1 byte each |

## D.4.1 Initialization History Record

The Initialization History Record records the last time the pack was initialized by FASTCHEK. The record contains the extended date/time stamp (see section D.4), the preallocated directory parameters (number of blocks and starting cylinder), and the initialization mode (i.e. FILL, WRITERECOVERY, or neither).

The record layout is as follows:

| Byte | Contents/Meaning |
|---|---|
| 0-1 | Record type indicator (always set to IN) |
| 2-37 | Extended date/time stamp |
| 38-39 | Number of blocks in preallocated directory |

|  |  |
|---|---|
| 40-41 | Requested starting cylinder of preallocated directory |
| 42-43 | Two character indicator of initialisation type - set to "FI" if FILL specified, "WR" if WRITERECOVERY specified, otherwise spaces |
| 44-47 | FILL fullword if FILL specified, else zero |
| 48-51 | Directory start sector address |
| 52-63 | not used - always zero |

## D.4.2 Name History Records

The Name History Records record the most recent and up to three immediately previous pack naming operations since PACKINFO.DIR was created. The pack name is set (and thus the history is updated) whenever FASTCHEK is used to perform an INITIALIZE or RENAME operation.

Each Name History record contains the extended date/time stamp (see section D.4) and the volume name of the pack as set by the naming operation. Note that unused Name History Records always contain binary zeros.

The record layout is as follows:

| Byte | Contents/Meaning |
|---|---|
| 0-1 | Record type indicator (always set to NA) |
| 2-37 | Extended date/time stamp |
| 38-47 | not used - always zero |
| 48-51 | Volume Name |
| 52-63 | not used - always zero |

## D.4.3 Surface Check History Records

The Surface Check History Records record the most recent and up to three immediately previous surface check operations since PACKINFO.DIR was created. A surface check is performed (and thus the history is updated) whenever FASTCHEK is used to perform an INITIALIZE/FILL, INITIALIZE/READCHECK or CHECK/READCHECK operation.

Each Surface Check History record contains the extended date/time stamp (see section D.4) and a count of the number of defective sectors on the pack. Note that unused Surface Check History Records always contain binary zeros.

The record layout is as follows:

| Byte | Contents/Meaning |
|---|---|
| 0-1 | Record type indicator (always set to SC) |
| 2-37 | Extended date/time stamp |
| 38-51 | not used - always zero |
| 52-53 | number of defective sectors |

| | |
|---|---|
| 54-55 | reserved for future use |
| 56-57 | reserved for future use |
| 58-59 | reserved for future use |
| 60-63 | not used - always zero |

### D.4.4 Close History Record

The Close History Record records the last time FASTCHEK was used to perform a CHECK/CLOSE operation on the pack. The record contains the extended date/time stamp (see section D.4) and the Close mode (i.e., CLOSE or CLOSEONLY).

The record layout is as follows:

| Byte | Contents/Meaning |
|---|---|
| 0-1 | Record type indicator (always set to CL) |
| 2-37 | Extended date/time stamp |
| 38-59 | not used - always zero |
| 60 | One character indicator of Close mode<br>- set to "O" if CLOSEONLY mode used, else space |
| 61-63 | not used - always zero |

### D.4.5 File Integrity Check History Record

The File Integrity Check History Record records the last time FASTCHEK was used to perform a CHECK/NOREADCHECK or CHECK/READCHECK operation on the pack. The record contains the extended date/time stamp (see section D.4) and the mode (i.e., NOREADCHECK or READCHECK).

The record layout is as follows:

| Byte | Contents/Meaning |
|---|---|
| 0-1 | Record type indicator (always set to CF) |
| 2-37 | Extended date/time stamp |
| 38-59 | not used - always zero |
| 60 | One character indicator of mode<br>- set to "N" if NOREADCHECK or "R" if READCHECK |
| 61-63 | not used - always zero |

### D.4.6 Full Backup History Record

NOTE: THIS TYPE OF HISTORY RECORD IS NOT SUPPORTED BY REVISION 0 OF OS/32 FASTBACK.

The above note is necessary because FASTBACK and FASTCHEK use the same PACKINFO.DIR file. Some records are implemented and updated by FASTCHEK and some by FASTBACK.

The Full Backup History Record records the last time OS/32 FASTBACK was used to perform a full backup of the pack. The record contains the extended date/time stamp (see section D.4), the device name of the mag tape drive on which the backup was written and the Tape Serial Number written on the tape.

The record layout is as follows:

| Byte | Contents/Meaning |
|---|---|
| 0-1 | Record type indicator (always set to BF) |
| 2-37 | Extended date/time stamp |
| 38-53 | not used - always spaces |
| 54-57 | Device Mnemonic of tape drive on which backup was written |
| 58-63 | Tape Serial Number (6 character ASCII) |

| To reduce the possibility of errors in a data transfer to magnetic tape, the recommended blocking factors are:

| TAPE RECORDING DENSITY | BLOCKING FACTOR |
|---|---|
| 800 BPI | 12.5K |
| 1600 BPI | 25.0K |
| 6250 BPI | 100.0K |

| Using larger blocking factors than those recommended gains little additional storage space and results in an insignificant reduction in processing time. However, it does increase the probability of data transfer errors resulting in verify errors.

## D.4.7 Incremental Backup History Records

NOTE: THIS TYPE OF HISTORY RECORD IS NOT SUPPORTED BY REVISION 0 OF OS/32 FASTBACK.

The Incremental Backup History Record records the most recent and up to three immediately previous times since PACKINFO.DIR was created that OS/32 FASTBACK was used to perform a incremental backup of the pack. That is, a backup in which the SINCE or BEFORE option was used but no other file selection was specified. The record contains the extended date/time stamp (see section D.4), the device name of the mag tape drive on which the backup was written, the Since/Before date/time, and the Tape Serial Number written on the tape.

The record layout is as follows:

| Byte | Contents/Meaning |
|---|---|
| 0-1 | Record type indicator (always set to BI) |
| 2-37 | Extended date/time stamp |
| 38 | Since/Before option indicator - set to ">" if Since was specified or "<" if Before was specified |
| 39-44 | Date limit in character format as yymmdd |
| 45-46 | space characters |
| 47-52 | Time limit in character format as hhmmss |
| 53 | space character |
| 54-57 | Device Mnemonic of tape drive on which backup was written |
| 58-63 | Tape Serial Number (6 character ASCII) |

## D.4.8 Selective Backup History Records

NOTE: THIS TYPE OF HISTORY RECORD IS NOT SUPPORTED BY REVISION 0 OF OS/32 FASTBACK.

The Selective Backup History Record records the most recent and up to three immediately previous times since PACKINFO.DIR was created that OS/32 FASTBACK was used to perform a selective backup of the pack. The record contains the extended date/time stamp (see section D.4), the device name of the mag tape drive on which the backup was written, the file descriptor of the select file, and the Tape Serial Number written on the tape.

The record layout is as follows:

| Byte | Contents/Meaning |
|---|---|
| 0-1 | Record type indicator (always set to BS) |
| 2-37 | Extended date/time stamp |
| 38-53 | File Descriptor of Select File (in packed format) or if the Immediate Select facility was used, up to 16 characters of the select entry starting with the character "(" and containing the closing ")" if the select entry is less than 15 characters |
| 54-57 | Device Mnemonic of tape drive on which backup was written |
| 58-63 | Tape Serial Number (6 character ASCII) |

D.4.9 Restore History Records

The Restore History Record records the most recent and up to three immediately previous times since PACKINFO.DIR was created that OS/32 FASTBACK was used to restore files onto the pack from a backup tape. The record contains the extended date/time stamp (see section D.4), the file descriptor of the select file, the volume name of the pack which was backed up to produce the backup tape, and the Tape Serial Number written on the tape.

The record layout is as follows:

| Byte | Contents/Meaning |
|---|---|
| 0-1 | Record type indicator (always set to RS) |
| 2-37 | Extended date/time stamp |
| 38-53 | File Selection parameter - set to<br>a) spaces if no file selection is specified; or if Select Criteria was read from a Device, e.g., FOX2: or CON:. This is because the 16-byte "Selection" field of the Restore History Record is too short to hold full file descriptors, and therefore, volume and device names are omitted from this field.<br>b) the File Descriptor of the Select File (in packed format); or<br>c) if the Immediate Select facility was used, up to 16 characters of the select entry starting with the character "(" and containing the closing ")" if the select entry is less than 15 characters; or<br>d) if the Since/Before option was used (and no other file was selected) the Date/Time limit in the format described in section D.4.7 |
| 54-57 | Volume Name of pack backed up to produce backup tape |
| 58-63 | Tape Serial Number (6 character ASCII) |

# APPENDIX E
# LINK PROCEDURE

FASTCHEK is supplied both in task image and object form. If the user wishes to establish FASTCHEK then OS/32 Link must be used and the required Link commands are given below.

```
ESTABLISH TASK
DCMD
MAP
INCLUDE FASTCHEK.OBJ
BUILD FASTCHEK.TSK
END
```

Note that the required options are set using Link commands imbedded in the FASTCHEK object file and are set as follows.

```
ABS=0
SYSSPACE=FFFFF
WORK=(4000,B0000)
SEGMENTED
DISC
ACP
NROLL
LU=9
IOBLOCKS=1
UT
NFLOAT
NDFLOAT
```

Link will give a warning message because the Absolute space is less than 100, and on completion, will terminate with end of task code 2.

# APPENDIX F
# LOGICAL UNIT USAGE

The logical unit assignments used by FASTCHEK are given in the following table.

| Logical Unit | Access Privilege | Usage |
|---|---|---|
| 0 | SWO | Journal listing output |
| 1 | ERW | Disc to be Initialized, Checked or Renamed (note that access privilege ERO is used if the REPORTONLY option is selected) |
| 3 | SWO | List device |
| 4 | ERW | Temporary file for Bit Map (only used if duplicate Bit Map is required and cannot be built in memory) |
| 5 | SRW | Command device (note that access privilege SRO is used if the command device is not interactive) |

# APPENDIX G
# COMPARISON WITH OS/32 DISCHECK AND OS/32 DISCINIT

## G.1 INTRODUCTION

OS/32 FASTCHEK is a functional replacement for both OS/32 DISCHECK and OS/32 DISCINIT. More specifically, the Check function of FASTCHEK replaces DISCHECK and the Initialize and Rename functions replace DISCINIT.

## G.2 CHECK FUNCTION

The following table gives the equivalent FASTCHEK command for each of the DISCHECK START parameters. Note that because of FASTCHEK's automatic mode switching feature, the NOREADCHECK modes are not strictly equivalent.

| DISCHECK Parameter | FASTCHEK Command |
|---|---|
| dev: | CHECK [=] devn: |
| list fd | LIST [=] fd |
| READCHECK | READCHECK |
| NOREADCHECK | NOREADCHECK |
| CLOSE | CLOSEONLY |

Thus, the following Start commands are equivalent.

```
DISCHECK:   ST ,DSC1:,PR:,READCHECK
      or    ST ,DSC1:,PR:
FASTCHEK:   ST ,CHECK=DSC1:,LIST=PR:,READCHECK

DISCHECK:   ST ,DSC1:,PR:,NOREADCHECK
FASTCHEK:   ST ,CHECK=DSC1:,LIST=PR:,NOREADCHECK
      or    ST ,CHECK=DSC1:,LIST=PR:,CLOSE
      or    ST ,CHECK=DSC1:,LIST=PR:

DISCHECK:   ST ,DSC1:,PR:,CLOSE
FASTCHEK:   ST ,CHECK=DSC1:,LIST=PR:,CLOSEONLY
```

## G.3 INITIALIZE FUNCTION

The following table gives the equivalent FASTCHEK command for each of the DISCINIT Start parameters when used to Initialize a pack.

| DISCINIT Parameter | FASTCHEK Command |
|---|---|
| DISC=dev: | INITIALIZE [=] devn: |
| CLEAR | implied by INITIALIZE |
| VOLUME=voln | VOLUME [=] voln |
| BLOCKS=n[/m] | BLOCKS [=] [n] [[/]m] |
| FILL=byte | FILL [=] [xxxxxxxx] |

Thus, the following Start commands are equivalent.

```
 DISCINIT:   ST ,DISC=DSC1:,CLEAR,VOLUME=SYS,BLOCKS=100/1,
                                                   FILL=BD
 FASTCHEK:   ST ,INITIALIZE=DSC1:,LIST=CON:,VOLUME=SYS,
                                  BLOCKS=100/1,FILL=BDBDBDBD

 DISCINIT:   ST ,DISC=DSC1:,CLEAR,VOLUME=SYS
 FASTCHEK:   ST ,INITIALIZE=DSC1:,LIST=CON:,VOLUME=SYS,
                                                  BLOCKS=0
```

## G.4 RENAME FUNCTION

The following table gives the equivalent FASTCHEK command for each of the DISCINIT Start parameters when used to Rename a pack.

| DISCINIT Parameter | FASTCHEK Command |
|---|---|
| DISC=dev: | RENAME [=] devn: |
| VOLUME=voln | VOLUME [=] voln |

Thus, the following Start commands are equivalent.

```
 DISCINIT:   ST ,DISC=DSC1:,VOLUME=SYS
 FASTCHEK:   ST ,RENAME=DSC1:,LIST=CON:,VOLUME=SYS
```

# APPENDIX H
# JOURNAL FEATURE

FASTCHEK has a Journal facility, which when invoked, causes diagnostic information to be output to the device or file assigned to logical unit 0. This diagnostic information is designed to assist the analyst who is diagnosing and maintaining FASTCHEK itself, the operating system, or the hardware.

The Journal feature is invoked by assigning logical unit 0. This can be done either before starting FASTBACK or after pausing the program by using the SEND PAUSE facility. (Note that if FASTCHEK is paused using the operating system or MTM PAUSE command, then the Journal feature will not be activated because FASTCHEK only checks the assignment of logical unit zero when it is started or is continued after receiving a PAUSE message.)

The Journal feature can be turned off by using SEND PAUSE to pause the program and then closing logical unit 0.

The Journal output is self-explanatory and is not documented here. Note that it is best to assign logical unit 0 to the device or file used as the list device.

The Journal feature should not be invoked for the everyday use of FASTCHEK since it will result in the output of very large amounts of data which are of no use to the normal user.

All Journal output is prefixed by an exclamation ("!") character and, thus, is easily distinguished from the normal list output. The Journal output is also fixed format. That is, the variable data is always output in fixed size fields. These two features aid in processing the output by the Editor or some other program.

# APPENDIX I
# COMPATABILITY WITH CTHER PRODUCTS

## I.1 16-BIT SYSTEMS

FASTCHEK can be used to Initialize packs for use by the OS/16 Operating System and also to Check and Rename packs used by this operating system. However, any pack processed by FASTCHEK must be marked on with the "NEW" when returned to the 16-bit system.

In addition (in common with CS/32 DISCHECK), FASTCHEK does not support files with account numbers greater than 255. Thus if FASTCHEK is used to check a pack containing files allocated under OS/16 with account numbers over 255, then these files will be deleted because they appear to have illegal file names.

## I.2 OS/32 DISCINIT

FASTCHEK can be used to Check or Rename packs initialized by OS/32 DISCINIT. Note however, that since DISCINIT does not record the Defective Sector information in the Pack Administration file, any Check function (other than in the CLOSEONLY mode) will be performed in READCHECK mode.

## I.3 OS/32 DISCHECK

DISCHECK can be used to check a pack which was previously initialized by FASTCHEK. However, the Pack Administration file, PACKINFO.DIR, will always appear to be open for write since FASTCHEK maintains this file with a write count of -1 (to protect it against deletion and update). DISCHECK will thus always "unprotect" this file by resetting its write count. However, whenever FASTCHEK is used to check a pack, it will set the write count for PACKINFO.DIR back to -1 (provided that the file is valid).

Note that FASTCHEK should always be used in preference to DISCHECK because of FASTCHEK's better performance.

## I.4 NON-STANDARD DISC DEVICES

FASTCHEK can be used to maintain discs supported by user written or non-standard drivers provided that the drivers obey the following conventions:

- the Device Control Block (DCB) is set up using the standard PDCB and DDCB structures

- the device attributes and flags are as for standard disc devices

- the device code used does not conflict with the standard disc device codes - in particular, the device codes given in Appendix C cannot be reused

- the driver supports the standard SVC 1 function codes in the same manner as the standard disc drivers

- the driver uses the standard device independent status codes - note also that a status of X'8283' returned for a write function is taken to indicate that the drive is hardware write protected

# INDEX

## PUBLICATION COMMENT FORM

We try to make our publications easy to understand and free of errors. Our users are an integral source of information for improving future revisions. Please use this postage paid form to send us comments, corrections, suggestions, ect.

1. Publication number____________________________________

2. Title of publication____________________________________

3. Describe, providing page numbers, any technical errors you found. Attach additional sheet if neccessary.

   ____________________________________________

   ____________________________________________

   ____________________________________________

4. Was the publication easy to understand? If not, why?

   ____________________________________________

5. Were illustrations adequate? ____________________________

   ____________________________________________

   ____________________________________________

6. What additions or deletions would you suggest? ________________

   ____________________________________________

   ____________________________________________

7. Other comments: ____________________________________

   ____________________________________________

   ____________________________________________

From __________________________ Date ____________________

Position/Title ______________________

Company ______________________

Address ______________________

______________________

______________________

STAPLE STAPLE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO. 22 OCEANPORT, N.J.

POSTAGE WILL BE PAID BY ADDRESSEE

**PERKIN-ELMER**

**Data Systems Group**
106 Apple Street
Tinton Falls, NJ 07724

**ATTN:**
**TECHNICAL SYSTEMS PUBLICATIONS DEPT.**

FOLD FOLD

STAPLE STAPLE