Burroughs Machines Ltd
Software Engineering
Livingston, Scotland
Date: 03 OCT 1986

CLASS CODE-4-0501

REV LTR	ISSUE DATE	DESCRIPTION	PREPARED BY	APPROVED BY
A	20 JUN 86	S40 System Software Disk Subsystem Design Specification	D. McLaren G. Steel D. McGugan S. Laxton	
B	03 OCT 86		S. Laxton D. McLaren	

The information contained within this document is confidential and proprietary to Burroughs Corporation.

%*COPYRIGHT**
%*   %*
%*   TITLE: S40 SYSTEM SOFTWARE DISK DESIGN SPEC   %*
%*   %*
%*   FILE ID: /d2/s40/doc/syst/stos/dspec/revb   %*
%*   %*
%*   PROPRIETARY PROGRAM MATERIAL   %*
%*   %*
%* This material is proprietary to BURROUGHS CORPORATION   %*
%* and is not to be reproduced, used or disclosed except   %*
%* in accordance with program license or upon written   %*
%* authorization of the patent division BURROUGHS   %*
%* CORPORATION, DETROIT, MICHIGAN 48232, USA.   %*
%*   %*
%* COPYRIGHT BURROUGHS CORPORATION 1986   %*
%*   %*
%* The within specification is not intended to be nor   %*
%* should such be construed as an affirmation of fact,   %*
%* representation or warranty by BURROUGHS CORPORATION   %*
%* of any type, kind, or character. The within product   %*
%* and the related materials are only furnished pursuant   %*
%* and subject to the terms and conditions of a duly   %*
%* executed license agreement.   %*
%* The only warranties made by BURROUGHS with respect to   %*
%* the products described in this material are set forth   %*
%* in the above mentioned agreement.   %*
%*   %*
%* The customer should exercise care to assure that use   %*
%* of the materials will be in full compliance with laws,   %*
%* rules and regularities of the jurisdictions with   %*
%* respect to which it is used.   %*
%*   %*
%*COPYRIGHT**

The information contained within this document is confidential
and proprietary to Burroughs Corporation.

CHANGE INFORMATION

Revision B of the specification was the first released version of the document on the third of October 1986.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

OUTSTANDING ISSUES

None

The information contained within this document is confidential and proprietary to Burroughs Corporation.

RELATED SPECIFICATIONS

1.	MTOS - 86 UP/MP USERS GUIDE Industrial Programming Inc.	(APRIL 1983)
2.	S40 System Software Functional Specification (Standard Version)	3679 2851
3.	S40 Disk Interface Specification	3704 8055
4.	S40 Disk Unit Hardware Functional Specification	3703 4592
5.	S40 Product specification	3678 5947
6.	MTOS 86UP/MP Writing a Device Driver	(APRIL 1983)
7.	MTOS 86UP/MP Design Documentation	(FEBRUARY 1983)
8.	Z8030/Z8530 SCC Serial Communications Controller Technical Manual	(JANUARY 1983)

The information contained within this document is confidential and proprietary to Burroughs Corporation.

1. INTRODUCTION

The purpose of this document is to describe the design of the S40 disk subsystem. This subsystem provides the interface between the application task and the physical disk.

It has been assumed in this document that the reader is familiar with all material stated above.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

2. OVERVIEW

The subsystem is composed of a number of 'C' modules and Intel 8086/88 assembler modules which interact in the manner illustrated by figure 1.

The modules of the subsystem are described in detail in the sections which follow. However, the subsystem has been designed using the following guidelines:

- The Disk Access Selection Control module passes requests to the Access Method Management Routines and Disk Access Routines. Each access method has a defined contiguous range of up to 32 function codes. Requests are directed to the correct module by performing a range check on the function code value supplied. Hexadecimal codes 00 - 1F are reserved for the Disk Access Routines. Thus the control task can handle up to seven different access methods.

- Each access method management routines module has a corresponding dummy module, which enables the user to customise the operating system to support only those access methods which are required. This minimises the size of the disk subsystem. A dummy module reports a DK_FN_NOT_IMPLEMENTED error, if any request is passed to it. (See appendix A for details)

- Each module has a single entry point to which the function code and parameter block are passed. This has the advantages of minimising the number of public symbols declared and minimising the size of dummy modules.

- The disk device driver has no intelligence built in with respect to the disk, it is only required to deal with the datacomm link level interface.

- The Control task modules will be designed in such a way that the Utility interface, and the output functionality will be conditionally compiled, so that unused functions can be compiled out of the loader system, and the utility functions can be compiled out of the application system.

- The utility function module is a module which can be inserted if required to perform low level utility functions. This module will be different for each utility. A utility module can make use of a set of low level functions which cannot be accessed directly via pio and pio-compl functions.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

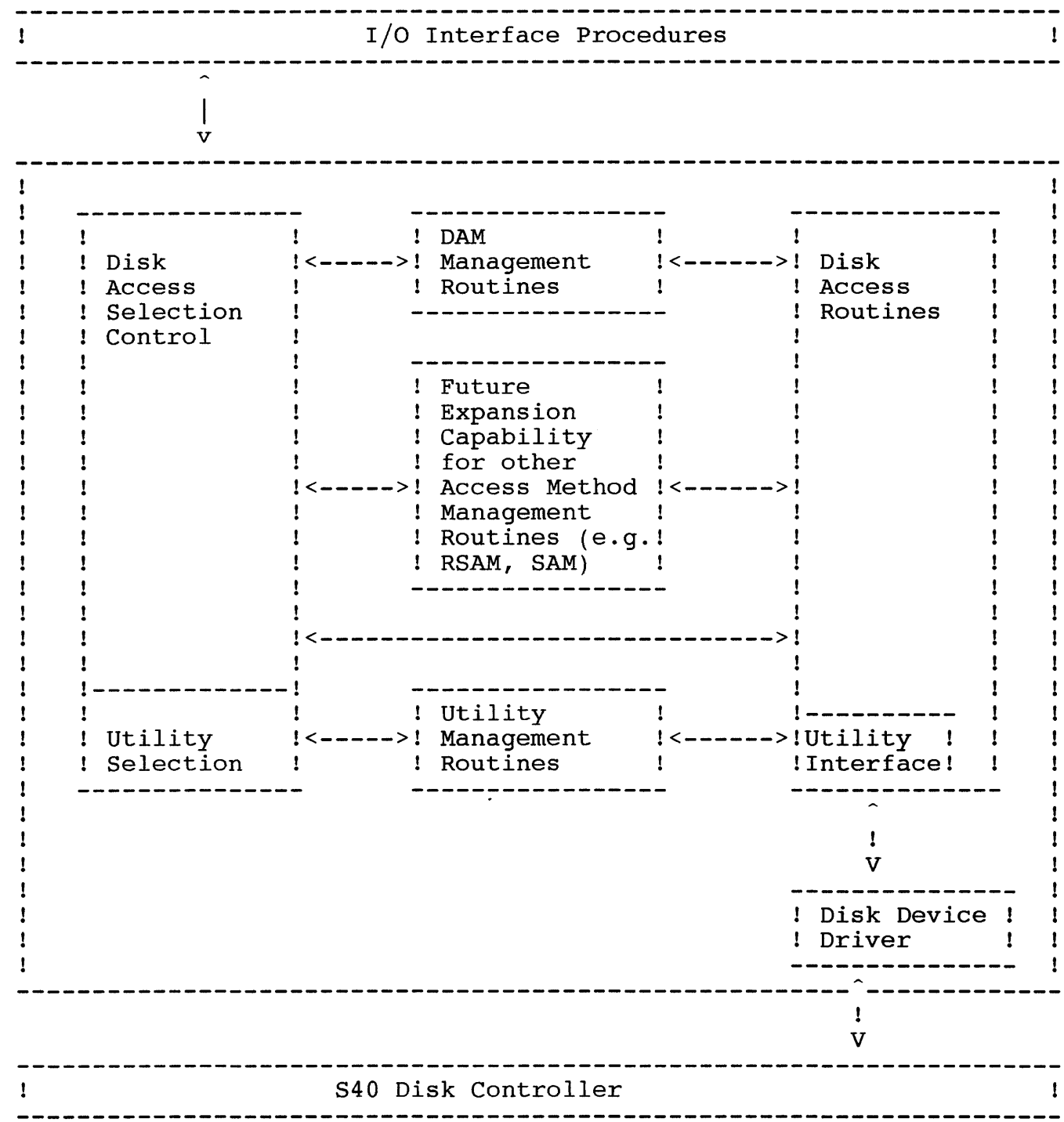


Figure 1. Disk Subsystem - High Level Modular Structure

The information contained within this document is confidential and proprietary to Burroughs Corporation.

3. B25 BTOS COMPATITILITY REQUIREMENTS FOR S40

In order for the S40 to create and maintain a B25 BTOS compatible disk the S40 must take into account a number of factors:

- Creation and maintenance of the main disk control structures.
- Creation, and if necessary maintenance of system files on the disk. The system files required are :
  - Sysimage.sys
  - Mfd.sys
  - BadBlk.sys
  - CrashDump.sys
  - Log.sys
  - FileHeaders.sys

The disk control structures are described in detail, in the section regarding Disk Access Routines, below. The system files are discussed in the subsections which follow.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

3.1 SYSTEM FILES

3.1.1 SYSIMAGE.SYS

The Sysimage.sys file is used by the B25 to contain a bootstrap operating system. The only compatibility requirement that the S40 must meet is the ability for the Initialise/Format utility to be able to create a null file.

3.1.2 MFD.SYS

The Mfd.sys file is the Master File Directory of the disk. The S40 will be required to support this file in full, in order to operate the disk.

3.1.3 BADBLK.SYS

The BadBlk.sys file contains a record of all bad blocks found on the disk at initialisation time. The S40 will be required to create this file.

3.1.4 CRASHDUMP.SYS

The CrashDump.sys file is used by the B25 when performing memory dump after a system failure. The only compatibility requirement that the S40 must meet is the ability for the Initialise/Format utility to be able to create a null file.

3.1.5 LOG.SYS

The Log.sys file is used by the B25 as a system event log, where it logs various key events, and exception conditions. The only compatibility requirement that the S40 must meet is the ability for the Initialise/Format utility to be able to create a null file.

3.1.6 FILEHEADERS.SYS

The FileHeaders.sys file contains all the file headers which are used in the structure of the disk. The S40 must maintain this file in order to operate the disk. The File header block for this file must be the first file header in the FileHeaders.sys file.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

4. DISK ACCESS SELECTION CONTROL

The Disk Access Selection Control (DASC) serves the basic functions of, disk device activation, servicing the disk subsystem mail box including routing of requests, and returning completion status.

4.1 DEVICE ACTIVATION

Immediately upon receipt of the first IO request after S40 power up, and after certain error situations, the DASC must perform the actions required to activate the disk drive, and acquire the information necessary for the control task to operate the device.

If the device is successfully activated then the applications' request is processed.

If no device is present, or the device is not successfully activated then the control task reports DK_NOT_PRESENT as byte 2 of the return status (See appendix A for details).

4.2 MAIL BOX SERVICING

The DASC waits until a mail box message is received, checks which range the function code falls in, and then passes the request to the appropriate module. The function code ranges currently supported are:

Range (hexadecimal)	Module
00 to 1F	Disk Access Routines and utility functions
20 to 3F	DAM Management Routines
40 to 5F	Utility Management Routines
60 to FF	Reserved for expansion.

TABLE 1. Function Code Ranges

The information contained within this document is confidential and proprietary to Burroughs Corporation.

4.3 RETURNING COMPLETION STATUS

Each module returns a four byte status to the DASC on completion of each function request, which is then returned to the application via the I/O interface procedures. The status bytes returned should be interpreted in the following general manner (See appendix A for specific details):

Byte 1 : Primary status as defined in ref 2.

Byte 2 : Disk subsystem error code, used as a general error type indicator. This error code normally originates from the Disk Access Routines.

Byte 3 and 4 : Module error code, used to give more specific error information. This error code is normally generated by the module which received the request.

The application can use as much, or as little of the return status as is necessary to resolve the error.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5. DISK ACCESS ROUTINES

The Disk Access Routines module is the main module of the control task. It contains the interface to the disk driver, and the volume, directory, and file control functions. It performs conversion of logical file addresses (lfa) to physical file addresses (pfa). It maintains the disk structures, enforces disk security, and maintains references to those files which are currently open on the disk.

In order to simplify the structure of the Disk Access Routines they are split in to two logical units, the User Access Functions, and the Disk Control Functions. The Disk Control Functions maintain the Disk Control Structures for each disk, and the User Access Functions provide the low level application interface.

5.1 DISK CONTROL STRUCTURES

There are 10 structures maintained on the disk, which are used during the location, and positioning of data on the disk. These are:

- Volume Home Block (Vhb).
- Sector Allocation Bit Map (Abm).
- Bad Block File (BadBlk.sys).
- Master file directory (Mfd.sys).
- Directory Entry Block (Deb).
- Directory.
- File Entry Block (Feb).
- File Headers File (Fileheaders.sys)
- File Header Block (Fhb).
- Disk Extent

The following subsections, describe the disk control structures and how they relate to each other.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.1 VOLUME HOME BLOCK

The Volume Home Block (Vhb) is the 'root' structure of the disk. There are two copies of the Vhb. The initial Vhb, which is set up when the disk is initialised, or when the volume name or password are changed, and the working Vhb which is set up on initialisation of the disk and updated as the disk is accessed. The initial Vhb is always located at pfa zero. The working Vhb is located on a different track, at a different rotational point on the disk, dependent on the type of disk. The working Vhb is accessed via the lfaVhb field of the initial Vhb (See figure below).

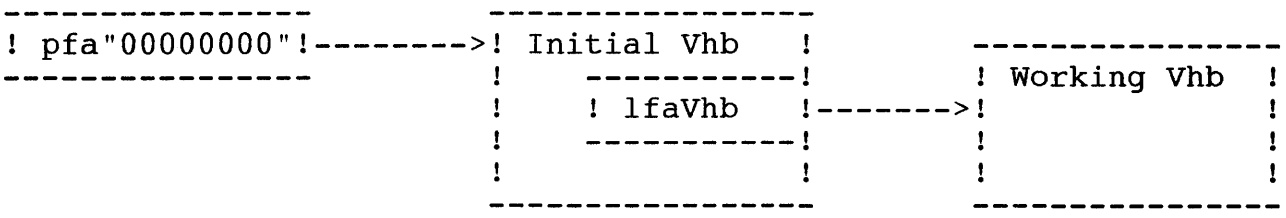


Figure 2. Access path to the working Vhb

The layout of the VHB is decribed in the following table.

Offset	Field	Size (in bytes)	Default Values
0	checksum	2	Variable
2	lfaSysImageBase	4	Variable
6	cPagesSysImage	2	User Defined
8	lfaBadBlkBase	4	Variable
12	cPagesBadBlk	2	Device Dependent
14	lfaCrashDumpBase	4	Variable
18	cPagesCrashDump	2	User Defined
20	sbVolName	13	User Defined
33	sbVolPassword	13	User Defined
46	lfaVhb	4	Variable
50	lfaInitialVhb	4	00000000
54	creationDT	4	Variable
58	modificationDT	4	Variable
62	lfaMfdBase	4	Variable
66	cPagesMfd	2	User Defined
68	lfaLogBase	4	Variable
72	cPagesLog	2	User Defined
74	currentLogPage	2	0
76	currentLogByte	2	0
78	lfaFileHeadersBase	4	Variable
82	cPagesFileHeader	2	User Defined

The information contained within this document is confidential and proprietary to Burroughs Corporation.

84	altFileHeadersPageOffset	2	User Defined
86	FreeFileHeaderNum	2	1
88	cFreeFileHeaders	2	User Defined
90	clusterFactor	2	1
92	defaultExtend	2	1
94	allocSkipCnt	2	0
96	lfaAllocBase	4	Variable
100	allocPageCnt	2	Device Dependant
102	lastAllocPg	2	Variable
104	lastAllocWd	2	Variable
106	lastAllocBit	2	Variable
108	cFreePages	4	Variable
112	idev	2	Variable
114	rgLruDirEntries (sRgLruDirEntries)	105	Variable
219	magicWd	2	7C39
221	BootBaseSector	1	Variable
222	BootBaseHead	1	Variable
223	BootBaseCyl	2	Variable
225	BootMaxPageCount	2	Variable
227	BadBlkBaseSector	1	Variable
228	BadBlkBaseHead	1	Variable
229	BadBlkBaseCyl	2	Variable
231	BadBlkMaxPageCount	2	Variable
233	DumpBaseSector	1	Variable
234	DumpBaseHead	1	Variable
235	DumpBaseCyl	2	Variable
237	DumpMaxPageCount	2	Variable
239	bytesPerSector	2	Device Dependant
241	sectorsPerTrack	2	Device Dependant
243	tracksPerCyl	2	Device Dependant
245	cylindersPerDisk	2	Device Dependant
247	interleaveFactor	1	Device Dependant
248	sectorSize	2	Device Dependant
250	spiralFactor	1	Device Dependant
251	startingSector	1	Device Dependant
252	Verify Code	1	0
253	Vendor Code	3	Device Dependant

TABLE 2. Volume Home Block Structure

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Values for the device dependent fields are given in the following table

Field	5 1/4 inch floppy
cPagesBadBlk	1
allocPageCnt	1
bytesPerSector	512
sectorsPerTrack	8
tracksPerCyl	2
cylindersPerDisk	77
interleaveFactor	1
sectorSize	606
spiralFactor	3
startingSector	1
Vendor Code	000000

TABLE 3. device dependent fields in the volume home block

The fields of the above structure are discussed in detail in the subsections which follow.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.1.1 checksum

The checksum field is used to modify the sum of the first 128 words of the Volume Home Block, such that it adds up to the "Magic Number". This field is used when validating the Vhb.

N.B The Magic Number has a value of 7C39 hex.

5.1.1.2 lfaSysImageBase

The lfaSysImageBase gives the pfa of the first sector of the system file Sysimage.sys. This field has no meaning to the S40, however it must be set up during volume initialisation, in order to maintain B25 BTOS compatibility.

5.1.1.3 cPagesSysImage

The cPagesSysImage field contains the size of the Sysimage.sys file in pages, or 512 byte sectors on the disk. This Field has no meaning to the S40, but is required, in order to maintain B25 BTOS compatibility.

N.B. A null file called Sysimage.sys must be created at volume initialisation time, as part of the requirement for B25 BTOS compatiblity.

5.1.1.4 lfaBadBlkBase

The lfaBadBlkBase field gives the pfa of the first sector of the bad block file on disk. The S40 uses this field when a volume is mounted, in order to map the bad blocks on the disk.

5.1.1.5 cPagesBadBlk

The cPagesBadBlk field gives the size of the bad blocks file in pages. This field tells the S40 how many bad blocks may exist on the disk.

For a floppy disk this field should be set to 1, however for Winchesters, this value may be larger as the number of bad sectors allowed on the disk may be larger.

5.1.1.6 lfaCrashDumpBase

The lfaCrashDumpBase field gives the pfa of the first sector of the CrashDump file. This field is not relevant to the S40. This entry must only be maintained for reasons of B25 BTOS compatability.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.1.7 cPagesCrashDump

The cPagesCrashDump field specifies the size of the crash dump file in pages.This entry must only be maintained for reasons of B25 BTOS compatability.

5.1.1.8 sbVolName

The sbVolName field is a string structure which contains the name of the volume. The field is thirteen bytes long and has the format described in the following table:

Offset in Bytes	Field	Size	Description
0	Size_Byte	1	Number of significant bytes in the name field.
1	Name_Field	12	Character array which contains the volume name.

TABLE 4. sbVolName structure

5.1.1.9 sbVolPassword

The sbVolPassword field is a thirteen character field, which contains the volume password. It has the same format as the sbVolName field. In order to maintain the disk security on the S40, this field should be zeroed out if a user request to read the Vhb is made. The field is filled with nulls as any other character would be valid as part of a password.

5.1.1.10 lfaVhb

The lfaVhb field contains the pfa of the working Vhb. In the working Vhb this field is a self pointer to the working Vhb.

5.1.1.11 lfaInitialVhb

The lfaInitialVhb field contains the pfa of the initial Vhb set up when the disk was initialised. On initialisation two copies of the Vhb are created. The working Vhb, and initial Vhb. The initial Vhb is not modified after the volume has been initialised. The working Vhb is continuously updated to reflect the current state of the volume. In the initial Vhb this field is a self pointer.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.1.12 creationDT

The creationDT field contains the date on which the volume was last initialised. The format of this field is described below.

The first word of the field contains the number of seconds which have elapsed since the last noon or midnight occurrence.

The second word of the field contains the number of 12 hour periods which have elapsed since midnight March 1 1952.

If the time value is zero this indicates a null date/time.

N.B. Midnight and a.m. periods are represented by the second word MOD 2 being equal to 0. Noon and p.m. periods are represented by the second word MOD 2 being equal to 1

For simpler conversion of date/time to an output format it is worth noting that March 1 1952 follows a leap day.

5.1.1.13 modificationDT

The modificationDT field contains the date on which the volume was last modified (i.e. the disk was physically written to). The format of this field is the same as for the creationDT field.

5.1.1.14 lfaMfdBase

The lfaMfdBase field contains the pfa of the Master File Directory file (Mfd.sys) which is the entry point to the directory system.

5.1.1.15 cPagesMfd

The cPagesMfd field gives the size of the Mfd.sys file in pages. The size of this file is dependent on certain of the initialisation parameters which are supplied by the user regarding the maximum number of directories which can be created on the disk.

5.1.1.16 lfaLogBase

The lfaLogBase field contains the pfa of the first sector of the file 'Log.sys' This file is used by the B25 to log error conditions, and certain key events. This must only be supplied to maintain B25 BTOS compatibility.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.1.17 cPagesLog

The cPagesLog field contains the size of the file 'Log.sys' in pages.

5.1.1.18 currentLogPage

The currentLogPage field gives the page number on which the next usable log location is held, offset from the page pointed to by lfaLogBase.

5.1.1.19 currentLogByte

The currentLogByte field contains the byte offset in to the page, indicated by the currentLogPage field, at which the next log entry can start.

5.1.1.20 lfaFileHeadersBase

The lfaFileHeadersBase field gives the pfa of the first sector of the file 'FileHeaders.sys' which contains the file header blocks.

5.1.1.21 cPagesFileHeader

The cPagesFileHeader field contains the size of the file headers file in pages.

5.1.1.22 altFileHeadersPageOffset

For safety and security the BTOS disk subsystem maintains two file headers for every file on the disk. This means that if a Fhb becomes unreadable due to a bad block, then the file can still be accessed. The offset in pages between the two Fhb's is contained in the altFileHeaderPageOffset field. If this field is equal to 0 then there are no alternate file headers.

5.1.1.23 FreeFileHeaderNum

The FreeFileHeaderNum field contains an index to the next available file header block in the chain of free file headers.

5.1.1.24 cFreeFileHeaders

The cFreeFileHeaders field contains the number of free file headers in the file 'FileHeaders.sys'. This refers to the number of file headers which are available for allocation. Alternate file headers are not included. If this field is equal to 0 then there are no free file headers.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.1.25 clusterFactor

This field must always be set to 1, it is not used on the S40.

5.1.1.26 defaultExtend

This field must always be set to 1, it is not used on the S40.

5.1.1.27 allocSkipCnt

This field must always be set to 0, it is not used on the S40.

5.1.1.28 lfaAllocBase

The lfaAllocBase field contains the address of the first sector of the Allocation Bit Map.

5.1.1.29 allocPageCnt

The allocPageCnt field contains the size of the Allocation Bit Map in pages.

5.1.1.30 lastAllocPg

The lastAllocPg field contains the page number of the Allocation Bit Map page on which the last allocated sector of the disk is represented.

5.1.1.31 lastAllocWd

The lastAllocWd field contains the word offset in the page represented by the lastAllocPg field in which the last allocated sector on the disk is represented

5.1.1.32 lastAllocBit

The lastAllocBit field contains the bit index into the word given by the lastAllocWord field of the bit which represents the last allocated sector on the disk.

N.B. The lastAllocPg, lastAllocWd and lastAllocBit fields are used when allocating disk space to even out surface usage on the disk.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.1.33 cFreePages

The cFreePages field contains the number of unallocated sectors on the disk.

5.1.1.34 idev

This is a device number, as configured in the BTOS system. This field is not relevant to the S40.

5.1.1.35 rgLruDirEntries (sRgLruDirEntries)

The rgLruDirEntries field is used to speed up directory access. The name is an abbreviation for Least Recently Used Directory Entries. Every time a directory is accessed the system scans this field for the directory name. This field can be accessed as an array of 3 directory entry blocks, which have the password fields filled with nulls.

5.1.1.36 magicWd

The magicWd field contains the value to which the Vhb should sum up to when being validated. The contents of this field are used as part of the validation of the Vhb. This field should contain 7C39 hex. If it does not, then the Vhb is not valid.

5.1.1.37 BootBaseSector

This field is not relevant to the S40.

5.1.1.38 BootBaseHead

This field is not relevant to the S40.

5.1.1.39 BootBaseCyl

This field is not relevant to the S40.

5.1.1.40 BootMaxPageCount

This field is not relevant to the S40.

5.1.1.41 BadBlkBaseSector

This field is not relevant to the S40.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.1.42 BadBlkBaseHead

This field is not relevant to the S40.

5.1.1.43 BadBlkBaseCyl

This field is not relevant to the S40.

5.1.1.44 BadBlkMaxPageCount

This field is not relevant to the S40.

5.1.1.45 DumpBaseSector

This field is not relevant to the S40.

5.1.1.46 DumpBaseHead

This field is not relevant to the S40.

5.1.1.47 DumpBaseCyl

This field is not relevant to the S40.

5.1.1.48 DumpMaxPageCount

This field is not relevant to the S40.

5.1.1.49 bytesPerSector

This field specifies the number of data bytes in a disk sector. For a floppy disk this field should be set to 512.

5.1.1.50 sectorsPerTrack

This field specifies the number of sectors on a track. For a floppy disk this should be set to 8.

5.1.1.51 tracksPerCyl

This field specifies the number of tracks on a cylinder. For a floppy disk this field should be set to 2.

5.1.1.52 cylindersPerDisk

This field specifies the number of cylinders on the disk. For a floppy disk this field should be set to 77.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.1.53 interleaveFactor

The interleave factor specifies the physical number of sectors by which logically contiguous sectors are separated. For a 5.25 inch floppy disk this field is set to 1.

5.1.1.54 sectorSize

This field indicates how many bytes of physical data, including the format data, make up a logical sector of the disk. For a 5.25 inch floppy disk this field is set to 606. This is related to the IBM system 34 disk format.

5.1.1.55 spiralFactor

The spiral factor of a disk specifies the number of sectors by which the first sector on side 1 is skewed for the last sector on side 0. For a 5.25 inch floppy disk this field is set to 3.

5.1.1.56 startingSector

This field specifies the lowest number of physical sector, which is recognised by the floppy disk controller. This field is related to the format of the disk, and for a 5.25 inch floppy disk this is set to 1.

5.1.1.57 VerifyCode

For a 5.25 inch floppy disk this field is set to zero, it is not used on the S40.

5.1.1.58 VendorCode

This field is used in conjunction with Winchesters to allow the operating system to identify the make of disk. For a 5.25 inch floppy disk this field is set to zero, it is not used on the S40.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.2 SECTOR ALLOCATION BIT MAP

The Allocation Bit Map (Abm) is used by the system to determine which sectors on the disk are free to be used. The map consists of an area of disk, which has a bit flag for every sector on the disk. The Allocation Bit Map is accessed by the lfaAllocBase field of the working Vhb. The size of the map is given by 'allocPageCnt'. The Vhb also contains information about the position of the last allocated sector on the disk. The pages of the Abm are initialised to 1's during the initialisation of the volume, with the exception of those bits in the last page of the Abm which do not refer to valid sectors, these bits are set to 0's to prevent allocation occuring in that zone.

The Abm is used by the system to allocate free sectors of disk to a file when requested.

The Allocation Bit Map is an array of bits which represent the disk as a linear sequence of sectors. The bits are indexed by page, word and bit number:

sector 0 is represented by page 0, word 0, bit 0.
sector 18 is represented by page 0, word 1, bit 2.
sector 284 is represented by page 0, word 16, bit 12.

If the value of a bit is 1 then this indicates that the corresponding sector is unallocated. If the value of a bit is 0 then the corresponding sector is allocated. When an allocation request is received the allocation routine will start from the bit indicated by the lastAllocPage, lastAllocWord and lastAllocBit fields of the Vhb, and search for a free sector. It then scans the follwing sectors to see if there are enough contiguous free sectors to fulfil the request. If the requested number of sectors are found then the scan stops, and the address, and size of the disk extent are returned. Also the lastAllocPage, lastAllocWord, and lastAllocBit fields are updated. If the search returns to the current starting point without finding a fit then the biggest available disk extent is returned to the caller.

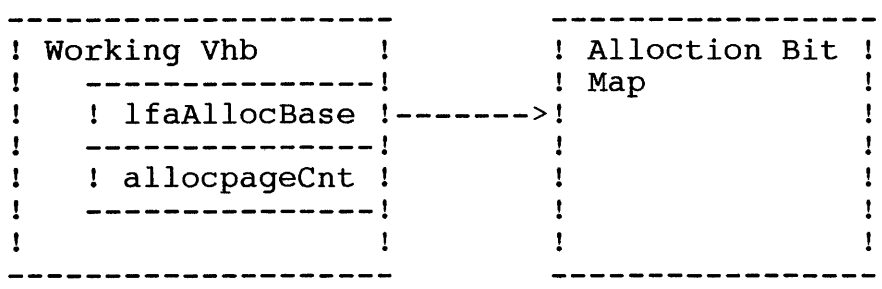


Figure 3. Access path to the Allocation Bit Map

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.3 BAD BLOCK FILE

The bad block file contains information regarding the location of all known bad sectors on the disk. The file consists of three arrays. The first array contains the bad sectors' 'sector number' within a track. The second array indicates the surface number. The third array indicates the cylinder number.

Bad blocks are recorded at initialisation time by marking them as allocated in the allocation bit map. The bad block file is not altered dynamically, if a bad sector is discovered during run-time then an I/O error is returned. The user may continue using the disk but to get the bad sector logged in the bad block file, the disk must be reinitialised.

For a 616 Kbyte floppy disk, the first array is a byte array, 128 elements long, and runs from byte 00H to 7FH within the bad block file. The second array is a byte array of 128 elements and runs from 80H to FFH. The third array is a word array of 128 elements, and runs from 100H to 1FFH.

So for a 616 Kbyte floppy disk, the address of the first bad sector can be found by reading the following elements:

Byte	00H	The sector number (1 to 8)
Byte	80H	The surface number (0 or 1)
Word	100H	The cylinder number (0 to 76)

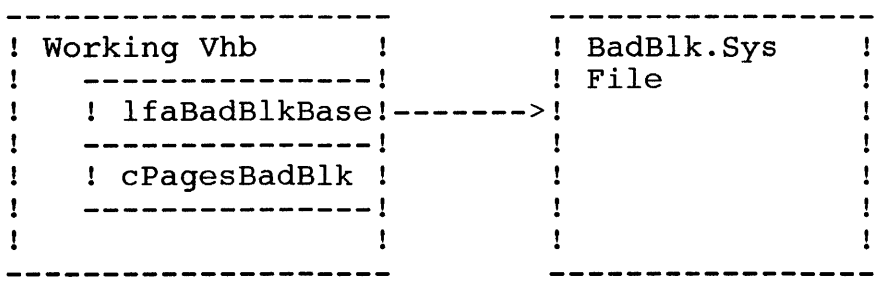


Figure 4. Access path to the bad block File

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.4 THE MASTER FILE DIRECTORY

The Master File Directory (Mfd) is a file called Mfd.sys, which is always located in the system directory of a disk. The Mfd is also pointed to by the lfaMfdBase entry in the working Vhb. The Mfd contains information regarding all of the directories on the volume, including the system directory. Each directory has a Directory Entry Block (Deb) in the Mfd. Each Deb is 35 bytes long, and includes a pointer to the first page of the directory.

The size of the Mfd is determined at disk initialisation time, by the 'maximum number of directories' parameter.

An Mfd page can be considered as an array of 14 directory entry blocks, starting at byte 1. Byte 0 is a dummy byte which should be zero. The minimum size of Mfd.sys is 1 page. The default size of Mfd.sys is 1 page.

When a directory is deleted all Deb's below the deleted Deb on an mfd page are moved up to close the gap, and all Deb's beyond the last valid Deb on an mfd page are filled with the hexadecimal value 00.

The size of the Mfd.sys file in pages is calculated using the 'number of directories' parameter at initialisation time. The calculation is as follows:

```
Number_of_mfd_pages := (no_of_directories + 13) / 14
```

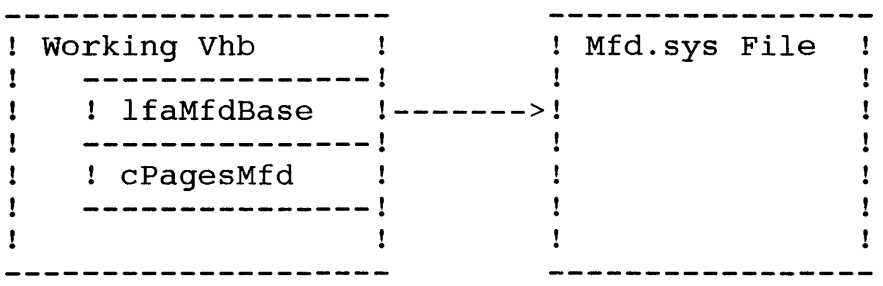


Figure 5. Access path to the Master File Directory.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.5 THE DIRECTORY ENTRY BLOCK

A Directory Entry Block (Deb) is a data structure used within the Master File Directory (Mfd). A Deb contains the information neccessary to find a directory page on the disk. In order to locate a Deb within the Mfd, the directory name is used to generate a modifier for the lfaMfdBase field to find the correct Mfd page. All the Debs in the Mfd page are then scanned until the entry for the named directory is found, or all the entries have been scanned.

The layout of a Deb is given in the following table :

Offset	Field	Size
0	dirEntryName	13
13	password	13
26	lfabase	4
30	cPages	2
32	defaultAccessCode	1
33	lruCnt	2

TABLE 5. Directory Entry Block Structure

The fields of the above structure are discussed in detail in the subsections which follow.

5.1.5.1 dirEntryName

The dirEntryName field contains the name of the directory in the following format:

Offset in Bytes	Field	Size	Description
0	Size_Byte	1	Number of significant bytes in the name field.
1	Name_Field	12	Character array which contains the directory name.

TABLE 6. Format of the dirEntryName field in the Deb

5.1.5.2 password

The password field has the same format as the dirEntryName field, and contains the directory password.

5.1.5.3 lfabase

The lfabase field contains the pfa of the first page of the directory.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.5.4 cPages

The cPages field contains the size of the directory in pages.

5.1.5.5 defaultAccessCode

This field contains the protection level which has been set on the directory. This access code is given by default to all files created in the directory.

5.1.5.6 lruCnt

This field is used to identify the least recently used Deb when it is resident in memory in the rglruDirEntries field of the Vhb.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.6 THE DIRECTORY

A directory contains one File Entry Block (Feb) for every file in that directory. The position of the Feb within the directory is determined by hashing techniques. The Feb contains the files' name, and a pointer to the File Header Block. Every file on the disk must have a Feb entry in some directory on the disk. The directory is accessed via the lfaBase field and the cPages field in the Deb as illustrated below.

Directory pages are allocated from the Allocation Bit Map at the time that the directory is created. The Minimum size of a directory is 1 page. The default size of a directory is 3 pages.

The first byte of a directory page is always a null character.

A directory page is made up of a number of contiguous file entry blocks. When a file entry is deleted the directory page is reorganised to ensure that the file entry blocks remain contiguous and any free bytes at the end of the directory page are set to nulls.

A file entry is placed in a directory page based on a hashing algorithm, which selects a directory page hashed on the file name. If the file entry cannot be accomodated in the hashed page, the system goes through the directory pages in a round-robin fashion in an attempt to accomodate the file entry. If this fails then a DK_DIR_FULL error is reported.

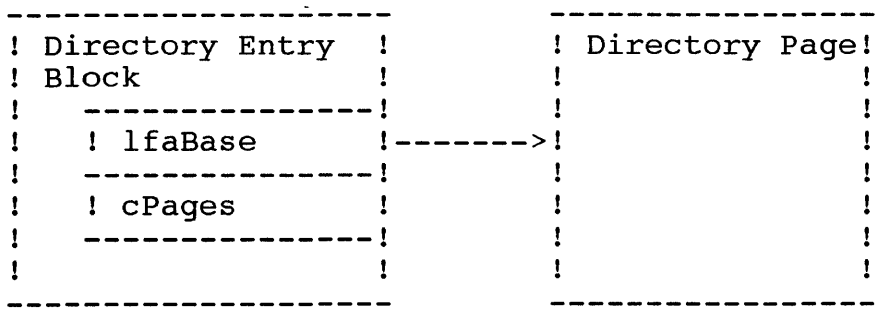


Figure 6. Access path to a directory

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.7 FILE ENTRY BLOCK

A File Entry Block (Feb) structure is an entry which is found in a directory page, and contains the information necessary to locate a File Header Block for a file. The structure of a File Entry Block is given in the following table:

Offset	Field	Size
0	Filename	Variable from 2 to 51 bytes.
Variable	FileHeaderNum	2

TABLE 7. File Entry Block Structure

The fields of the above structure are discussed in detail in the subsections which follow.

5.1.7.1 Filename

The Filename field contains the name of the file in the following form.

Offset in Bytes	Field	Size	Description
0	Size_Byte	1	Number of bytes in the name field.
1	Name_Field	Variable	Character array which contains the file name. This field is of variable size, dependent on the file name, from 1 to 50 bytes.

TABLE 8. Format of the Filename field of the Feb

5.1.7.2 FileHeaderNum

The FileHeaderNum field contains the file header number of the primary file header block for this file.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.8 THE FILE HEADERS FILE

The File Headers File is a system file, which contains all of the File Header Blocks (Fhb) used in the file system. The size of the file headers file is decided at initialisation time, by the 'maximum number of files' parameter, and the 'primary file headers only' parameter. To calculate the size of the file headers file the following calculations are performed (where maxfiles represents the maximum number of files):

```
If maxfiles = 0 then maxfiles := freeSectors/20

If maxfiles > 1500 then maxfiles := 1200 + maxfiles/5

If maxfiles > 21500 then error(DiskNotLargeEnough)

If noAlternateHeaders then
        If maxfiles < 10 then maxfiles := 10
        AlternateHeaderOffset := 0
        fhbPages := maxfiles * 3 / 2

If AlternateHeaders then
        If maxfiles = 0 then maxfiles := 1
        AlternateHeaderOffset := sectorsPerTrack * bytesPerSector
                                 / pageSize * 3 / 2
        fhbPages := (((maxfiles*3)-1)/(AlternateHeaderOffset * 2))+1
                    * (AlternateHeaderOffset * 2)
```

Key

maxfiles	=	Maximum number of files that the user would like to put on the disk. If this value is zero, then the value is taken to be the number of unallocated sectors on the disk divided by 20. (The assumption being made here is that the average size of a file is 20 sectors).
noAlternateHeaders	=	A boolean variable indicating if the user wishes the disk to contian duplicates of the file header blocks. True = Primary headers only.
AlternateHeaderOffset	=	A field in the Vhb.
fhbPages	=	A field in the Vhb.
SectorsperTrack	=	Number of physical sectors per track on the disk.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Bytespersector = Number of bytes per sector.

PageSize = Size of disk page = 512 bytes.

If AlternateHeaders are required then the secondary file headers must be located at different rotational offsets from the primary file headers for maximum effect.

The free primary file headers are chained together on initialisation of the FileHeaders.sys file. The chain is initialised by stepping through the primary file headers, and setting the fileHeaderPageNum field equal to the file header page number, the extensionHeaderNumChain field equal to the next primary file header page number, and the fileHeaderNum to zero. All other fields are set to zero, and the checksum adjusted accordingly. Each primary file header block is also copied to the alternate file header block (if these are selected at initialisation).

When the system is required to allocate a file header block for a file, the following sequence of actions is performed:

- Check the volume home block numFreeFileHeaders field for zero. If this field is zero then a DK_NO_FREE_FH error is reported.
- If there are free file headers then the file header indicated by the FreeFileHeaderNum field of the volume home block is read from the disk.
- If the fileHeaderNum field is not zero then the free headers chain is broken, and a DK_FH_CHAIN_BROKEN error is reported.
- If the chain is unbroken then the freeFileHeaderNum field of the volume home block is set equal to the extensionHeaderChainNum field of the file header block and the numFreeFileHeaders field of the volume home block is decremented by 1.
- Finally the file header block is zeroed out with nulls.

When a file header is to be returned to the system the following sequence of actions must be performed.

- The length of the file name is set to zero.
- The fileHeaderNum field is set to zero.
- The extensionHeaderNumChain field is set to the freeFileHeaderNum field of the volume home block.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

- The fileHeadersequence number is set to zero.

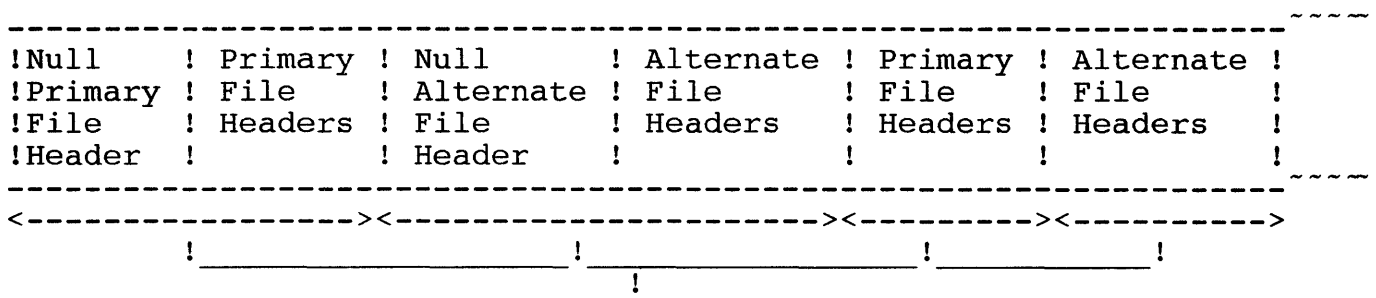
- File header is rewritten to disk.

- The freeFileHeaderNum field of the volume home block is set equal to the primary file header page number.

- The freeFileHeader count of the volume home block is incremented by 1.

N.B. This must be done for any extension file headers also. The file header in page 0, and if alternate file headers are used, the file header at the AlternateHeaderOffset page are initialised but never used. This is probably due to the use of zero in the fileHeaderNum field of the file header block to indicate a valid 'Free Fhb Chain'. If the 'Primary file headers only' option was selected then the file headers file contains no alternate file headers.

So during normal operation the layout of the FileHeaders.sys file is as follows:



Each section is a fixed number of pages long. This size is specified by the alternateHeaderPageOffset field in the Vhb.

If the 'primary file headers' only option is selected (i.e. the alternateHeaderPageOffset is zero) then the file layout will be as follows :

The information contained within this document is confidential and proprietary to Burroughs Corporation.

If the number of disk extents required for a file exceeds 32 then it is necessary to allocate an 'Extension File header'. When this is done, the system allocates a free file header block, which is chained to the old Fhb via the extensionHeaderNumChain field. In order to set up an extension header the following actions must be performed:

- Set the extensionHeaderNumChain field of the old file header block to the fileHeaderPageNum field of the new extension header.
- Set the headerSequenceNum field of the extension header equal to the headerSequenceNum field of the old file header, plus 1.
- All other fields in the extension header should be set up as detailed in the following section on the file header block.

5.1.9 THE FILE HEADER BLOCK

Every file has a File Header Block (Fhb) associated with it. All Fhb's are contained in a file in the system directory called Fileheaders.sys. There are two copies of every Fhb, the primary copy and the secondary copy, which are located at different rotational positions, and on different cylinders. This is done for reliability, and efficiency. Each Fhb is 1 sector long (i.e. 512 bytes). The Fhb is accessed using the lfaFhb field of the Feb as illustrated below:

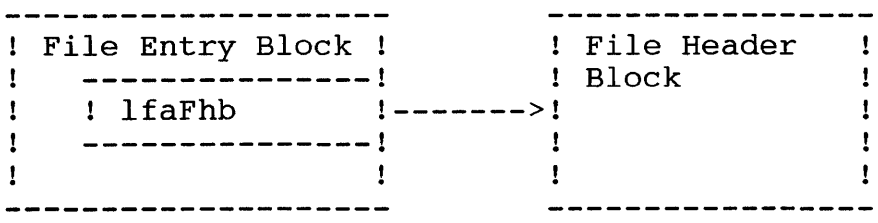


Figure 7. Access path to the File Header block

The information contained within this document is confidential and proprietary to Burroughs Corporation.

The structure of the Fhb is defined in the table below:

Offset	Field	Size
0	checksum	2
2	fileHeaderPageNum	2
4	fileName	51
55	password	13
68	dirName	13
81	fileHeaderNum	2
83	extensionHeaderNumChain	2
85	headerSequenceNum	1
86	fileClass	1
87	accessProtection	1
88	lfaDirPage	4
92	creationDT	4
96	modificationDT	4
100	accessDT	4
104	expirationDT	4
108	fNoSave	1
109	fNoDirPrint	1
110	fNoDelete	1
111	lfaEndOfFile	4
115	defaultExpansion	4
119	freeRunIndex	2
121	vda(runsPerFhb)	128
249	runLength(runsPerFhb)	128
377	(reserved for expansion)	71
448	application-specific field	64

TABLE 9. File Header Block Structure

The fields of the above structure are discussed in detail in the subsections which follow.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.9.1 checksum

The checksum field is used to ensure that when the Fhb is validated, the bytes add up to the 'magic number'.

5.1.9.2 fileHeaderPageNum

This is the page number of the file header within the fileHeaders.sys file. This field is always set to the page number, even in an extension Fhb.

5.1.9.3 fileName

The fileName field is a structure which contains the length of the file name in the first byte, followed by up to 50 characters of the name.

Offset in Bytes	Field	Size	Description
0	Size_Byte	1	Number of significant bytes in the name field.
1	Name_Field	50	Character array which contains the file name.

TABLE 10. Format of the fileName field of the Fhb

N.B. This field is not used in an extension Fhb.

5.1.9.4 password

The password field is a structure which contains the length of the password in the first byte followed by up to 12 characters of a password.

Offset in Bytes	Field	Size	Description
0	Size_Byte	1	Number of significant bytes in the name field.
1	Name_Field	12	Character array which contains the password.

TABLE 11. Format of the password field in the Fhb

N.B. This field is not used in an extension Fhb.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.9.5 dirName

The dirName field is a structure which contains the name of the directory which the file is associated with. The format of this field is given in the following table.

Offset in Bytes	Field	Size	Description
0	Size_Byte	1	Number of significant bytes in the name field.
1	Name_Field	12	Character array which contains the directory name.

TABLE 12. Format of the dirName entry field in the Fhb

N.B. This field is not used in an extension Fhb.

5.1.9.6 fileHeaderNum

This field always contains the page number within fileHeaders.sys of the original file header block in a chain of file header blocks. In the original file header block, this field is the same as the fileHeaderPageNum field.

5.1.9.7 extensionHeaderNumChain

If there are more than 32 disk extents associated with a file, this field gives the page number of the next extension file header block associated with the file. If there are no more extension header blocks, then this field is set to zero.

5.1.9.8 headerSequenceNum

This field indicates where in a sequence of extension file headers this file header is placed. For the initial file header, this field is set to 0. For each extension Fhb in a chain this field is set to 1 greater than the corresponding field in the previous Fhb in the chain.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.9.9 fileClass

This field indicates the type of file, it is zero in all cases on the S40.

N.B. This field is not used in an extension Fhb.

5.1.9.10 accessProtection

The accessProtection field contains the value of the files' protection level. For more details see the section on Disk Security.

N.B. This field is not used in an extension Fhb.

5.1.9.11 lfaDirPage

The lfaDirPage field gives the pfa of the directory page in which the File Entry Block for this file can be found.

N.B. This field is not used in an extension Fhb.

5.1.9.12 creationDT

The creationDT field indicates when the file was first created. The field format is the same as the creationDT field in the Vhb.

N.B. This field is not used in an extension Fhb.

5.1.9.13 modificationDT

The modificationDT field indicates when the file was last changed, i.e when the disk was last written to due to an operation on this file. The format of this field is the same as the creationDT field.

N.B. This field is not used in an extension Fhb.

5.1.9.14 accessDT

The accessDT field indicates when the file was last accessed, i.e. when the file was last read from, or written to. The format of this field is the same as the creationDT field.

N.B. This field is not used in an extension Fhb.

5.1.9.15 expirationDT

This field is not used, and should be set to zero.

N.B. This field is not used in an extension Fhb.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.9.16 fNoSave

If this field contains 0FFh then the file cannot be saved (overwritten) by the user.

N.B. This field is not used in an extension Fhb.

5.1.9.17 fNoDirPrint

If this field contains 0FFh then the file cannot be directly printed by the user.

N.B. This field is not used in an extension Fhb.

5.1.9.18 fNoDelete

If this field contains 0FFh then the file cannot be deleted by the user.

N.B. This field is not used in an extension Fhb.

5.1.9.19 lfaEndOfFile

This field contains the length of the file in bytes

N.B. This field is not used in an extension Fhb.

5.1.9.20 defaultExpansion

This field is set to zero in all cases, on the S40.

N.B. This field is not used in an extension Fhb.

5.1.9.21 freeRunIndex

This field contains the index to the vda and runlength fields for the next slot which can take a new disk extent. i.e if there are two disk extents associated with the file then this field will be 2 (0,1,2....). If the value of this field is 32 then an extension file header is required.

5.1.9.22 vda

The vda field is a 32 by 4 byte array of pfa's which point to the base sectors of disk extents which are associated with the file (See section on disk extents).

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.9.23 runLength

The runLength field is a 32 by 4 byte array which gives the sizes of the disk extents pointed at by the corresponding vda entries.

5.1.9.24 application specific field

This field is usable by an application, but is not relevant to the operation of the disk subsystem.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.1.10 DISK EXTENTS

A disk extent is a number of contiguous sectors of the disk, which normally contain user data. One exception to this is the file headers file (Fileheaders.sys) which is a disk control structure which exists on a single disk extent.

The Fhb contains pointers to the disk extents which contain a files' data. A file is constructed of a number of disk extents which are accessed by elements of the vda, and RunLength fields of the Fhb as shown below:

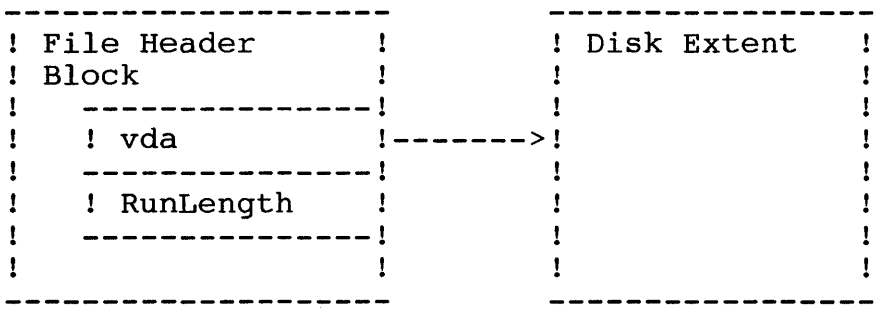


Figure 8. Access path to a Disk Extent of a file

5.1.11 HASHING TECHNIQUE

Deb's and Feb's are positioned in Mfd.sys and a directory by Hashing a page offset in to the structure based on the file or directory name. The hashing algorithm is not case sensitive, and all lower case characters are converted to upper case. The hashing algorithm is as follows:

```
temp  := 0
index := 0
While index < No-of-characters-in name do
Begin
   temp  := 73 * temp + Name[index]
   index := index + 1
End
Page_Number := temp mod  Number-of-pages-in-structure
```

Where name is a character array containing the name with all lower case characters converted to upper case.

The information contained within this document is confidential
and proprietary to Burroughs Corporation.

5.2 INTERNAL DATA STRUCTURES

This section decribes the internal data structures used in the Disk Access Routines, for the purposes of controlling access paths to the various devices and files. Temporary storage variables which change from function call to function call are not described.

The Disk Access Routines module has been designed to control 'N' devices where N is specified at compile time. The structure descriptions which follow are for a system in which 'N' is equal to 1, in line with the immediate subsystem requirements.

The Disk Access Routines module has also been designed to control 'F' files where F is specified at compile time. The structure descriptions which follow are for a system in which 'F' is equal to 16.

There are 8 internal control structures within the Disk Access Routines module, which are used to control access paths to devices, and files, and to control disk space resources. Some of the structures are also utilised during recovery from certain disk errors. This usage will be explained in the section regarding Error Recovery. The structures are:

- Device control block table
- Volume name table
- Current Volume Home Block table
- Last written Volume Home Block table
- Current Allocation Bit Map table
- Last written Allocation Bit Map table
- File Control Block table
- Default path table

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.2.1 DEVICE CONTROL BLOCK TABLE

The Device control block (Dcb) table is used by the disk access routines to hold information about a specific device. There is one Device control block entry for every device in the link. The table is indexed using the device number, i.e. the position of the device in the system. The first device is device number zero. A Dcb entry is 4 bytes long, and consists of the following fields:

Offset in Bytes	Field	Size	Description
0	state	2	The current state of the device.
2	dev_address	1	The address by which the device can be accessed by the device driver.
3	type	1	The type code of the device.

TABLE 13. Device Control Block Structure

The fields of the above structure are discussed in more detail in the sections which follow.

5.2.1.1 state

The device state is a state variable which contains the current state of the device within the device state machine. The devices are controlled using a local state machine which applies only on a logical device level.

There are 6 states in the state machine, these are:

INACTIVE state 0

ACTIVE state 1

VHB_LOADED state 2

ERROR state 3

RESET state 4

RECOVER state 5

The information contained within this document is confidential and proprietary to Burroughs Corporation.

There are 8 transition triggers:

- Power on
- Reset
- Wakeup/Mode successful
- Not Ready
- Timeout
- Vhb/Abm/Bbm read successfully
- Disk recovered successfully
- Disk recovery failed

The information contained within this document is confidential
and proprietary to Burroughs Corporation.

The state machine can be represented in the following manner:

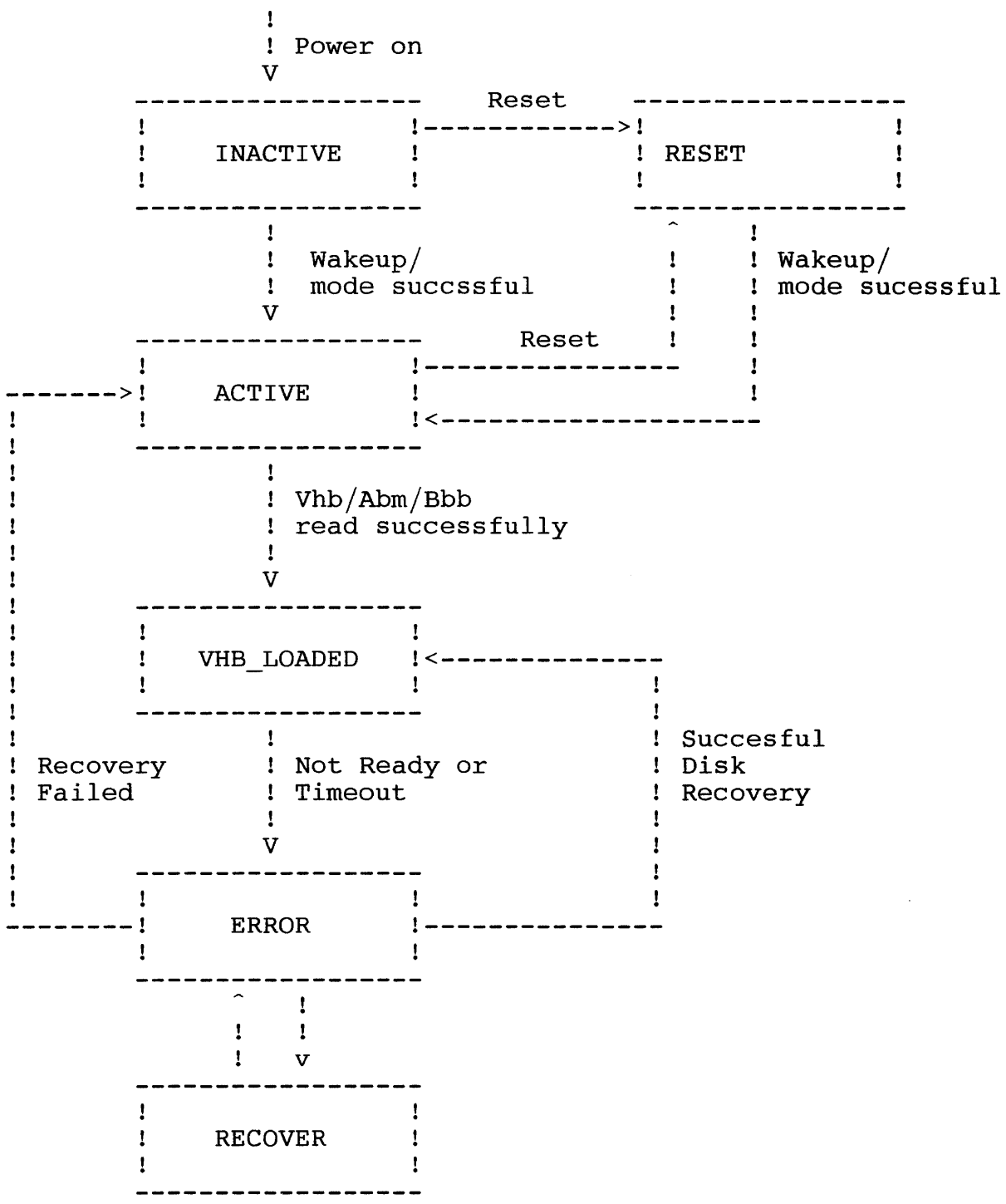


Figure 9. Device state machine diagram

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.2.1.2 dev_address

The dev_address field contains the data communications address of the disk device. This address is calculated after the device has been woken up, by adding 10 hex to the device number. i.e the first device will have the address 10 hex, the second device will have the address 11 hex etc.

5.2.1.3 type

The type field contains the device type code which was returned to the S40 after a wakeup command was issued to the device. This code identifies the hardware configuration to the S40, and is used to control any parameter/status conversion which may be necessary between the S40 and the device. The type codes which are currently valid are given in the table below:

Type Code	Device Type
0	5.25 inch floppy disk drive double sided, double density soft sectored, 80 tracks, 2 surfaces, B25 compatible format (includes extended tracks not normally used).

Device can operate with 128, 256, 512 or 1024 byte sectors [3].

Note S40 uses mode 3 as defined in reference 3 only.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.2.2 VOLUME NAME TABLE

The volume name table is used to map volume names to device numbers. The volume name table consists of a number of entries of the same structure as the sbVolName field in the volume home block. The table is used as a lookup table to match a volume name provided by a request, to a volume which is known to the subsystem.

The first entry in the table is always made for the system volume, and contains the name 'SYS'. After this the table entries are paired. i.e. there are two entries for every volume, the first contains the volume name taken from the volume home block, the second contains the device name which is a way of identifying a volume by the device type and position in the communications line. e.g the first floppy disk in the system could be accessed as 'F0', a second floppy disk would be 'F1', etc.

The table is searched entry by entry, firstly by comparing the name_size field with the name_size of the target, and if these match, then by performing a byte for byte comparison of the names, for the name_size, until a mismatch is found, or until a comparison is found. The S40 is not case sensitive when making name comparisons.

On finding the target name, the device number is obtained by using the index of the matching entry. If the index is zero then the volume name is sys, and the device number is that of the designated system device. Otherwise the device number is calculated by subtracting 1 from the index and dividing by 2.

5.2.3 CURRENT VOLUME HOME BLOCK TABLE

The current Vhb table contains one entry for each device. The table is indexed by device number. The entry in the table is the working copy of the volume home block. All modifications to the volume home block are made to this structure, before being written to disk. After the structure has been succesfully written on to the disk, the entry is copied in to the corresponding entry in the last written Vhb table.

5.2.4 LAST WRITTEN VOLUME HOME BLOCK TABLE

The last written Vhb table contains one entry for each device. The table is indexed by device number. The entry in the table is a copy of the last volume home block which was written to a disk. The last written Vhb table entry is updated from the current Vhb table, after a Vhb has been successfully written to the disk. This structure is used by the S40 disk subsystem during recovery from certain faults. For details of this usage see the section on error recovery.

The information contained within this document is confidential
and proprietary to Burroughs Corporation.

5.2.5 CURRENT ALLOCATION BIT MAP TABLE

The Current Abm table contains the most up to date allocation bit map, for a volume. There is one entry in this table for every device. The entry is used when an I/O request to the disk subsystem requires the allocation or deallocation of space. If the current Abm is altered then it is rewritten to disk, and a copy of it is placed in the last written Abm table.

5.2.6 LAST WRITTEN ALLOCATION BIT MAP TABLE

The last written Abm table is used to maintain a copy of the allocation bit map which was last written to the disk. The last written Abm entry is updated from the corresponding current Abm table entry immediately after the current abm has been successfully written to the disk.

5.2.7 FILE CONTROL BLOCK TABLE

The file control block table is a table which contains one entry for every file which the disk subsystem will allow to be open at any one time. The structure of the table is as follows:

Offset in Bytes	Size	Description
0	2	State
2	2	Mode
4	2	dev_num
6	2	fhb_num
8	2	last_vda_index
10	2	last_io_fhb
12	4	last_lfa
16	4	fhb

TABLE 14. File Control Block Structure

These fields are described in detail in the sections which follow.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.2.7.1 State

File access is controlled by a simple state machine, which has 4 states. These are:

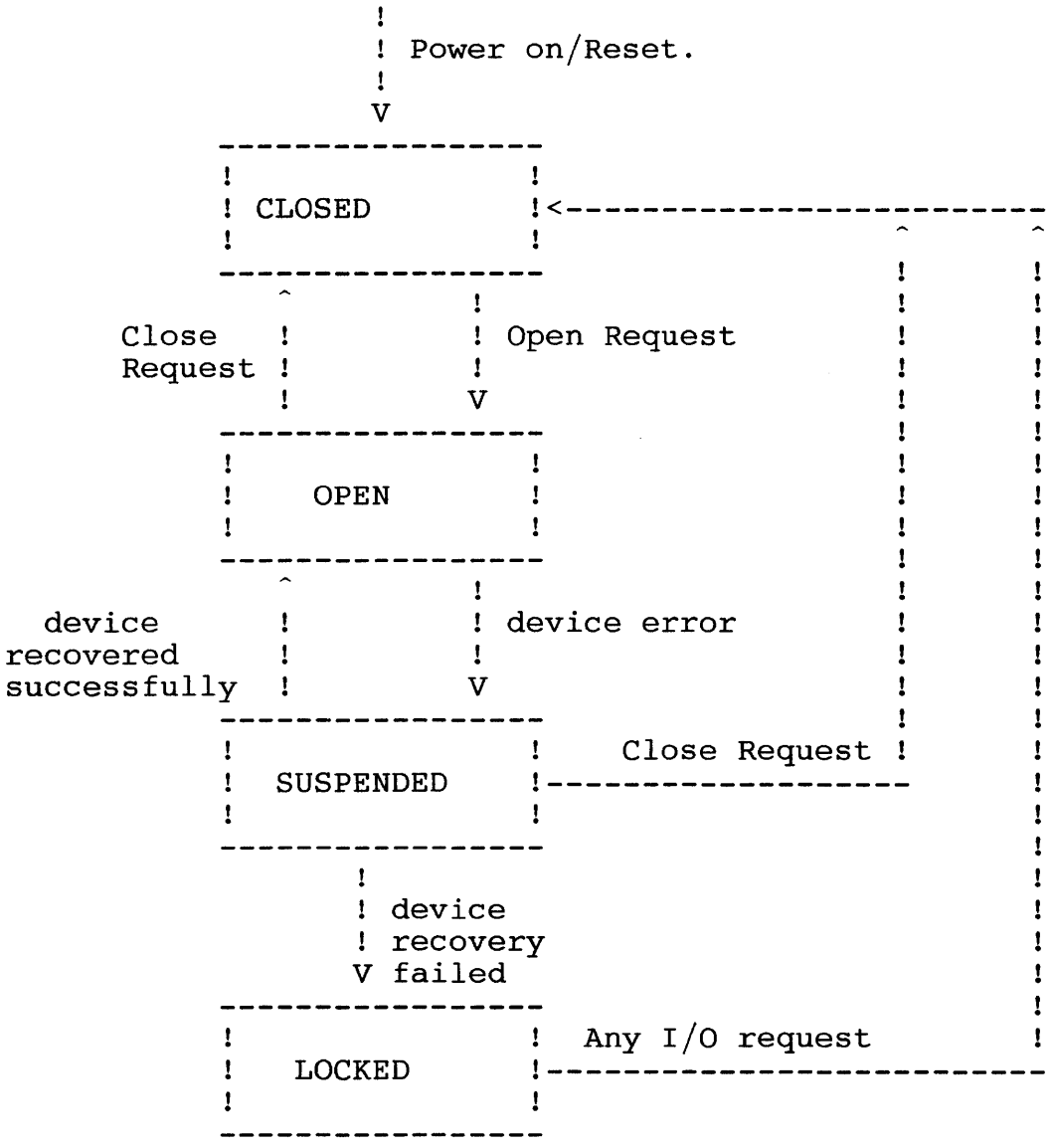
- CLOSED
- OPEN
- SUSPENDED
- LOCKED

The state machine is triggered by 7 events. These are:

- Power on/Reset.
- An open request.
- A close request.
- A device error.
- Successful device recovery.
- Unsuccessful device recovery.
- Any I/O request.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

The state machine can be represented by the following diagram:



The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.2.7.2 Mode

A disk file can be opened in 2 modes, MODE READ and MODE MODIFY. The mode field contains a record of the open mode of a file, and is used to validate access rights when an I/O request is issued.

5.2.7.3 dev_num

The dev_num field contains the device number of the disk device on which the file is located. This field is used to index the volume control structures when a file I/O request is made.

5.2.7.4 fhb_num

The fhb_num field contains the File Header Block number of the file header block which is currently in memory, for this file.

5.2.7.5 last_vda_index

This field is used to speed up file I/O by providing part of a reference point within the file, from which a search for a specific sector may start. This field gives the index into the fhbs' vda field, of the vda in which the last search terminated.

5.2.7.6 last_io_fhb

This field gives the file header number of the Fhb to which the last_vda_index refers.

5.2.7.7 last_lfa

The last_lfa field specifes the lfa, within the file, at which the vda specified by the last_vda_index starts.

5.2.7.8 fhb_pointer

This field is a pointer to the fhb, within the file work area.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.2.8 DEFAULT PATH TABLE

The Default path table contains 4 elements which make up the default access path name. These elements are:

- Volume name.
- Directory name.
- File name prefix.
- Password.

Each of these fields is a standard internal name structure.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.3 NAMES, DESCRIPTIONS, PATHS AND PASSWORDS

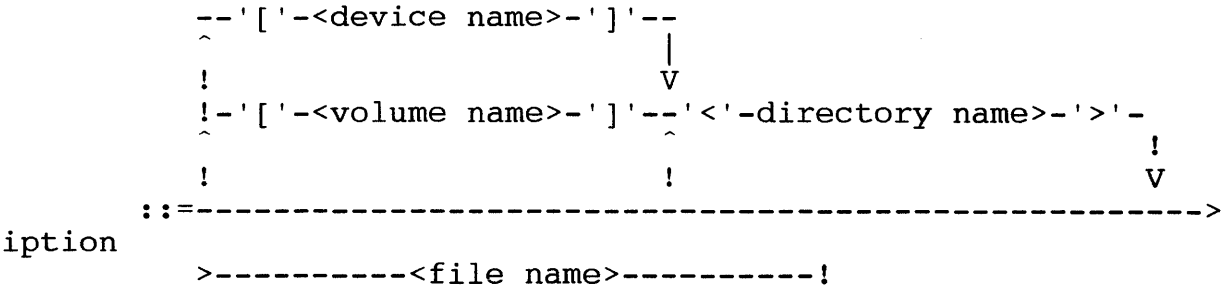
In order to access files on a disk the user must be able to describe the file in terms of the disk structure. The file description must identify the volume or device on which the file resides, the directory in which the file is located, and the name of the file. It is permissible to omit the volume or device name, and the directory name. It is not permissible however to provide a volume or device name, and omit the directory name. The syntax of a file description is given by the following definitions. Note including ']' in volume/device names or '>' in directory names may cause undefined results as these characters are also used as name delimiters.

volume name ::= a string of 1 to 12 characters.

device name ::= a string of 1 to 12 characters.

directory name ::= a string of 1 to 12 characters.

file name ::= a string of 1 to 50 characters.



If the directory name and/or volume/device name are omitted then the default values for the directory, and/or volume/device names will be used. These values are set using the DK_SETPTH_FN, and are initialised to be :

Volume/device name = SYS
Directory name = SYS

If the default directory name is used then the prefix description is also used. The prefix description is used to provide any part of the file description which is the same for all files. However, the prefix description is only used in conjuntion with the default directory name. It is set by the DK_SETPTH_FN, and is initialised to a null string.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Prefix description ::= a string of 0 to 40 characters.

An example of a valid file description would be :

```
[Myvol]<Mydir>Myname
```

Where : The volume name is Myvol
The directory name is Mydir
The file name is Myname.

If the user supplies the following file description:

```
<Mydir>Myname
```

then the default volume/device name will be used, so that the full description of the file would be:

```
[Defaultvol]<Mydir>Myname
```

If the user supplies the file description:

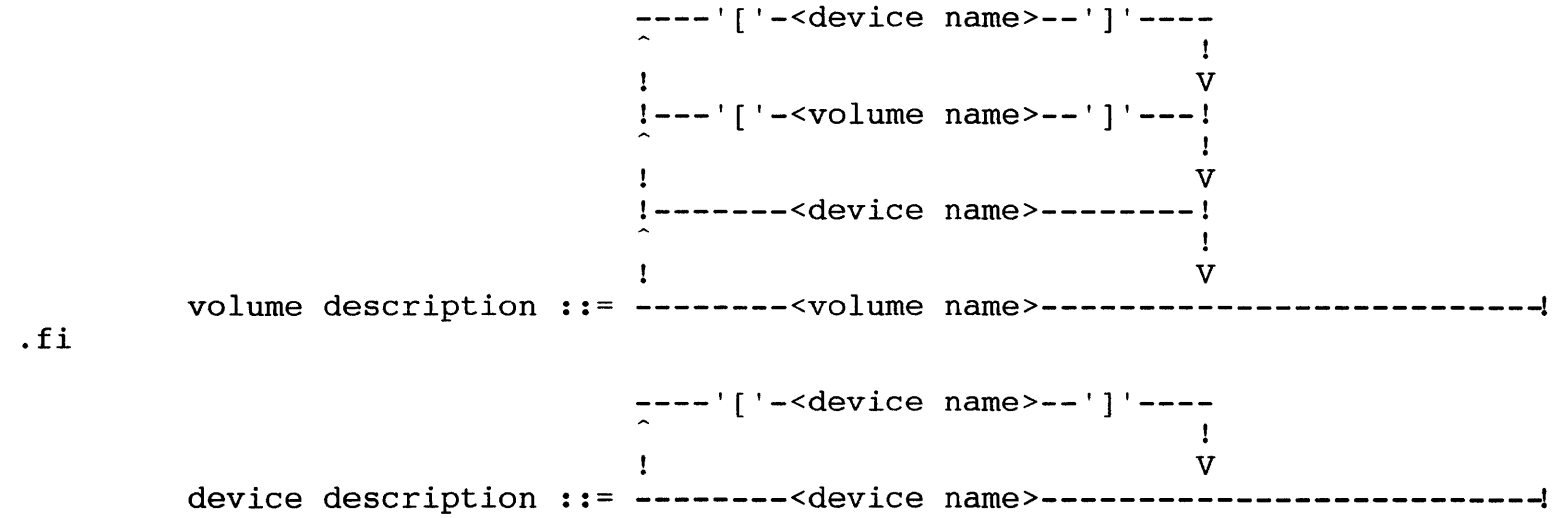
```
Myname
```

then the full file description would be

```
[Defaultvol]<Defaultdir>prefix/Myname
```

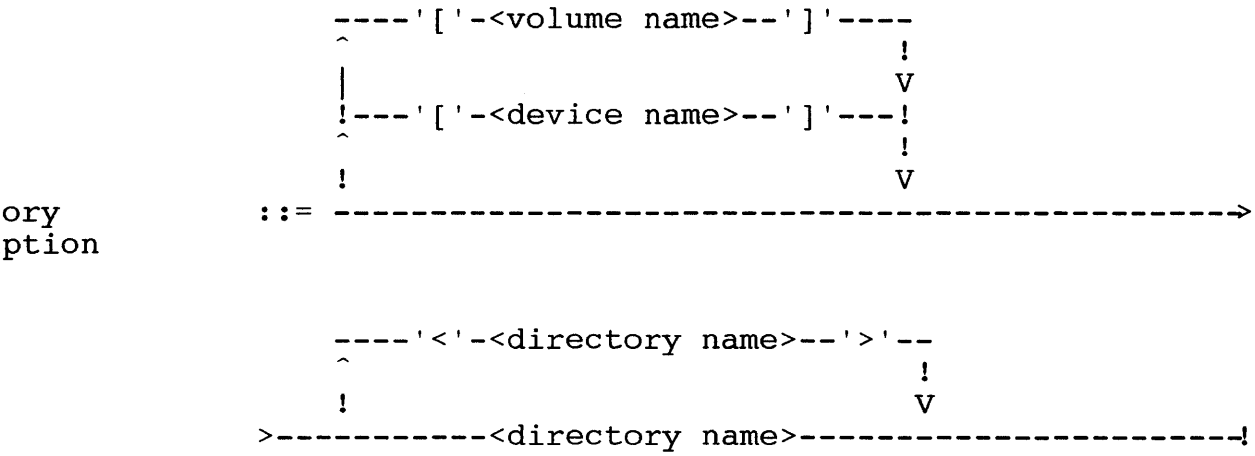
where 'prefix/' is the prefix description.

Some function calls do not require a file description, but they require a volume description, a device description or a directory description. These are described below.



.fi

The information contained within this document is confidential and proprietary to Burroughs Corporation.



Where file, volume or directory descriptions are required, a password is also required. A password is defined as follows

password ::= a string of 0 to 12 characters.

If a zero length password is provided then the default password will be used. The default password can be set using the DK_SETPTH_FN function, and it is initialised to a null string.

No restrictions are placed on the characters used in the above names. It is recommended that only alphabetic and numeric characters are used. Users are warned that use of the volume or directory delimiter characters within a name may result in an incorrect file name or path being generated. The S40 is not case sensitive as far as names are concerned, upper and lower case alphabetic characters being considered equivalent.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.4 STANDARD NAME OR PASSWORD STRUCTURE

A standard filename or password throughout the disk subsystem has the following structure:-

Field	Size (bytes)	Description
Name Length	1	Length of name (in bytes) to follow
Filler	1	Unused
Name	X	Variable size name field of characters up to a maximum of 78 characters if directory and volume names are present. Password may only be 12 characters long maximum.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.5 FILE ACCESS MODES

There are two access modes which are recognised by the disk access routines.

- mode read.

- mode modify.

5.5.1 MODE READ

When a file is opened in mode read, the user is restricted as to what operations can be performed on the file. In mode read, only the following user access functions are permitted on the file (See section on user access functions):

- DK_READ_FN

- DK_CLOSE_FN

- DK_GETST_FN

5.5.2 MODE MODIFY

When a file has been opened in mode modify, all user access functions are valid (See section on User Access Functions).

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.6 DISK CONTROL FUNCTIONS

The Disk Control Functions provide support to the User Access functions, handle all of the positional calculations for access to the disk, and manage all disk control structures. The Disk Control Functions provide the following categories of disk structure management:

- Volume structure management.
- Directory structure management.
- File structure management.
- Disk security management.

These management tasks are further described in the following subsections.

5.6.1 VOLUME STRUCTURE MANAGEMENT

Management of the volume structure requires the code to perform the following tasks:

- Verification of Volume Home Block (Vhb) when a volume is mounted and creation of a working Vhb in memory.
- Validation of the Allocation Bit Map (Abm), and bad block map and creation of a working Allocation Bit Map (Abm) in memory.
- Maintainance of the working Vhb during operation.
- Maintainance of a working Allocation Bit Map (Abm).
- Update of Vhb, and Abm on disk, when files are closed, or checkpoint request is issued.
- Allocation and deallocation of sectors for data. This should be performed with regard to disk and file access efficiency in order to avoid serious fragmentation of the disk.
- Maintainance of volume security when requests such as getVHB are made.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.6.2 DIRECTORY STRUCTURE MANAGEMENT

The code to maintain the directory structure must perform the following functions:

- Validate, and maintain the file Mfd.sys.
- Validate, and maintain the directories that are referenced by Mfd.sys.
- Provide file access information to other routines when requested.
- Maintain directory level security at all times.
- Maintain the LruDirEntries field in the VHB.
- Update directory structures, when a file is created, closed, or when a checkpoint request is made.

5.6.3 FILE STRUCTURE MANAGEMENT

The code to maintain file structures is the code most used in the disk control task, it must perform the following functions:

- Validate, and maintain file headers.
- Read and write data sectors on request.
- Update file headers, when file is closed, or when a checkpoint request is received.
- Maintain file level security at all times

5.6.4 DISK SECURITY MANAGEMENT

There are 3 access levels to a file on disk:

- Volume access
- Directory access
- File access

Files and directories with no password protection and files with protection level 15 can be accessed without knowledge of any password.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.6.4.1 VOLUME ACCESS

Volume level access to a valid password protected volume requires that the volume password be provided. This level of security under B25 BTOS only prevents the disk being re-initialised. Volume level security is designed to control:-

- Disk initialisation.

- Directory creation and deletion.

Use of the volume password should be restricted to those activities which require it, as the volume password overrides all other disk security levels.

5.6.4.2 DIRECTORY ACCESS

Directory level protection is provided by the directory password and the directory protection level. Access to a protected directory requires knowledge of the directory password or volume password.

Directory level protection prevents actions which would modify the directory structure, or provide information regarding the directory structure, i.e.

- Directory deletion.

- Directory listing.

The only valid protection levels for directories are 15,5 and 0. See table below, under File Access.

A file within a protected directory may be accessed using the directory password or file password dependent on the protection level of the file.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.6.4.3 FILE ACCESS

The file protection level controls which types of passwords, if any, are required to gain Read or Modify access to a specific file.

A protection level is assigned only to files. A directory has a default protection level, however, that is used to assign a protection level to each file at the time that the file is created.

The protection level is a number used by BTOS and based on a bit pattern described in table 16. Protection level numbers are not hierarchical.

Nine protection levels are available. Table 15 shows name, number, and type of access allowed for each protection level.

File level access requires knowledge of the file, directory or volume password, in order to access a protected file in different modes, see table below :

File Protection Mode	Volume Password		Directory Password		File Password	
	Read	Write	Read	Write	Read	Write
unprotected(15)	-	-	-	-	-	-
Modify protected(5)	-	Yes	-	Yes	-	No
Access protected(0)	Yes	Yes	Yes	Yes	No	No
Modify Password(7)	-	Yes	-	Yes	-	Yes
Access Password(3)	Yes	Yes	Yes	Yes	Yes	Yes
Read Password(1)	Yes	Yes	Yes	Yes	Yes	No
Nondirectory Modify password(23)	-	Yes	-	No	-	Yes
Nondirectory Access password(19)	Yes	Yes	Yes	No	Yes	Yes
Nondirectory Password(51)	Yes	Yes	No	No	Yes	Yes

Key : Yes means that password allows access.
- means that password is not required for access.
No means that password does not allow access.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

TABLE 15. Protection Levels and Access Passwords

When a file is created, the file protection level is set to the protection level of the directory in which the file is being created.

If a null password exists at the volume and/or directory and/or file level, then this will automatically match any password provided and consequently allow access to the file at that level, unless restricted by the protection level.

The unprotected level allows Read and Modify access without a password.

Three levels allow Read access without a password, but require a password for Modify access. They are:

- Modify
- Modify password
- Non-directory modify password

Five levels require a password for both Read and Modify access. They are:

- Access protected
- Access password
- Read password
- Non-directory password
- Non-directory access password

File passwords are ignored by three of the protection levels; directory passwords are ignored by three levels.

A default file protection level can be assigned to a directory. This does not affect the passwords or the protection of the directory, but is used only as a default level for files created within the directory. If a directory has a password and is assigned the lowest level of protection (15 unprotected), then it is not truely unprotected, since a directory or volume password is still required to create or rename files within that directory. When created, files within that directory are assigned a protection level of 15 unprotected, but the protection level can be changed with the DK_SETST_FN function.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Bit numbers zero through seven

7 6 5 4 3 2 1 0

designate the file protection level, as shown in table 16.

Bit 0:	if the value is 1 and if there is a file password, it is valid for opening with mode 'mr'
Bit 1:	if the value is 1 and if there is a file password, it is valid for opening with mode 'mm'
Bit 2:	if the value is 1, then no password is required for 'mr'
Bit 3:	if the value is 1, then no password is required for 'mm'
Bit 4:	if the value is 0, then directory password is valid for 'mm'
Bit 5:	if the value is 0, then directory password is valid for 'mr'
Bit 6:	reserved for internal use
Bit 7:	reserved for internal use

TABLE 16. Bit Number Designations for Protections Level

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.6.5 MANIPULATION OF DISK CONTROL STRUCTURES

The following subsections describe the tasks of locating, reading validating and modifying the major control structures of the disk.

5.6.5.1 The Volume Home Block

As stated in previous sections, the Vhb is the 'root' structure of the disk. If this is corrupt then it is not possible to operate a normal disk subsystem. The Vhb exists in two forms:

- The Initial Volume Home Block
- The Working Volume Home Block

The Initial Vhb is created when the disk is initialised, and it is never changed during operation. It is however used to locate the Working Vhb on the disk.

The Working Vhb reflects the current status of the disk, though like the Initial Vhb, it originally created during disk initialisation. After initialisation the Working Vhb is coninuously updated during operation so that it will always reflect the true status of the disk.

Before a volume can be accessed by the disk subsystem, both the Initial and Working Vhb must be read in and validated. If either Vhb is invalid then the disk cannot be accessed, and an DK_VHB_INVALID error must be reported to the user.

A Vhb is validated by calculating the sum of the first 128 words of the Vhb. These should add up to the 'Magic Number', as used throughout the B25 disk structure. If it does not, then the validity of the volume is in question, and it cannot be used.

When a field of the Working Vhb is modified, the Working Vhb should be rewritten to the disk. Before this is done, the 'checksum' field of the Vhb must be modified, so that the first 128 words of the Vhb add up to the magic number.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.6.5.2 The Allocation Bit Map

This structure is used to control the distribution of disk space on a disk. Before a disk can be accessed it must have a valid allocation bit map.

The Allocation bit map is located on the disk using fields found in the working volume home block. It is therefore not possible to access the allocation bit map, without first reading in the working volume home block.

The allocation bit map is validated by counting the number of free pages represented in the allocation bit map, and comparing the result with the total number of free pages which the Vhb has a record of. If these two values do are not equal then, then there is an inconsistancy between the working volume home block, and the allocation bit map. As the working Vhb has already been validated, it is assumed that the fault lies with the allocation bit map, so and DK_ABM_INVALID error is reported, and the disk subsystem will not allow the disk to be accessed.

ALLOCATION

When a request is made to increase the size of a file, the control task performs the following actions :

1 - Scan the allocation bit map, from the point where the last allocation stopped, until a free sector is found, or the whole bit map has been scanned.

2 - If no free sectors are found then the largest disk extent found up to this point is returned. This would be zero if the whole allocation bit map where scanned at the first attempt. The function will return the details of the extent, and clear all the bits in the allocation bit map which represent the disk extent.

3 - If a free sector is found then the disk extent is scanned until the end of the bit map is found, or an allocated sector is found. If the disk extent is larger than, or the same size as the requested extent then the function will clear the bits in the allocation bit map, starting at the beginning of the extent, and clearing 1 bit for every sector requested.

    If the extent is smaller than the requested extent, but larger than any other disk extent found in the map, then the location of the extent is recorded, and processing restarts at step 1.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

DEALLOCATION

If a request is made to deallocate a disk extent, then the control task will locate the start of the disk extent in the allocation bit map, and then set all the bits which correspond to the sectors in the disk extent.

5.6.5.3 The Bad Block Files

The Bad Block file is created on a disk when the disk is initialised. It is not used during normal disk operation, however the S40 disk subsystem uses this file, when a volume is to be mounted, as a validity test. After the working volume home block, and allocation bit map have been validated, the bad block file is read, and the bad blocks are checked against the allocation bit map to see if all the bad block are allocated. If any bad block is found to be unallocated in the allocation bit map, then a DK_ABM_INVALID error is reported because an inconsistancy has arisen between the allocation bit map, and the bad block file. The S40 disk subsystem will not allow the disk to be accessed.

If the bad block file passes this consistancy test then the s40 disk subsystem deems the disk to be a valid volume, and will allow I/O requests to be made to the disk.

5.6.5.4 The Directory Entry Block

There is one directory entry block in the file Mfd.sys for every directory on the disk. When the disk subsystem is searching for a directory entry, it will first search the 'least recently used directory entries field in the working Vhb. If the entry is not found then the directory name is put through a hash algorithm, in order to locate the Mfd page, on which the directory entry block should be found.

A directory entry block is created when a create directory request is made, and it is only ever modified by a DK_SETDS_FN call.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7 USER ACCESS FUNCTIONS

This is the lowest level of access available to the application. and provides the following functions to the application, or to access methods:

- Create File (DK_CREATE_FN)
- Delete File (DK_DELETE_FN)
- Open File (DK_OPEN_FN)
- Read Sectors (DK_READ_FN)
- Write Sectors (DK_WRITE_FN)
- Close File (DK_CLOSE_FN)
- Get File Status (DK_GETST_FN)
- Set File Status (DK_SETST_FN)
- Rename File (DK_RNAME_FN)
- Change File Length (DK_CHNGFLEN_FN)
- Close All Files (DK_CLSALL_FN)
- Clear Path (DK_CLRPTH_FN)
- Set Path (DK_SETPTH_FN)
- Get Path (DK_GETPTH_FN)
- Create Directory (DK_CRTDIR_FN)
- Delete Directory (DK_DELDIR_FN)
- Set Directory Status (DK_SETDS_FN)
- Get Directory Status (DK_GETDS_FN)
- Get Volume Home Block (DK_GETVHB_FN)
- Get Stam File Header (DK_GETSFH_FN)

  (N.B Stam = Standard Access Method)

The information contained within this document is confidential and proprietary to Burroughs Corporation.

The functions provided are analogous to functions provided by the B25 BTOS file management procedures, and can be split into 6 categories:

- Allocation functions
- Access functions
- Input/Output functions
- Set Default functions
- Directory functions
- File Utilities

5.7.1 ALLOCATION FUNCTIONS

There are 3 allocation functions:

- Change File Length (DK_CHNGFLEN_FN)
- Create File (DK_CREATE_FN)
- Delete File (DK_DELETE_FN)

These functions are described in the sections which follow.

5.7.1.1 Change File Length (DK_CHNGFLEN_FN)

This function will allocate or deallocate disk sectors with respect to the given file. The user must provide 3 parameters :-

- The file index as returned from a DK_OPEN_FN.
- New file length in bytes, must be a multiple of 512.
- The address of a location where the actual new file length will be returned.

The function must validate the file index, and verify that the file was opened in MODE_MODIFY. If both these items are in order then the new file length is examined to determine whether the file length is to be increased, or decreased.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

If the file length is to be increased, then the allocation bit map is searched for a suitable disk extent. If no disk extent of the requested size is available, then the largest available disk extent will be returned. The 'returned length' location is then set up to indicate the new number of sectors in the file.

If the allocation request would result in the number of disk extents allocated in the current file header block > 32, then a new extension fhb is allocated to the file automatically, provided that a free file header exists.

If the file length is to be decreased then the function will deallocate all, or part of last disk extent in the file, and will continue to do so, until the required number of sectors have been deallocated, or the file length is reduced to zero sectors. As each file header block has all of its disk extents deallocated, the function deallocates that fhb, except for the original fhb which is left allocated, representing a null file.

This function will result in the update of the following control structures:

- The Allocation Bit Map.
- The Volume Home Block.
- The File Header Block(s).

Possible Error Codes : (See appendix A for detailed descriptions.)

N.B. All changes of file length will be applied to the end of the file. Note also that the DK_CHNGFLEN_FN function does not automatically change the end of file pointers, use the DK_SETST_FN to do this.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

DK_ACCESS_DENIED
DK_DEV_NOT_PRESENT
DK_DISK_FULL
DK_FHB_CHAIN_BROKEN
DK_FHB_INVALID
DK_FINDEX_INVALID
DK_FN_NOT_IMPLEMENTED
DK_FRAGMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NOT_READY
DK_NO_FHB_FREE
DK_PROTECTED
DK_VHB_INVALID
DK_WRONG_DISK

Field	Size in bytes	Description
File Index	2	File Index returned from an open request
New File Length	4	New file size in bytes, must be a multiple of 512.
Return Length Address	4	Address of location where actual new file length in bytes is to be returned.

TABLE 17. Parameter list for a DK_CHNGFLEN_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.1.2 Create File (DK_CREATE_FN)

This function will create the necessary entries in the disk control structures for a file, and if requested will attempt to allocate space for the file. The user must provide 4 parameters to the function:

- The name of the file to be created.
- The password to be used to create the file.
- The number of sectors to be allocated in the first disk extent of the file.
- File work area address, used for recovery, reusable after completion.

The function must first check to see if the file already exists. If there is no existing file then the function can attempt to create one. The file is created by allocating a file header block, making a directory entry, and if requested, allocating space to the file.

The function will cause the following disk control structures to be modified :

- File headers file (i.e. allocating a Fhb).
- Directory page.
- Allocation Bit Map.
- Volume Home Block.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Possible Error Codes:

Note that the DK_CREATE_FN does not set the end of file pointer when it creates a file, use DK_SETST_FN to do this. If the initial disk space required for the file cannot be allocated in a single disk extent then DK_DISK_FULL or DK_FRAGMENTED will be reported and the file will not be created. To avoid this problem a file should be created with zero length, and the DK_CHNGFLEN_FN function should be used to set the length.

Possible Error Conditions (See Appendix A for error descriptions):

DK_ABM_INVALID
DK_ACCESS_DENIED
DK_BAD_NAME_SPEC
DK_DEV_NOT_PRESENT
DK_DIR_FULL
DK_DIR_NOT_FOUND
DK_DISK_FULL
DK_FHB_CHAIN_BROKEN
DK_FHB_INVALID
DK_FL_ALREADY_EXISTS
DK_FN_NOT_IMPLEMENTED
DK_FRAGMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NOT_READY
DK_NO_FHB_FREE
DK_PROTECTED
DK_VHB_INVALID
DK_VOL_NOT_MOUNTED
DK_WRONG_DISK

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Field	Size in bytes	Description
File Name Address	4	The address of a standard name structure which contains the file description of the file to be created.
Password Address	4	The address of a standard password structure which contains the password which is to be used to create the file.
FWA Address	4	Address of file work area for this function, reusable once function completed.
File Size	2	Initial size of file to be created, in sectors.

TABLE 18. Parameter list for a DK_CREATE_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.1.3 Delete File (DK_DELETE_FILE)

This function will remove all references to a file from the disk control structures, deallocating the sectors used by the file at the same time. The user must provide only one parameter to the function:

- The file index of the file to be deleted.

In order to remove a file it must first be opened in Mode-Modify. If the file is successfully removed then the delete function will perform an implicit close on the file.

When removing a file the function first checks the file index to make sure that it is valid, and that the file is open in mode modify. If so then the space in use by the file is deallocated by an internal call to the 'change file length' function to change the file length to zero. Then the file header block is deallocated, and the file entry block for the file is removed.

The function will cause the following volume structures to be updated:

- File headers file (i.e. de allocating the Fhb)
- Directory page
- Allocation Bit Map
- Volume home block

Possible Error Conditions (See Appendix A for error descriptions):

DK_ACCESS_DENIED
DK_DEV_NOT_PRESENT
DK_FINDEX_INVALID
DK_FN_NOT_IMPLEMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NOT_READY
DK_PROTECTED
DK_VHB_INVALID
DK_WRONG_DISK

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Field	Size in bytes	Description
File Index	2	File index returned from an open request

TABLE 19. Parameter list for a DK_DELETE_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.2 ACCESS FUNCTIONS

The access functions, are those functions which allow the application to open or close a file. These functions are:

- Close all files (DK_CLSALL_FN)
- Close file (DK_CLOSE_FN)
- Open file (DK_OPEN_FN)

5.7.2.1 Close All Files (DK_CLSALL_FN)

The DK_CLSALL_FN is a special purpose function, which requires no parameters and instructs the lower level of the disk control task to close all open files regardless of errors. After a DK_CLSALL_FN has been executed there will be no open files on any disk. All file work areas will be free for re-use. This function has been provided only for use in exceptional circumstances where unrecoverable disk errors have occurred, to make it possible for the operator to remove an unusable disk and replace it with a good disk. The DK_CLOSE_FN should be used in all normal cases to close files individually.

It should be noted that if DAM files are open on the disk and a DK_CLSALL_FN is called, the DAM files will also be closed. However, the DAM subsystem will not be aware of this, so all DAM work areas should be filled with zeros to prevent DM_DWA_IN_USE errors on reopening DAM files.

N.B. All DAM files will be closed when this function is issued.

Possible Error Conditions (See section 4.9.10 for error descriptions) :

DK_FN_NOT_IMPLEMENTED

This function requires no parameters.

This function may cause the following disk control structures to be modified

- Allocation Bit Map
- Volume Home Block

This does however depend on the state of the volume.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.2.2 Close File (DK_CLOSE_FN)

The DK_CLOSE_FN function will cause a file to be closed by removing all knowledge of the I/O path to that file. Once the I/O path has been purged any further I/O calls to that file will result in an error. The caller must provide 1 parameter:

- The file index.

Possible Error Conditions (See Appendix A for error descriptions):

DK_DEV_NOT_PRESENT
DK_FINDEX_INVALID
DK_FN_NOT_IMPLEMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NOT_READY
DK_PROTECTED
DK_VHB_INVALID
DK_WRONG_DISK

Field	Size in bytes	Description
File Index	2	File index returned from an open request

TABLE 20. Parameter list for a DK_CLOSE_FN call

The function will update the following disk control structures:

- Allocation Bit Map
- Volume home block

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.2.3 Open File(DK_OPEN_FN)

The DK_OPEN_FN is used to establish an I/O path between the application and a specific file on a volume. The caller must provide the following parameters to the function:

- Address of the File name.
- Address of the Password.
- Address of a location where the file index can be placed.
- Address of a file work area which the control task can utilise while the file is open.
- Mode

The File name is used in conjunction with the default path to establish the access path to a file. Once the path name has been established an attempt is made to read the file header block of the file. This may involve reading other control structures from the disk, or these structures may already be in memory.

If the file header block is successfully read from the disk, and it is validated successfully, then the access permission is checked. This is done by checking the supplied password against the access mode. If the password is correct for the files protection level then the file is opened, and the file index is returned to the caller for use with future I/O calls.-

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Possible Error Conditions (See Appendix A for error descriptions):

DK_ABM_INVALID
DK_ACCESS_DENIED
DK_BAD_NAME_SPEC
DK_DEV_NOT_PRESENT
DK_DIR_NOT_FOUND
DK_FHB_INVALID
DK_FL_IN_USE
DK_FL_NOT_FOUND
DK_FN_NOT_IMPLEMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NOT_READY
DK_NO_SLOTS_FREE
DK_PROTECTED
DK_VHB_INVALID
DK_VOL_NOT_MOUNTED
DK_WRONG_DISK

The information contained within this document is confidential
and proprietary to Burroughs Corporation.

Field	Size in bytes	Description
File Name Address	4	The address of a standard name structure which contains the file description of the file to be opened.
Password Address	4	The address of a standard password structure which contains the password to be used to open the file.
File Index Address	4	Location in to which a file index can be placed
File Access Mode	2	Access mode
FWA Address	4	Address of file work area.

TABLE 21. Parameter list for a DK_OPEN_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.3 INPUT/OUTPUT FUNCTIONS

There are 2 Input/Output functions:

- Read (DK_READ_FN)
- Write (DK_WRITE_FN)

5.7.3.1 Read (DK_READ_FN)

The DK_READ_FN function will read a number of logically contiguous 512 byte sectors from a disk. The user must supply the following parameters to the function:

- The file index that was returned from the open request for the file.
- The byte offset from the first byte of the file, at which the read operation is to commence. This must be a multiple of 512.
- The number of sectors which are to be read.
- The number of additional retry attempts which are to be made, over and above the default of 3, in the event of an i/o failure.
- The address of a standard data buffer, into which the data is to be placed.

The read function will then convert the supplied offset into a physical disk address, and read in the requested number of sectors. If the logically contiguous sectors are split over one or more disk extents, then the read function will automatically deal with this. This allows the user to consider a disk file as a logically contiguous array of 512 byte sectors.

In order to perform a read, the function must access the following disk structures:

- File Header Block.
- Extension File Header Block(s) (if there are any associated with the file).

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Possible Error Conditions (See Appendix A for error descriptions):

DK_DEV_NOT_PRESENT
DK_EOM_ENCOUNTERED
DK_FHB_INVALID
DK_FINDEX_INVALID
DK_FN_NOT_IMPLEMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NON_ALIGNED_BUFF
DK_NOT_READY
DK_VHB_INVALID
DK_WRONG_DISK

Field	Size in bytes	Description
File Index	2	File Index returned from an open request
Buffer Address	4	Address of S40 standard data buffer
Number of Sectors	2	Number of sectors to be read.
File Offset	4	Offset in bytes from start of file (must be a multiple of 512)
Number of Retries	1	Number of supplimentary retries made by disk controller in the event of a read error. Normally set to 0, as the 3 automatic retries made in addition to these is considered sufficient.

TABLE 22. Parameter list for a DK_READ_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.3.2 Write (DK_WRITE_FN)

The DK_WRITE_FN function will write a number of logically contiguous 512 byte sectors to a disk. The user must supply the following parameters to the function:

- The file index that was returned from the open request for the file.
- The byte offset from the first byte of the file, at which the write operation is to commence. This must be a multiple of 512.
- The number of sectors which are to be written.
- The number of additional retry attempts which are to be made, over and above the default of 3, in the event of an i/o failure.
- The address of a standard data buffer, from which the data is to be taken.

The write function will convert the supplied offset into a physical disk address, and write the specified number of sectors to the disk. If the logically contiguous sectors are split over one or more disk extents then the write function will automatically deal with this.

In order to perform a write, the function must access the following disk structures:

- File Header Block.
- Extension File Header Block(s) (if there are any associated with the file).

Possible Error Conditions (See Appendix A for error descriptions):

DK_ACCESS_DENIED
DK_DEV_NOT_PRESENT
DK_EOM_ENCOUNTERED
DK_FHB_INALID
DK_FINDEX_INVALID
DK_FN_NOT_IMPLEMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NON_ALIGNED_BUFF
DK_NOT_READY
DK_PROTECTED
DK_VHB_INVALID
DK_WRONG_DISK

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Field	Size in bytes	Description
File Index	2	File Index returned from an open request.
Buffer Address	4	Address of S40 standard data buffer
Number of Sectors	2	Number of sectors to write.
File Offset	4	Offset in bytes from start of file (must be a multiple of 512)
Number of Retries	1	Number of supplimentary retries made by disk controller in the event of a read error. Normally set to 0, as the 3 automatic retries made in addition to these is considered sufficient.
Options	1	Read after write (RAW) verify check and data mark indicators, normally set to 3. Bit 0:- 1 - RAW verify 0 - No verify Bit 1:- 1 - Write data mark 0 - Write deleted data mark

TABLE 23. Parameter list for a DK_WRITE_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.4 SET DEFAULTS FUNCTIONS

The functions which come under this category are those which allow the user to change a number of default subsystem parameters. There are 3 functions under this heading:

- Clear Path (DK_CLRPTH_FN)
- Set Path (DK_SETPTH_FN)
- Get Path (DK_GETPTH_FN)

5.7.4.1 Clear Path

The ClearPath function clears the values set by the SetPath, function. The values are returned to the default values of:

```
Volume name = 'SYS'
Directory name = 'SYS'
Password = NULL
Prefix = NULL
```

There are no parameters for this function.

No disk control structures are altered by this function. Also path validation is not performed.

Possible Error Conditions (See Appendix A for error descriptions):

DK_FN_NOT_IMPLEMENTED

5.7.4.2 Set Path

The SetPath function sets up the default values for volume, directory, password and prefix. The caller must provide the following parameters:

- The address of a structure containing the default volume name.
- The address of a structure containing the default directory name.
- The address of a structure containing the default password.
- The address of a structure containing the default file prefix.

This function does not alter any disk control structures. However the path is validated, and warnings may be returned. These warnings are for information only, and will not affect the setting up of the default path.

The information contained within this document is confidential
and proprietary to Burroughs Corporation.

Possible Error Conditions (See Appendix A for error descriptions):

DK_ABM_INVALID
DK_BAD_NAME_SPEC
DK_DEV_NOT_PRESENT
DK_DIR_NOT_FOUND
DK_FN_NOT_IMPLEMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NOT_READY
DK_VHB_INVALID
DK_VOL_NOT_MOUNTED
DK_WRONG_DISK

The information contained within this document is confidential
and proprietary to Burroughs Corporation.

Field	Size in bytes	Description
Volume Name Address	4	The address of a standard name structure which contains the volume name to be used for the default.
Directory Name Address	4	The address of a standard name structure which contains the directory name to be used for the default.
Password Address	4	The address of a standard password structure which contains the password to be used for the default.
Prefix Address	4	The address of a standard name structure which contains the prefix description which is to be used.

TABLE 24. Parameter list for a DK_SETPTH_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.4.3 Get Path

The GetPath function returns the current default values for volume, directory, password and prefix. The caller must provide the following parameters:

- The address of a structure to contain the default volume name.
- The address of a structure to contain the default directory name.
- The address of a structure to contain the default password.
- The address of a structure to contain the default file prefix.

This function does not alter any disk control structures.

Possible Error Conditions (See Appendix A for error descriptions):

DK_FN_NOT_IMPLEMENTED

Field	Size in bytes	Description
Volume Name Address	4	The address of a standard name structure in to which the default volume name can be placed.
Directory Name Address	4	The address of a standard name structure in to which the default directory name can be placed.
Password Address	4	The address of a standard password structure in to which the default password can be placed.
Prefix Address	4	The address of a standard name structure in to which the default prefix description can be placed.

TABLE 25. Parameter list for a DK_GETPTH_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.5 DIRECTORY FUNCTIONS

The Directory functions are utility functions for creating, and deleting directories. There are 2 functions:

- Create Directory (DK_CRTDIR_FN)
- Delete Directory (DK_DELDIR_FN)

5.7.5.1 Create Directory (DK_CRTDIR_FN)

The create directory function will set up the necessary entries in the disk control structures for a directory to become usable. The user must supply 5 parameters to the function:

- The address of a standard name structure which contains the directory description.
- The address of a standard password structure which contains the directory password to be given to the directory.
- The address of a standard password structure which contains the volume password to be used to create the directory.
- The maximum number of files which the user intends to create in the directory.
- The default file protection level.

The function will then check to see if the directory exists, and if not then the disk subsystem will create a directory entry block in the mfd, and allocate a single disk extent to the directory.

The function will cause the following structures to be updated

- mfd.sys (i.e. the Deb)
- Allocation Bit Map
- Volume Home Block

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Possible Error Conditions (See Appendix A for error descriptions):

DK_ABM_INVALID
DK_ACCESS_DENIED
DK_ACC_CODE_INVALID
DK_BAD_NAME_SPEC
DK_DEV_NOT_PRESENT
DK_DIR_ALREADY_EXISTS
DK_DISK_FULL
DK_FN_NOT_IMPLEMENTED
DK_FRAGMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_MFD_FULL
DK_NOT_READY
DK_PROTECTED
DK_VHB_INVALID
DK_VOL_NOT_MOUNTED
DK_WRONG_DISK

The information contained within this document is confidential
and proprietary to Burroughs Corporation.

Field	Size in bytes	Description
Directory Name Address	4	Address of a standard name structure which contains the directory description to be used to create the directory.
Directory Password Address	4	Address of a standard password structure which contains the directory password to be used.
Volume Password Address	4	Address of a standard password structure which contains the volume password to be used.
Maximum Files	2	Maximum number of files in the directory.
File Protection level	2	Default file protection level for files created within this directory.

TABLE 26. Parameter list for a DK_CRTDIR_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.5.2 Delete Directory (DK_DELDIR_FN)

The delete directory function will remove all references to a directory if the directory does not contain any files. The user must supply 2 parameters to the function:

- The address of a standard name structure which contains the directory description of the directory to be deleted.

- The address of a standard password structure which contains the password of either the volume or the directory.

The function will check to see if the directory exists, and if it is empty. If the directory exists and is empty then directory pages are deallocated, and the directory entry block in the mfd is deleted. The function will cause the following disk structures to be modified:

- mfd.sys (for the Deb)

- Allocation bit map

- Volume home block.

Possible Error Conditions (See Appendix A for error descriptions):

DK_ABM_INVALID
DK_ACCESS_DENIED
DK_BAD_NAME_SPEC
DK_DEV_NOT_PRESENT
DK_DIR_IS_NOT_EMPTY
DK_DIR_NOT_FOUND
DK_FN_NOT_IMPLEMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NOT_READY
DK_PROTECTED
DK_VHB_INVALID
DK_VOL_NOT_MOUNTED
DK_WRONG_DISK

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Field	Size in bytes	Description
Directory Name Address	4	Address of a standard name structure which contains the directory description of the directory to be deleted.
Password Address	4	Address of a standard password structure which contains the volume or directory password to be used to delete the directory.

TABLE 27. Parameter list for a DK_DELDIR_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.6 FILE SYSTEM UTILITY FUNCTIONS

The File System Utility functions are most likely to be used for file system maintenance, rather than during normal operation. There are 5 functions in this catagory:

- Get File Status (DK_GETST_FN)
- Set File Status (DK_SETST_FN)
- Rename File (DK_RNAME_FN)
- Set Directory Status (DK_SETDS_FN)
- Get Directory Status (DK_GETDS_FN)
- Get Volume Home Block (DK_GETVHB_FN)
- Get Stam File Header (DK_GETSFH_FN)

5.7.6.1 Get File Status (DK_GETST_FN)

This function will return certain details which are associated with a file. Using this function the following items of file status information can be obtained:

- The physical file length in bytes which will always be a complete sector boundary.
- The contents of the file type field in the file header block.
- The contents of the file protection level field of the file header block.
- The file password.
- The Date/Time of creation in S40 date time format.
- The Date/Time last modified in S40 date time format.
- The contents of the End-of-File pointer field in the file header block.
- A complete copy of the File Header Block except for the password.
- a complete copy of the Volume Home Block of the volume on which the file resides except for the volume password.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

- A copy of the application specific field in the file header block.

The user must supply 4 parameters.

- The file index which was returned from an open file function call.

- The status code which identifies what information is required. These are detailed in the table below.

- The address of an area of memory where the status information is to be put.

- The size of the memory area. This is supplied because for some items the amount of information which is returned can be varied. The sizes are specified in table 29 below.

The function will then read the appropriate control structure and transfer the requested data in to the user supplied area.

No disk control structures are modified.

Possible Error Conditions (See Appendix A for error descriptions):

DK_DEV_NOT_PRESENT
DK_FHB_INVALID
DK_FINDEX_INVALID
DK_FN_NOT_IMPLEMENTED
DL_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NOT_READY
DK_VHB_INVALID
DK_WRONG_DISK

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Field	Size in bytes	Description
File Index	2	File index returned from open request.
Status Code	2	See Table below.
Status Area Address	4	Address of area where file status is to be returned.
Status Area Size	2	Size of status area. (refer to table below)

TABLE 28. Parameter list for a DK_GETST_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Status Codes Table

CODE	ITEM	SIZE (BYTES)	FORMAT
0	File length	4	INTEGER4
1	File type	1	***
2	File protection level	1	Unsigned byte
3	Password	13	See 5.4
4	Date/time of creation	7	**
5	Date/time last modified	7	**
6	End-of-file pointer	4	INTEGER4
7	File Header Block	512	*
8	Volume Home Block	256	*
9	Invalid		
10	Application-specific field in the File Header Block	64	User defined

* Note that the password field of the FHB or VHB is blanked to nulls on return, see layout of FHB and VHB below.

** The date/time format returned here is described in the section regarding S40 date/time format in ref 2.

*** File type or file class (see section 5.1.9.9.) is a field in the file header block of type "unsigned bytes", which is not used by the S40 and will in all default cases (eg. file creation) be set to zero.

TABLE 29. Status Codes for a DK_GETST_FN call

B25 Date/Time Format

The following is a description of the date/time format used in the VHB and FHB, to be B25 BTOS compatible.

The date/time format is represented in 32 bits to an accuracy of one second. The high-order 15 bits of the high-order word contain the count of days since March 1, 1952. The use of a 15-bit field allows dates up to the year 2042 to be represented. The low-order bit of the high-order word is 0 to represent AM or 1 to represent PM. The low-order word contains the count of seconds since midnight/noon. Valid values are 0 to 43199.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.6.2 Set File Status (DK_SETST_FN)

This function will accept information provided by the caller, and use that information to update certain items which go to make up the status of a file. These items are:

- The contents of the file type field in the file header block.
- The contents of the file protection level field in the file header block.
- The file password.
- The date/time of creation.
- The date/time of modification.
- The contents of the End-of-File pointer field in the file header block.
- The application specific field in the file header block.

The user must supply 4 parameters:-

- The file index which was returned from an open file function call.
- The status code which identifies what information is required. These are detailed in the table below.
- The address of an area of memory which contains the value to which the status item is to be set.
- The size of the memory area. This is supplied as the amount of data supplied for some items can be varied.

The function will then set up the appropriate items in the control structures, and then re-write them.

This function causes the following control structures to be updated.

- Fileheaders.sys (i.e. the Fhb).

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Possible Error Conditions (See Appendix A for error descriptions):

DK_ACCESS_DENIED
DK_DEV_NOT_PRESENT
DK_FHB_INVALID
DK_FINDEX_INVALID
DK_FN_NOT_IMPLEMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NOT_READY
DK_PROTECTED
DK_VHB_INVALID
DK_WRONG_DISK

Field	Size in bytes	Description
File Index	2	File index returned by open request.
Status Code	2	See table below.
Status Area Address	4	Address of area containing new file status.
Status Area Size	2	Size of status area. (refer to table below)

TABLE 30. Parameter list for a DK_SETST_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Status Code Table

CODE	ITEM	SIZE (BYTES)	FORMAT
1	File type	1	***
2	File protection level	1	Unsigned byte
3	Password	*	See 5.4
4	Date/time of creation	7	**
5	Date/time last modified	7	**
6	End-of-file pointer	4	INTEGER4
7	invalid		
8	invalid		
9	invalid		
10	Application-specific field in the File Header Block	64	User defined

* The length of the password field is variable.

** The date/time format to be provided here is described in the section regarding S40 date/time format in ref 2. The date/time provided is checked to ensure that the individual bytes lie in the legal range, as defined in the reference given above, if not then the date/time is set to zero's indicating date/time not valid and IO_SUCCESS is returned.

*** Files type is a field in the file header block which is not used by the S40 and will in all default cases (eg. file creation) be set to zero.

TABLE 31. Status codes for a DK_SETST_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.6.3 Rename File (DK_RENAME_FN)

The Rename File function changes the name of an open file to a new, user supplied name. The user must supply 3 parameters to the function:

- The file index of the file to be renamed, as returned from the open function call. Note that the file must be opened in mode modify.

- The address of a standard name structure which holds the new file name.

- The address of a standard password structure which contains the password to be used to rename the file.

The function will then modify the directory pages, and the file header block of the file to reflect the new name.

The rename function will cause the following disk control structures to be modified:

- Fileheaders.sys (i.e. the Fhb)

- Old directory page

- New directory page (may be same as old directory page)

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Possible Error Conditions (See Appendix A for error descriptions):

DK_ACCESS_DENIED
DK_BAD_NAME_SPEC
DK_DEV_NOT_PRESENT
DK_DIR_NOT_FOUND
DK_DIR_FULL
DK_FHB_INVALID
DK_FINDEX_INVALID
DK_FL_ALREADY_EXISTS
DK_FN_NOT_IMPLEMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NOT_READY
DK_PROTECTED
DK_VHB_INVALID
DK_VOL_NOT_MOUNTED
DK_WRONG_DISK

Field	Size in bytes	Description
File Index	2	File index of the file to be re-named.
New Name Address	4	Address of a standard name structure which contains the file description to which the is to be renamed.
Password Address	4	Address of a standard password structure which contains the password to be used to rename the file.

TABLE 32. Parameter list for a DK_RNAME_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.6.4 Set Directory Status (DK_SETDS_FN)

This function will modify the following fields of a directory entry block:

- The directory password field
- The default file protection level

The user must supply 5 parameters:

- The address of a standard name structure containing the directory description.
- The address of a standard password structure which contains the current directory or volume password.
- The status code which defines what is to be changed. These are defined in the table below.
- The address of a memory area where the new value of the field is held.
- The size of the memory area.

The system will then update the fields in the directory entry block.

This function modifies the following structures

- Mfd.sys (i.e. the Deb).

Possible Error Conditions (See Appendix A for error descriptions):

DK_ABM_INVALID
DK_ACCESS_DENIED
DK_ACC_CODE_INVALID
DK_BAD_NAME_SPEC
DK_DEV_NOT_PRESENT
DK_DIR_NOT_FOUND
DK_FN_NOT_IMPLEMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NOT_READY
DK_PROTECTED
DK_VHB_INVALID
DK_VOL_NOT_MOUNTED
DK_WRONG_DISK

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Field	Size in bytes	Description
Directory Name Address	4	Address of a standard name structure containing the directory description of the directory for which status is to be set.
Password Address	4	Address of a standard password structure containing the volume or directory password to be used to set the status.
Status Code	2	See table below.
Status Area Address	4	Address of area containing new directory status.
Status Area Size	2	Size of status area. (refer to table below)

TABLE 33. Parameter list for a DK_SETDS_FN call

Status Code Table

CODE	ITEM	SIZE (BYTES)	FORMAT
0	invalid		
1	invalid		
2	Default file protection level	1	Unsigned byte
3	Password	*	See 5.4

* The length of the password field is variable.

TABLE 34. Status codes for a DK_ SETDS_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.6.5 Get Directory Status (DK_GETDS_FN)

This function will return the value of certain field found in the directory entry block for a given directory. These fields are:

- Directory size
- Default file protection level
- Password

The user must supply 5 parameters to the function which are described below.

This does not modify any of the disk control structures

Possible Error Conditions (See Appendix A for error descriptions):

DK_ABM_INVALID
DK_BAD_NAME_SPEC
DK_DEV_NOT_PRESENT
DK_DIR_NOT_FOUND
DK_FN_NOT_IMPLEMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NOT_READY
DK_VHB_INVALID
DK_VOL_NOT_MOUNTED
DK_WRONG_DISK

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Field	Size in bytes	Description
Directory Name Address	4	Address of a standard name structure which contains the directory description of the directory for which the status is required.
Password Address	4	Address of a standard password structure which contains the volume or directory password to be used.
Status Code	2	See table below.
Status Area Address	4	Address of area where directory status is to be returned.
Status Area Size	2	The size of the area in to which the data is to be placed. (refer to table below).

TABLE 35. Parameter list for a DK_GETDS_FN call

Status Codes Table

CODE	ITEM	SIZE (BYTES)	FORMAT
0	Directory Size	4	INTEGER4
1	Invalid		
2	Default file Protection Level	1	Unsigned byte
3	Password	13	See 5.4

TABLE 36. Status codes for a DK_GETDS_FN

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.6.6 Get Volume Home Block (DK_GETVHB_FN)

This function will supply the caller with a copy of the volume home block of a disk with the volume password set to nulls.

The caller must provide 3 parameters which are described below:

No volume control structures are modified.

Possible Error Conditions (See Appendix A for error descriptions):

DK_ABM_INVALID
DK_DEV_NOT_PRESENT
DK_FN_NOT_IMPLEMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NOT_READY
DK_VHB_INVALID
DK_VOL_NOT_MOUNTED
DK_WRONG_DISK

Field	Size in bytes	Description
VHB Address	4	The address of the area into which the Vhb is to be copied.
Number of Bytes	2	Number of bytes to be copied from Vhb.
Volume Name Address	4	The address of a standard name structure which contains the volume description of the volume for which the VHB is required.

TABLE 37. Parameter list for a DK_GETVHB_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.7.6.7 Get Standard Access Method File Header

This function will open a file, verify that it was created by a standard access method, return the first 512 bytes to the caller and close the file.

The caller must provide 3 parameters which are described below.

This function does not modify any of the disk control structures.

Possible Error Conditions (See Appendix A for error descriptions):

DK_ABM_INVALID
DK_ACCESS_DENIED
DK_BAD_NAME_SPEC
DK_DEV_NOT_PRESENT
DK_DIR_NOT_FOUND
DK_FHB_INVALID
DK_FL_NOT_FOUND
DK_FN_NOT_IMPLEMENTED
DK_HARD_ERROR
DK_INV_VOL_NAME
DK_IO_ERROR
DK_NON_ALIGNED_BUFF
DK_NOT_READY
DK_STAM_FH_INVALID
DK_VHB_INVALID
DK_VOL_NOT_MOUNTED
DK_WRONG_DISK

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Field	Size in bytes	Description
File Name Address	4	Address of a standard name structure which contains the file description of the file for which the STAM file header is required.
Password Address	4	Address of a standard password structure which contains the password to be used to obtain the STAM file header.
File Header Area Address	4	Address of word-aligned area where file header is to be returned (512 bytes).

TABLE 38. Parameter list for a DK_GETSFH_FN call

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.8 UTILITIES INTERFACE

The UTILITIES INTERFACE provides a set of functions for the use of the disk utility programs provided for the S40. The interface is conditionally compiled into the disk subsystem control task in order to avoid duplicating code in the utilities and to reduce the operating system size for the normal application build.

The interface consists of 7 functions. These are:

DU_READ_FN

DU_WRITE_FN

DU_FORMAT_FN

DU_CREATE_FN

DU_TIME_FN

DU_GETSYSDATA_FN

DU_ALLOC_FN

N . B These functions allow access to the disk in a manner which will bypass normal disk security checks, and will allow a utility to read and write to physical disk addresses. In all cases it is assumed that the caller understands the consequences of the actions performed.

Use of these functions should be carefully controlled, they will not be required to be called from normal application programs.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.8.1 DU_READ_FN

The DU_READ_FN function will allow up to 127 contiguous sectors of disk to be read in one operation. The parameters to this function are all given in the following table.

Field	Size	Description
Dev_Num	2	The number of the device to which this I/O request is directed
Pfa	4	The physical disk address
len	2	The number of 512 byte sectors to read
rtry	2	The number of disk retries to perform before an error is reported, in the event of a disk I/O error
buf	4	The address of a standard data buffer

Possible Errors:

T.B.S.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.8.2 DU_WRITE_FN

The DU_WRITE_FN will allow up to 127 contiguous sections to be written to the disk in one operation. The parameters are given in the following table

Field	Size	Description
Dev_Num	2	The number of the device to which this I/O request is directed
Pfa	4	The physical disk address
len	2	The number of 512 byte sectors to read
rtry	2	The number of disk retries to perform before an error is reported, in the event of a disk I/O error
buf	4	The address of a standard data buffer
options	2	This parameter controls the verification process, and the data mark

Possible Errors:-

T.B.S.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.8.3 DU_FORMAT_FN

The DU_FORMAT_FN will format and/or surface test a disk, and report any bad sectors found during surface testing. The function does not check for a valid volume before formating. It is assumed that the caller has already done this.

The parameters for the function are:-

Field	Size	Description
Surface_tests	1	Number of surface test to be performed
Suppress_format	1	Surface test only flag
Dev_Num	2	The device number of the device to be formatted
bad	4	The address of the bad block data buffer

Bad block data is returned as a 512 byte block of data. The data is structured as an array of up to 128 sector addresses where byte 0 to 3 contains the first sector address 4 - 7 the second sector address etc.

A sector address of FFFFFFFF hexadecimal indicates the end of bad sector data in the data block.

Possible Errors:

T.B.S.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.8.4 DU_CREATE_FN

The DU_CREATE_FN will create a file, making a file entry in the appropriate directory, and allocating the file header block. It will also allow the caller to set the FNOSAVE, FNODELECT, and FNODIR PRINT flag's, and set up vda, and runlength entries without using the normal routine used by the control task.

The parameters are:-

Field	Size	Description
File name	4	The address of the file name for which the flag is to be created
Password	4	The address of the file password
FNOSAVE	1	The no save flag. (Can be 00 or FF hexadecimal)
FNODELETE	1	The no delete flag. (Can be 00 or FF hexadecimal)
FNODIRPRINT	1	The no direct print flag. (Can be 00 or FF hexadecimal)
Vda	4	Disk address of intial disk extent
Runlength	4	Size of initial disk entent. (If zero then no initial disk extent

Possilbe Errors:

T.B.S.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.8.5 DU_TIMECONVERT_FN

The DU_TIMECONVERT_FN provides access to the disk control tasks time conversion routines, which will allow the caller to convert between B25 and S40 standard time formats. For more details see the section on Time Conversion and Time Stamping.

Parameters:-

Field	Size	Description
Direction Flag	1	Direction of convertion 0 = B25 to S40 1 = S40 to B25
B25_Time	4	Address of a B25 time structure
S40_Time	4	Address of an S40 time structure

Possible Errors: None

The information contained within this document is confidential and proprietary to Burroughs Corporation.

5.8.6 DU_GETSYS_DATA_FN

The function will return the device control block for a given device name.

Parameters:-

Field	Size	Description
Device name	4	Address of the device name
DCB	4	Address of a device control block area (See Section 5.2.1)

Possible Errors: OK_DEV_NOT_PRESENT

The information contained within this document is confidential and proprietary to Burroughs Corporation.

6. MAINTAINANCE LOGGING

The S40 disk subsystem control task will log the following information in the maintenance log counters.

- Number of Read Retries due to CRC errors, or record not found.
- Number of Write Retries due to write faults or verification errors.
- Number of Seek Retries due to seek errors.
- Number of disk hard errors.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

7. TIME CONVERSION AND TIME STAMPING

The time stamps found within the B25 disk structures are not in the format that is used by the S40. To compensate for this all date/time stamps will be automatically converted within the disk subsystem:

- from B25 format to S40 format for functions which return a time stamp.

- from S40 format to B25 format for functions which set a time stamp.

N.B. For functions which read structures containing one or more time stamps, no conversion will take place. It will be the users responsibility to make any conversions that they require. Details of the time formats are given in appendix C.

7.1 TIME STAMPING

The disk control structures will be time stamped in the following manner.

Vhb -
- Creation date will be set when the volume is introduced.

- Modification date will be set every time the volume home block is written to disk.

Fhb -
- Creationbdate will be set when then File is created.

- Access date will be set every time the file is opened unless write protected.

- Modification date will be set every time the file is written to disk unless mode = MODE READ.

- Fix printer date is never set.

N.B. If the system clock is not set then the time will be invalid.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

8. ERROR RECOVERY

The S40 disk subsystem has been designed to be robust with regard to dealing with, and/or avoiding operator, or programmer errors. The objective has been to build in to the subsystem, automatic recovery where ever possible, and to reduce application/operator intervention to a minimum when automatic recovery fails, or is not possible.

The error conditions which are dealt with are:

- Failure of the S40 - Disk device data communications link.
- Door open/ device not ready failures.
- Power failure of the disk.
- Disk removal/reinsertion/swap.
- I/O failure of the disk hardware or media.

If the initial, automatic recovery attempts fail, then the problem will be reported to the application, and the subsystem will not make any further recovery attempts until the application re-issues the I/O request.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

9. DEBUG INFORMATION AND EVENT TRACING

The disk control task will trace all abnormal events which occur within the system, i.e. any request which does not terminate with a successful status will cause information to be traced. The contents of this trace information will be very much context dependant, and in some cases may well extend over more than one trace buffer entry.

e.g. If an error status is returned from the disk drive for any reason the control task will trace the 10 bytes of the command parameters, followed by the 10 bytes of the return status. If there is any other relevant information this will be traced in the following trace buffers.

For debugging purposes there will be conditionally compiled code in the control task which will perform much more detailed tracing, including lfa/pfa conversions, path finding, and any other items which may be deemed to be of use.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

10. DEVICE DRIVER

This section specifies the Disk Driver of the S40 Disk Sub-system. It describes the interface to the Disk Control Task and the interface to the Disk Controller.

10.1 GENERAL OPERATION

The Driver is designed to be an unintelligent link between the Control Task and the Controller. The Driver only has a knowledge of the communications link and the communications protocol. It knows what command parameters and completion statuses are, but does not know or interpret data within them.

The only decisions the Driver can make concern the breakdown of the communications interface. The Driver attempts to maintain a robust interface by retrying up to three times any command which apparently fails because the interface protocol is broken. If recovery is made then the error will be transparent to the Control Task. Hardware failures are not retried. Where a failure cannot be recovered the Driver reports the error to the Control Task.

When a communications failure is reported to the Control Task the Disk Driver takes no further recovery action. Any further action must be dictated by the Control Task.

10.2 POSITION OF THE DRIVER WITHIN DISK SUB-SYSTEM

The Driver is a distinct layer in the Disk Sub-system. All communication between the Controller and the Control Task must pass through the Driver. The flow of information is shown in the figure below.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

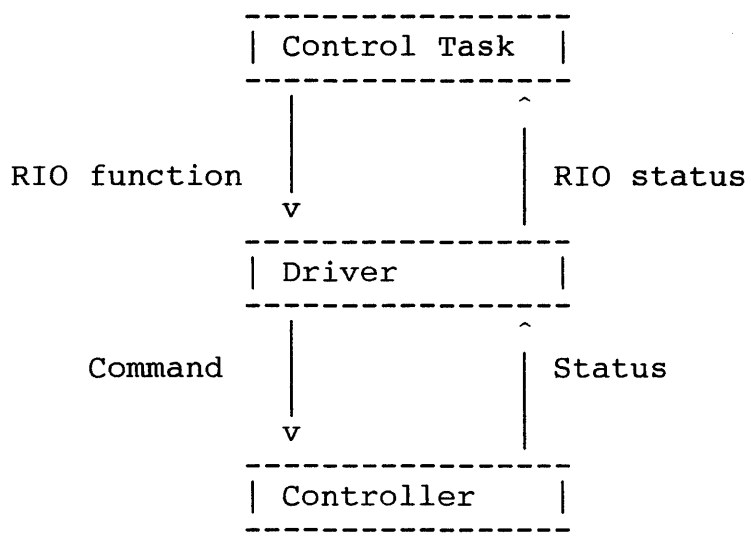


Figure 10. Driver Position and Information Flow

The information contained within this document is confidential and proprietary to Burroughs Corporation.

10.3 INTERFACE TO DISK CONTROL TASK

10.3.1 RIO FUNCTIONS

For each RIO function available in the Driver there is a corresponding command to be issued to the disk device, with the exception of the C_RESET function. The commands to the Controller take the form of a single byte. The Parameter Data always takes the form of a ten byte block, as does the Status Data. The commands and their associated Parameter and Status Data are documented in the Disk Interface Specification [3].

A complete list of RIO functions is given below.

C_RESET	Perform a hard reset of the disk device and a soft reset of the SCC and the Driver. This function does not issue any commands to the Disk Controller and so no Status Data will be returned.
C_WAKEUP	This function must be requested after a C_RESET and not at any other time. It awakens the disk device and causes it to return its device type identifier.
C_DISK_MODE	Set the disk format to be used for all subsequent disk operations.
C_FILES_OPEN	Instructs the Disk Controller that files are currently open.
C_FILES_CLOSED	Instructs the Disk Controller that all disk files are currently closed.
C_FORMAT	Format a disk and perform surface tests. The function may be used to perform either or both of the format and surface test operations. Formatting configures the track and sector layout on the floppy disk surface, it does not write any data on the disk.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

C_WRITE_SECTORS — Write data to a disk extent. This function can perform a write with verify (the verification is performed by the Disk Controller). The maximum length in bytes that can be written as a disk extent is calculated using the block size i.e.

(65535 DIV block_size) * block_size

The size of a block is dependent upon the disk format in use.

C_READ_SECTORS — Read data from a disk extent. The maximum length in bytes of the disk extent is the same as for the C_WRITE_SECTORS function.

10.3.2 RIO PARAMETERS

The Control Task makes function requests to the Driver using the normal MTOS [1,6,7] RIO mechanism. All RIO functions require an identical format of parameter block as shown below. The RIO parameters include a pointer to this block, which MTOS passes to the Driver through the request IRB (I/O Request Block [6]).

Name	Size	Type	Description
P_DEVICE	1	byte	Device address
P_PARAM	4	pointer	Pointer to ten bytes of Parameter Data
P_STATUS	4	pointer	Pointer to area to receive ten bytes of Status Data
P_DATA	4	pointer	Pointer to S40 software data buffer [2]

TABLE 39. RIO Parameter Block

Not all the fields in the RIO parameter block are used for every function. P_DEVICE is the one mandatory field: the address of the disk device on the SDLC link.

The Parameter Data and Status Data details are contained in the Disk Interface Specification [3].

The information contained within this document is confidential and proprietary to Burroughs Corporation.

The P_DATA field will be used to point to an S40 software data buffer whenever the Control Task passes data to the Driver, and whenever the Driver returns data to the Control Task. In the latter case, the Driver writes the length of the data returned into the data length element of the software data buffer. If the Control Task is passing data to the Driver, the Driver will not corrupt the software data buffer during processing. The Driver raises a system error if the Device requests more data than the data length element of the data buffer indicates is available, or if the Device returns more data than the buffer length element indicates that it will hold. Note that none of the functions both require and return data.

The fields used for each RIO are shown below.

RIO Name	P_DEVICE	P_PARAM	P_STATUS	P_DATA
C_RESET	I	-	-	-
C_WAKEUP	I	R	W	-
C_DISK_MODE	I	R	W	-
C_FILES_OPEN	I	-	W	-
C_FILES_CLOSED	I	-	W	-
C_FORMAT	I	R	W	W
C_WRITE_SECTORS	I	R	W	R
C_READ_SECTORS	I	R	W	W

Key:
- I — Data read directly from the parameter block
- R — Data read from the location indicated by this pointer
- W — Data written to the location indicated by this pointer
- \- — This parameter is not used or checked

TABLE 40. RIO parameter field requirements

The information contained within this document is confidential and proprietary to Burroughs Corporation.

10.3.3 RIO STATUSES

On completion of the RIO function the Driver returns a four byte RIO status to the requesting task. The primary status byte is used to show the completion status of the RIO. The primary status byte values are shown below.

S_SUCCESSFUL The RIO completed without any errors. The ten bytes of Status Data should be examined to determine the command result.

S_FAILURE The RIO has failed. No Status Data is returned if this error is generated. The secondary status byte indicates the source of error as shown below.

- S_HARDWARE Hardware communications error. Eg: Driver received 3 P_NAKs in succession.
- S_PROTOCOL Communications protocol error. Looks like a firmware bug.
- S_RESET The Driver requires a C_RESET command. It enters this state in preference to raising a system error.

S_TIMEOUT An interface timeout has occurred. This indicates that the Driver has failed to receive an expected response from the Controller within a reasonable time. A timeout may be caused by hardware or software failure. No Status Data is returned if this error is generated.

S_SYSTEM_ERROR This error covers RIO function parameter validation and internal software errors. No Status Data is returned if this error is generated. The cause of the error is given in the secondary status byte as shown below.

- S_INTERNAL This error is generated if the software finds itself in a state that is unexpected. It indicates a bug in the Driver.

The C_RESET function always returns S_SUCCESSFUL. All other functions may return any of the primary statuses.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

10.4 CONTROLLING THE DISK USING THE DISK DRIVER

All communications with the disk device must be directed to the correct device by means of the device address. The address is supplied as part of the RIO function parameters.

10.4.1 INITIALISING

The C_RESET command must be the first RIO function invoked. C_RESET takes the Controller from an unknown state to a definite state of "asleep". After the reset the Controller requires to be woken up and configured. The C_WAKEUP function must be the next RIO and allows the Controller to identify itself to the S40. Before the Controller is ready to accept commands it must be configured to operate in a specific operating mode using the C_DISK_MODE function. This takes the Controller to the "ready" state, where it is able to process commands. Whenever the Controller is reset using the C_RESET function it must be reconfigured in the above manner.

10.4.2 DETECTING DISK DEVICE PRESENCE

The presence of a disk device can only be detected when it is possible to communicate with it. If there is no communication on the link then it is impossible to determine whether the disk is

- inoperable because of hardware failure,
- not connected to the communication link,
- not powered on.

The most likely RIO status when the disk device is not functioning is S_TIMEOUT because the disk will not respond to commands.

10.4.3 ISSUING RIO FUNCTIONS

Commands to the disk device are given using RIO functions. Each function, with the exception of C_RESET, issues a command to the disk device.

The parameters and data for each command are supplied using the RIO parameter block. Details of the specific parameters and data needed for each disk command are given in the Disk Interface Specification [3]. When data is supplied for a disk command or data is read from the disk, the Driver verifies that the buffer containing the data is large enough to hold it (see section 10.3.2).

The information contained within this document is confidential and proprietary to Burroughs Corporation.

10.4.4 INTERPRETING RIO COMPLETION STATUS

The completion status of an RIO indicates the success or failure of the function. The S_SUCCESSFUL status indicates that the RIO function has succeeded, but does not indicate the success of the disk command. When S_SUCCESSFUL is returned then the ten bytes of Status Data pointed to by the RIO function parameters should be examined to find the result of the command. Details of the Status Data returned by each command are given in the Disk Interface Specification [3].

Any RIO status that is not S_SUCCESSFUL indicates that the command or its results have not been communicated to the disk device as specified in the communications protocol. No Status Data is returned.

10.4.4.1 INTERFACE TIMEOUT

The Driver employs timeouts to ensure that no function will wait indefinitely for a response from the Controller. The timeouts vary in length with relation to the expected response time. A mechanism is employed to detect and avoid race conditions where a timeout occurs at the same time as the expected response arrives (see section 10.10).

10.4.5 ERROR HANDLING

There are two types of serious errors that must be handled in a specific way to ensure a robust interface. The errors are

- an RIO status of S_SUCCESSFUL where the Status Data indicates a fatal error,

- any RIO status that is not S_SUCCESSFUL.

These error conditions indicate a serious problem with the controlling software, the communications link or the disk device. When an error of this type is reported the Control Task must reset the disk device. Any further action is dependent on the Control Task but it is appropriate to reset and retry the RIO once after such a failure. Where the disk recovers after a reset the error will be transparent to the application program, allowing robustness and elegant recovery from problems.

There are many less serious problems that may be reported in the Status Data. These problems are of no concern to the Driver and only require to be handled by the Control Task.

The information contained within this document is confidential
and proprietary to Burroughs Corporation.

10.5 MAINTENANCE LOGGING

The driver records the following activities in the standard system maintenance log area [2]:

- the number of hard resets to the disk unit,
- the number of retries following a NAK or Frame/CRC error,
- the number of disk reads (the number of DMA transfers of read data from the controller to the driver).
- the number of disk writes (the number of DMA transfers of write data from the driver to the controller).

10.6 HARDWARE LINK LEVEL

The disk device is connected to the S40 by a high speed synchronous data link controlled by the SCC. The SCC is operated in SDLC mode, and only channel B is used. It is fed by an external clock into channel B. Further detail can be obtained from the SCC Technical Manual [8] and the Disk Interface Specification [3].

10.7 MODULES

The Driver is coded in ASM86 assembly language and is contained in one module. A dummy driver is provided to allow operating systems to be built excluding the disk.

10.8 IMPLEMENTATION OVERVIEW

Though the Driver requires separate Service Request Processors for each function that it supports, it is not viable to implement them as separate routines, since they must co-operate with interrupt handling, and the Driver must return control to MTOS whilst waiting for interrupts.

The Disk Controller is implemented as a state machine. The Driver needs to be aware of the operation of the machine as a whole, and of the current state at any particular time. With this in mind, the Driver is also implemented as a state machine, though the states are not in exact correspondence with those of the Controller, since in particular, the Driver and Controller are going to see the transitional operators at different times.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

A state machine consists of:

- A well defined set of states. The current state may be held as a value within a variable.

- A well defined set of transition operators. There is a valid set of these associated with each state, and the machine must be able to cope with apparently illegal operators.

- For each state, and for each transition operator associated with that state, there is a course of action to follow; the end of which defines the new state. Once the state changes, the machine waits for the next transition operator.

In the implementation, the machine recognises events such as a request for service from the Disk Control Task, or an interrupt from the SCC. Upon occurrence of an event, the Driver must first perform whatever servicing/processing is necessary to determine a transition operator. This processing may be dependent upon the current state. Examples of transition operators are a specific function requested by the Control Task, and a status code returned from the Controller.

The Driver has one interrupt handler, servicing both receive and transmit interrupts. A receive interrupt signifies that the transmission of data from the Disk Controller into the DMA RAM buffer has completed. A transmit interrupt signifies that the transmission of data from the DMA RAM buffer to the Disk Controller has completed.

In summary, when the Driver is idle, it is in a fixed state, waiting for an event to occur. Upon occurrence of an event, the Driver must perform whatever processing is required to determine a transition operator. The combination of current state and transition operator defines a course of action to follow, at the end of which the state is changed to reflect what has happened.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

10.9 STATE MACHINE

CURRENT STATE	EVENT	TRANSITION OP	ACTION	NEW STATE
Uninit-ialised	Enter driver init routine Nothing else possible	Enter driver init routine	Program SCC. Place the rest of the hardware into its initial state.	Asleep
Asleep	Service req	C_RESET	Soft reset to SCC and hard reset to DC. End service.	Ready
		Other	End service with fail S_FAILURE, S_RESET.	Asleep
	Other		Ignore.	Asleep
Ready	Service req	C_WAKEUP	Send command to DC. Poll for byte reply. Retry P_NAK twice.	
			- If P_ACK or Frame/ CRC error then prepare to receive byte status report on interrupt.	Exec_cmd
			- If 3 P_NAKs then End service with FAILURE, HARDWARE.	Asleep
			- If other reply then End Service with FAILURE, PROTOCOL	Asleep
			- If timeout then End Service with TIMEOUT.	Ready
		C_DISK_MODE	Send command to DC. Poll for byte reply. Retry P_NAK twice.	
			- If P_ACK or Frame/ CRC error then send 10 bytes of parameters and poll for byte reply.	Xfer_dh

The information contained within this document is confidential and proprietary to Burroughs Corporation.

		- If 3 P_NAKs then End service with FAILURE, HARDWARE.	Asleep
		- If other reply then End Service with FAILURE, PROTOCOL	Asleep
		- If timeout then End Service with TIMEOUT.	Asleep
	C_FORMAT	Send command to DC. Poll for byte reply. Retry P_NAK twice.	
		- If P_ACK or Frame/ CRC error then send 10 bytes of parameters and Poll for byte reply.	Exec_cmd
		- If 3 P_NAKs then End service with FAILURE, HARDWARE.	Asleep
		- If other reply then End Service with FAILURE, PROTOCOL	Asleep
		- If timeout then End Service with TIMEOUT.	Asleep

The information contained within this document is confidential and proprietary to Burroughs Corporation.

	C_FILES_OPEN	Send command to DC. Poll for byte reply. Retry P_NAK twice. - If P_ACK or Frame/ CRC error then prepare to receive byte status report on interrupt. - If 3 P_NAKs then End service with FAILURE, HARDWARE. - If other reply then End Service with FAILURE, PROTOCOL - If timeout then End Service with TIMEOUT.	 Exec_cmd Asleep Asleep Asleep
	C_FILES_CLOSED	Send command to DC. Poll for byte reply. Retry P_NAK twice. - If P_ACK or Frame/ CRC error then prepare to receive byte status report on interrupt. - If 3 P_NAKs then End service with FAILURE, HARDWARE. - If other reply then End Service with FAILURE, PROTOCOL - If timeout then End Service with TIMEOUT.	 Exec_cmd Asleep Asleep Asleep

The information contained within this document is confidential and proprietary to Burroughs Corporation.

	C_RD_SECTORS	Send command to DC. Poll for byte reply. Retry P_NAK twice. - If P_ACK or Frame/ CRC error then send 10 bytes of parameters and poll for byte reply. - If 3 P_NAKs then End service with FAILURE, HARDWARE. - If other reply then End Service with FAILURE, PROTOCOL - If timeout then End Service with TIMEOUT.	Rd_param Asleep Asleep Asleep
	C_WRT_SECTORS	Send command to DC. Poll for byte reply. Retry P_NAK twice. - If P_ACK or Frame/ CRC error then send 10 bytes of parameters and poll for byte reply. - If 3 P_NAKs then End service with FAILURE, HARDWARE. - If other reply then End Service with FAILURE, PROTOCOL - If timeout then End Service with TIMEOUT.	Wrt_param Asleep Asleep Asleep
	C_RESET	Issue soft reset to SCC and DC. End service.	Ready
Other		Nothing.	Asleep

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Exec_cmd	Interrupt	S_IO_COMPLETE	Send P_ACK. Prepare to receive 10 bytes completion data on interrupt.	Compl_io
		Frame/CRC error (1, 2)	Send P_NAK. Prepare to receive byte status report on interrupt.	Exec_cmd
		Frame/CRC error (3)	End service with fail S_FAILURE, S_HARDWARE	Asleep
		Other status	End service with fail S_FAILURE, S_PROTOCOL	Asleep
	Timeout		End service with fail S_TIMEOUT.	Asleep
	Other		End service with fail S_SYSTEM_ERROR, S_INTERNAL.	Asleep
Compl_io	Interrupt	Receive block	Copy from RAM to ret status. Send P_ACK. End Service.	Ready
		Frame/CRC error (1, 2)	Send P_NAK. Prepare to receive 10 bytes of completion data on interrupt.	Compl_io
		Frame/CRC error (3)	End service with fail S_FAILURE, S_HARDWARE	Asleep
		Other	End service with fail S_FAILURE, S_PROTOCOL	Asleep
	Timeout		End service with fail S_TIMEOUT.	Asleep
	Other		End service with fail S_SYSTEM_ERROR, S_INTERNAL.	Asleep

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Xfer_dh	Poll	P_ACK, Frame/CRC err	Prepare to receive byte status report on interrupt.	Exec_cmd
		P_NAK (1, 2)	Send 10 bytes of parameters and poll for byte reply.	Xfer_dh
		P_NAK (3)	End service with fail S_FAILURE, S_HARDWARE	Asleep
		Other	End service with fail S_FAILURE, S_PROTOCOL	Asleep
		Timeout	End service with fail S_TIMEOUT.	Asleep
	Other		End service with fail S_SYSTEM_ERROR, S_INTERNAL.	Asleep
Fmt_ params	Poll	P_ACK, Frame/CRC err	Prepare to receive byte status report on interrupt.	Fmtting
		P_NAK (1, 2)	Send 10 bytes of parameters and poll for byte reply.	Fmt_parm
		P_NAK (3)	End service with fail S_FAILURE, S_HARDWARE	Asleep
		Other	End service with fail S_FAILURE, S_PROTOCOL	Asleep
		Timeout	End service with fail S_TIMEOUT.	Asleep
	Other		End service with fail S_SYSTEM_ERROR, S_INTERNAL.	Asleep

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Fmtting	Interrupt	S_IO_COMPLETE	Send P_ACK. Prepare to receive 10 bytes of completion data on interrupt.	Compl_io
		S_FORMATTING	Send P_ACK. Prepare to receive 512 bytes of bad block data on interrupt.	Fmt_rslt
		S_NOT_DEAD	Send P_ACK. Prepare to receive byte status report on interrupt.	Fmtting
		Frame/CRC error (1, 2)	Send P_NAK. Prepare to receive byte status report on interrupt.	Fmtting
		Frame/CRC error (3)	End service with fail S_FAILURE, S_HARDWARE	Asleep
		Other status	End service with fail S_FAILURE, S_PROTOCOL	Asleep
	Timeout		End service with fail S_TIMEOUT.	Asleep
	Other		End service with fail S_SYSTEM_ERROR, S_INTERNAL.	Asleep

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Fmt_rslt	Interrupt	Receive block	Copy from RAM to data buffer. Send P_ACK. Prepare to receive byte status report on interrupt.	Exec_cmd
		Frame/CRC error (1, 2)	Send P_NAK. Prepare to receive 512 bytes of bad block data on interrupt.	Fmt_rslt
		Frame/CRC error (3)	End service with fail S_FAILURE, S_HARDWARE	Asleep
		Other	End service with fail S_FAILURE, S_PROTOCOL	Asleep
	Timeout		End service with fail S_TIMEOUT.	Asleep
	Other		End service with fail S_SYSTEM_ERROR, S_INTERNAL.	Asleep
Rd_ params	Poll	P_ACK, Frame/CRC err	Prepare to receive byte status report on interrupt.	Reading
		P_NAK (1, 2)	Send 10 bytes of parameters and poll for byte reply.	Rd_params
		P_NAK (3)	End service with fail S_FAILURE, S_HARDWARE	Asleep
		Other	End service with fail S_FAILURE, S_PROTOCOL	Asleep
		Timeout	End service with fail S_TIMEOUT.	Asleep
	Other		End service with fail S_SYSTEM_ERROR, S_INTERNAL.	Asleep

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Reading	Interrupt	S_IO_COMPLETE	Send P_ACK. Prepare to receive 10 bytes of completion data on interrupt.	Compl_io
		S_READ_OK	Send P_ACK. Prepare to receive 2 byte count on interrupt.	Rd_sec_n
		Frame/CRC error (1, 2)	Send P_NAK. Prepare to receive byte status report on interrupt.	Reading
		Frame/CRC error (3)	End service with fail S_FAILURE, S_HARDWARE	Asleep
		Other status	End service with fail S_FAILURE, S_PROTOCOL	Asleep
	Timeout		End service with fail S_TIMEOUT.	Asleep
	Other		End service with fail S_SYSTEM_ERROR, S_INTERNAL.	Asleep

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Rd_sec_cnt	Poll	Receive block	Read count from RAM. Send P_ACK. Prepare to receive data block on interrupt.	R_blk
		Frame/CRC error (1, 2)	Send P_NAK. Prepare to receive 2 byte count on interrupt.	Rd_sec_cn
		Frame/CRC error (3)	End service with fail S_FAILURE, S_HARDWARE	Asleep
		Other	End service with fail S_FAILURE, S_PROTOCOL	Asleep
	Timeout		End service with fail S_TIMEOUT.	Asleep
	Other		End service with fail S_SYSTEM_ERROR, S_INTERNAL.	Asleep

The information contained within this document is confidential and proprietary to Burroughs Corporation.

R_blk	Interrupt	Receive block	Copy from RAM to data buffer. Send P_ACK. Prepare to receive byte status report on interrupt.	Reading
		Frame/CRC error (1, 2)	Send P_NAK. Prepare to receive data block on interrupt.	R_blk
		Frame/CRC error (3)	End service with fail S_FAILURE, S_HARDWARE	Asleep
		Other	End service with fail S_FAILURE, S_PROTOCOL	Asleep
	Timeou		End service with fail S_TIMEOUT.	Asleep
	Other		End service with fail S_SYSTEM_ERROR, S_INTERNAL.	Asleep
Wrt_ params	Poll	P_ACK, Frame/CRC err	Prepare to receive byte status report on interrupt.	Writing
		P_NAK (1, 2)	Send 10 bytes of parameters and poll for byte reply.	Wrt_par
		P_NAK (3)	End service with fail S_FAILURE, S_HARDWARE	Asleep
		Other	End service with fail S_FAILURE, S_PROTOCOL	Asleep
		Timeout	End service with fail S_TIMEOUT.	Asleep
	Other		End service with fail S_SYSTEM_ERROR, S_INTERNAL.	Asleep

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Writing	Interrupt	S_IO_COMPLETE	Send P_ACK. Prepare to receive 10 bytes of completion data on interrupt.	Compl_io
		S_WRT_MORE	Send P_ACK. Prepare to receive 2 byte count on interrupt.	Wrt_secn
		Frame/CRC error (1, 2)	Send P_NAK. Prepare to receive byte status report on interrupt.	Writing
		Frame/CRC error (3)	End service with fail S_FAILURE, S_HARDWARE	Asleep
		Other status	End service with fail S_FAILURE, S_PROTOCOL	Asleep
	Timeout		End service with fail S_TIMEOUT.	Asleep
	Other		End service with fail S_SYSTEM_ERROR, S_INTERNAL.	Asleep
Wrt_ sec_cnt	Interrupt	Receive block	Read count from RAM. Send P_ACK. Send data block.	W_blkA
		Frame/CRC error (1, 2)	Send P_NAK. Prepare to receive 2 byte count on interrupt.	Wrt_secn
		Frame/CRC error (3)	End service with fail S_FAILURE, S_HARDWARE	Asleep
		Other	End service with fail S_FAILURE, S_PROTOCOL	Asleep
	Timeout		End service with fail S_TIMEOUT.	Asleep

The information contained within this document is confidential and proprietary to Burroughs Corporation.

W_blkA	Interrupt		Prepare to receive byte reply on interrupt.	W_blkB
	Timeout		End service with fail S_TIMEOUT.	Asleep
	Other		End service with fail S_SYSTEM_ERROR, S_INTERNAL.	Asleep
W_blkB	Interrupt	P_ACK, Frame/CRC err	Prepare to receive byte status report on interrupt.	Writing
		P_NAK (1, 2)	Send data block. Prepare for TX interrupt.	W_blkA
		P_NAK (3)	End service with fail S_FAILURE, S_HARDWARE	Asleep
		Other	End service with fail S_FAILURE, S_PROTOCOL	Asleep
		Timeout	End service with fail S_TIMEOUT.	Asleep
	Other		End service with fail S_SYSTEM_ERROR, S_INTERNAL.	Asleep

The information contained within this document is confidential and proprietary to Burroughs Corporation.

10.9.1 NOTES ON STATE TABLE

1. Where the event is polling, the Driver never actually pauses its processing. There is a fixed maximum number of times around the loop, and if this limit is reached, the transition operator is timeout. This is different to the timeouts in the event column, where MTOS restarts the Driver after too long a wait for an interrupt [6].

2. On poll and interrupt events, the Controller has sent information to the Driver, which has been deposited in the DMA RAM buffer. This information is either a block of data or a single byte message. The former can be left in the RAM buffer for the state handler to extract. The latter must be read and decoded to determine the transition operator. The Driver must maintain a flag which indicates to the event handler which type of transfer has occurred.

3. Some states are very similar:

   - Exec_cmd and Fmtting
   - Compl_io and Fmt_rslt
   - Xfer_dh and Fmt_params.

   Advantage is taken in the implementation by having the state handling routines of the Driver share code. The shared code is separated from the main state handling routines and placed in a section of the Driver designated as Common Code Subroutines for State Handlers.

4. The Driver must only send information to the Controller whilst the Controller indicates that it is ready to accept information. The Controller signals that it is ready to receive by raising the CTS pin on the Driver SCC, which the Driver can monitor through SCC RR0. In the state machine, it is necessary to poll for CTS high prior to the actions Send command to DC, Send P_ACK, Send P_NAK and Trigger DMA.

5. The Driver must signal to the Controller when it is ready to receive information. It does this by raising the RTS line, to which it has access in SCC WR5. In the state machine, it is necessary to do this prior to waiting on poll or interrupt events.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

10.10 TIMEOUT HANDLING

The Driver waits on external events (Disk Controller and SCC) by polling and by enabling interrupts. To ensure that the Driver does not loop for ever on occurrence of some hardware problem, polls take the form of a finite loop, and if these run to completion, a timeout transition operator results. The Disk hardware should always respond well within 5 milliseconds to any polled event, and the loop counts are set to allow approximately this length of time.

For all events where the time spent waiting for their occurrence is likely to be too long for the polling approach, interrupts are used. MTOS TMBs (Time Monitor Blocks) are employed to guard against waiting on an interrupt for ever. When the Driver prepares for an interrupt, it queues a TMB. If the TMB expires before interrupt servicing takes place, MTOS automatically invokes the designated timeout handler. Receive interrupt events are allowed 30 seconds, and transmit interrupt events are allowed 300 milliseconds. It is necessary to guard against race conditions between the interrupt handler and timeout handler, because both are invoked on interrupt (SCC interrupt and MTOS clock interrupt respectively), but both operate in S-state. A flag in the Driver UCB Status Byte (Z_WAIT_INT) is used for this purpose. It is important to note that this flag is examined and changed in S-state only. The Driver sets the flag when it is waiting for an interrupt. The interrupt and timeout handlers clear the flag when they handle their events. However, before either handler proceeds, it examines the flag. If the flag is clear, it signifies that the other event was handled first, and the beaten handler merely dismisses its event.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

10.11 DATA STRUCTURES

10.11.1 DRIVER STATE VARIABLE

The states making up the state machine are identified by a contiguous set of numbers. The variable Z_UCB_STATE is used to hold the identifier of the current state. The state machine operates as follows:

- Event occurs, initiating an event handler.
- Event handler determines a transition operator.
- The transition operator is passed to the state handler identified by Z_UCB_STATE. This is effected by using the state identifier as an index to a jump table.
- The state handler takes action depending upon the transition operator.
- The state handler determines a new state (modifies Z_UCB_STATE).
- The Driver 'sleeps' until the next event.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

10.11.2 RETRY COUNT VARIABLE

For the sake of robustness, the Driver retries certain failed operations in a manner invisible to the Control Task, though retries are recorded in the system Maintenance Log. The Driver will retry in the following circumstances:

- The Controller sends a NAK in response to a command.
- The Controller sends a NAK in response to the ten bytes of Parameter Data.
- The Controller sends a NAK in response to a sector of data to be written.
- A frame/CRC error is detected when the Controller should be sending a byte status report.
- A frame/CRC error is detected when the Controller should be sending ten bytes of Status Data.
- A frame/CRC error is detected when the Controller should be sending a 2 byte sector size.
- A frame/CRC error is detected when the Controller should be sending a sector of data read.
- A frame/CRC error is detected when the Controller should be sending the bad sector data table.

If the operation fails three times in succession, the Driver reports an error to the Control Task. The number of retries is limited to three per operation using the variable Z_UCB_RETRIES. This variable is set to the maximum number of retries allowed prior to attempting each operation, and is decremented on each fail. The operation is aborted if the variable decrements to zero, otherwise it is retried.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

10.11.3 IRB PARAMETERS

With the exception of C_RESET, every Service Request supplies an identical parameter list, which the Driver can access through the request IRB (section 5.2). The Driver takes a working copy of the parameters in its UCB:

- The byte device address is copied straight into the UCB (Z_UCB_DEVICE).

- The four byte pointer to the Parameter Data is copied into the UCB (Z_UCB_PARAM). The Driver sends these ten bytes of data to the Controller when it acknowledges the command associated with the Service Request.

- The four byte pointer to the area to receive the Status Data is copied into the UCB (Z_UCB_STATUS). The Controller sends this data when it has carried out the command.

- The Software Data Buffer (not the pointer to it) is copied into the UCB. This consists of:

    - A word buffer length (Z_UCB_BLEN). This defines the capacity of the data area, and the Driver ensures that this is not exceeded when receiving bad block data for the C_FORMAT request or data for the C_READ_SECTORS request.

    - A word data length (Z_UCB_DLEN). This defines the quantity of data to be sent to the Controller or received from the Controller. The Driver uses this as a downcount variable when sending blocks of data for the C_WRITE_SECTORS request, ensuring that it never goes negative. The variable is used as a count of bytes received during the C_READ_SECTORS request. Once the C_READ_SECTORS request is complete, the Driver copies the total count to the data length field in the Software Data Buffer to which the IRB parameter block points.

    - A four byte pointer to the data area (Z_UCB_DPTR). The Driver updates this copy of the pointer as it reads from or writes to the data area.

The four byte pointer variables are all subdivided into pairs of words (suffixed with _H and _L) within the UCB, since the 8086/88 processor can only transfer data word at a time.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

10.11.4 DISK COMMAND VARIABLE

At the start of a service request, the Driver writes the corresponding Controller command to variable Z_UCB_CMD. It is convenient to store the command, since it will have to be sent more than once if the Controller NAKs the command the first time around.

10.11.5 TRANSFER SIZE VARIABLES

Variable Z_UCB_RX_SIZE is used to hold the number of bytes that the Driver expects to receive from the Controller in an individual DMA transfer. Utility routines to read data from the DMA buffer examine this variable to determine the high address of the buffer from which they should start reading.

Variable Z_UCB_TX_SIZE is used to hold the number of bytes to be sent from the Driver to the Controller in a single DMA transfer. Utility routines to send the data examine this variable to determine the quantity of data to send, and hence the high address in the DMA buffer to which they should start writing. It will be necessary to refer to this value more than once if the Controller NAKs the block the first time around.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11. DIRECT ACCESS METHOD

11.1 INTRODUCTION

The Direct Access Method (DAM) provides application programs with a buffered random access method for files with fixed length records. Records are referenced by record number. The DAM functions provided are :

- Open a file
- Close a file
- Read a record
- Write a record
- Query record status
- Query last record
- Delete a record
- Truncate a file

In the above functions DAM does calculations based on the record number to find the relevant sector(s) of the file on the disk. The most recently read sectors are buffered to increase performance of records sequentially accessed.

All DAM functions receive a parameter block with the call from the application, set up one or more parameter blocks and call the low level disk functions. In the remainder of this DAM software specification the calling of a low level disk function will be referred to as calling disk.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.2 GENERAL NOTES

11.2.1 DAM WORK AREA

The DAM functions require certain information on every open file in order to manipulate it. This information is held in a unique DAM work area supplied by the application for each DAM file which is opened. The address of this work area is passed whenever any DAM function is called.

The information required is as follows:

- a file index (16 bit unsigned integer) : a number assigned, by the low level disk access routines, to an open file to identify it.
- a description of a disk buffer for the exclusive use of DAM code (not applications buffer).
    - buffer length (16 bit unsigned integer) : the length of the disk buffer supplied by the application.
    - data length (16 bit unsigned integer) : a number set up by disk specifying the number of bytes read/written.
    - buffer area (pointer) : the address of an area where sectors can be read/written.
- access mode (16 bit unsigned integer) : type of file access required read only or read/write.
- DAM work area signature (16 bit unsigned integer) : a mark used to identify an area of memory as a DAM work area.
- work area in use flag (16 bit unsigned integer) : a flag used to show whether the DAM work area is currently in use or not.
- current sector in disk buffer (32 bit unsigned integer) : the number of the sector currently occupying the first page of the DAM disk buffer.
- file record length (16 bit unsigned integer) : the length, in bytes, of records in the opened file.
- file size (32 bit unsigned integer) : the length, in bytes, of the file.
- work area for disk access routines : see disk specification.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.2.2 ERROR HANDLING

All errors received from the low level disk functions are passed back to the application without further processing of the command with the exception of DK_FL_NOT_FOUND in the open DAM file function. The handling of this error will be described in the Open Dam File section.

Error Classes

- 1 : Fatal - Cannot proceed.
- 2 : Recoverable - Issue sequence of commands.
- 3 : Retryable - Re-issue command.
- 4 : Warning - Operation complete : check warning for unexpected result.

Errors

Errors returned by DAM functions are described below. The error is followed by an error class and a description of why the error was caused. The errors are divided into functions and then sub_divided into causes.

11.2.2.1 OPEN FILE

General

- DK_FL_NOT_FOUND (3) only if access mode is read and file doesn't exist.
- DM_DWA_IN_USE (3) DAM work area supplied is linked to an open file.
- DM_DISK_FULL (2) an attempt was made to create a file on a disk with no space.

Parameter Block

- DM_BAD_BUFFER_SIZE (3) buffer size not a multiple of 512 and/or buffer size too small.
- DM_BAD_REC_SIZE (3) record size > 63992 bytes.
- DM_INV_ACC_MODE (3) access mode is not "mr" or "mm".

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Parameter Block (after comparison to file header block)

- DM_REC_MISMATCH (3) requested record size <> file record size.

File Header Block

- DM_NOT_A_DAM_FL (3) offset<>0, header size<>50, bcheck invalid, signature<>"am", file ID<>4 or size of access method dependant information<>0.

- DM_HEADER_BAD_CHECK (3) bad checksum.

- DM_NOT_FIX_LEN_REC (3) min record size <> max record size.

Other

DM_FL_CREATED (4) the specified file did not exist and was created

Other status codes may be returned from the following low level disk access functions:

- DK_OPEN_FN

- DK_READ_FN

- DK_GETST_FN

- DK_CREATE_FN

- DK_WRITE_FN

- DK_SETST_FN

11.2.2.2 CLOSE FILE

Parameter Block

- DM_INV_DWA (3) invalid DAM work area passed as parameter.

Other

Other status codes may be returned from the following low level disk access functions:

- DK_CLOSE_FN

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.2.2.3 READ RECORD

Parameter Block

- DM_INV_DWA (3) invalid DAM work area passed as parameter.
- DM_INV_REC_NO (3) record does not physically exist within file.
- DM_BAD_BUFFER_SIZE (3) the data length to be written does not equal the record size.

File Format

- DM_REC_DELETED (4) record deleted.
- DM_REC_NOT_EXIST (4) record does not logically exist.
- DM_MALFORMED_REC (4) record size invalid, bcheck invalid or offset wrong.

Other

Other status codes may be returned from the following low level disk access functions:

- DK_READ_FN

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.2.2.4 WRITE RECORD

Parameter Block

- DM_INV_DWA (3) invalid DAM work area passed as parameter.
- DM_FL_READ_ONLY (2) request to write when in read mode.
- DM_DISK_FULL (2) an attempt was made to extend a file on a disk with no space.
- DM_BAD_BUFFER_SIZE (3) the buffer is not big enough to hold the DAM record requested.

File Format

- DM_MALFORMED_REC (4) record size invalid, bcheck invalid or offset wrong.

Other

Other status codes may be returned from the following low level disk access functions:

DK_GETST_FN
DK_CHNGFLEN_FN
DK_SETST_FN
DK_READ_FN
DK_WRITE_FN

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.2.2.5 QUERY RECORD STATUS

Parameter Block

- DM_INV_DWA (3) invalid DAM work area passed as parameter.

File Format

- DM_MALFORMED_REC (4) record size invalid, bcheck invalid or offset wrong.

Other

Other status codes may be returned from the following low level disk access functions:

- DK_READ_FN

11.2.2.6 QUERY LAST RECORD

Parameter Block

- DM_INV_DWA (3) invalid DAM work area passed as parameter.

File Format

- DM_MALFORMED_REC (4) record size invalid, bcheck invalid or offset wrong.

Other

Other status codes may be returned from the following low level disk access functions:

- DK_READ_FN

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.2.2.7 DELETE RECORD

Parameter Block

- DM_INV_DWA (3) invalid DAM work area passed as parameter.
- DM_FL_READ_ONLY (2) request to delete when in read mode.
- DM_INV_REC_NO (3) record does not physically exist within file.

File Format

- DM_REC_DELETED (4) record already deleted.
- DM_REC_NOT_EXIST (4) record does not logically exist.
- DM_MALFORMED_REC (4) record size invalid, bcheck invalid or offset wrong.

Other

Other status codes may be returned from the following low level disk access functions:

- DK_READ_FN
- DK_WRITE_FN

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.2.2.8 TRUNCATE FILE

Parameter Block

- DM_INV_DWA (3) invalid DAM work area passed as parameter.
- DM_FL_READ_ONLY (2) request to truncate when in read mode.
- DM_INV_REC_NO (3) record does not physically exist within file.

File Format

- DM_MALFORMED_REC (4) record size invalid, bcheck invalid or offset wrong.

Other

Other status codes may be returned from the following low level disk access functions:

- DK_GETST_FN
- DK_CHNGFLEN_FN
- DK_SETST_FN

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.3 OPEN FILE (DAM_OPEN)

There are no functional differences between this command and the BTOS function OpenDaFile. When a file is opened in modify mode then write through is assumed and cannot be altered.

The parameter block received looks like:

- filename (pointer) : the address of the name of the file to be opened.
- password (pointer) : the address of the password for the file to be opened.
- DAM work area (pointer) : the address of a DAM work area.
- buffer area (pointer) : the address of an area where sectors can be read/written.
- buffer length (16 bit unsigned integer) : the length of the buffer area in bytes supplied by the application.
- file record length (16 bit unsigned integer) : the length, in bytes, of records in the file to be opened.
- access mode (16 bit unsigned integer) : type of file access required read only or read/write.

The parameter block is checked as follows:

- Record Size

  The record size must be in the range 0 to 63992 inclusive if not the error DM_BAD_REC_SIZE is returned.

- DAM Work Area

  The DAM work area is checked to see if it is already assigned to an open DAM file. If it is the error DM_DWA_IN_USE is returned.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

- Disk Buffer

  The disk buffer is checked for word alignment by the disk access routines.

  The size of the disk buffer is checked as follows:

  - The buffer size must be a multiple of 512 bytes.

  - The buffer size must be greater than or equal to the record size plus 519 unless 512 is a multiple of the record size plus 8 in which case the buffer size must be a minimum 512 bytes or

  - the record size plus 8 is a multiple of 512 in which case the buffer size must be a minimum of the record size plus 8.

  If any of these conditions are not met then the error DM_BAD_BUFFER_SIZE is returned

- Access Mode

  The access mode must be either mode read ("mr") or mode modify ("mm"), otherwise the error DM_INV_ACC_MODE is returned, where the "m" is the most significant BYTE and "m" or "r" is the least significant byte of the word.

If the parameter block is valid then the necessary information is stored in the DAM work area. The parameter block for DK_OPEN_FN is then set up and an attempt is made to open the specified file.

If the error DK_FL_NOT_FOUND is returned from DK_OPEN_FN and the access mode is mode write then a file is created. If the file is opened successfully then it is validated.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Validating a DAM file

The DK_GETST_FN parameters are set up to read the file length and disk is called. If the file size is less than 512 then the error DM_NOT_A_DAM_FL is returned. The parameter block for DK_READ_FN is set up to read the file header record and disk called. The fields of the file header record, as defined in the functional specification are then validated as follows:

- offset = 0
- file header record size=50
- bcheck>15
- signature="am"
- file ID=4
- access method dependent information=0
- dbcheck=bcheck

If any of these conditions are not met then the error DM_NOT_A_DAM_FL is returned. If the checksum is wrong then the error DM_HEADER_BAD_CHECK is given. If minimum record size is not equal to maximum record size then the error DM_NOT_FIX_LEN_REC is returned. If file record size is not equal to requested record size then the error DM_REC_MISMATCH is given. If the file header is valid and the access mode is mode modify ("mm") then the file header block bcheck is updated and the parameter block for DK_WRITE_FN is set up to write the file header record to disk and disk is called.

Creating a DAM file

The parameter block for DK_CREATE_FN is set up and disk called. The file header record, as described in the functional specification, is then created in the disk buffer. The parameters for DK_OPEN_FN are set up again and the file is opened by calling disk. The parameter block for DK_WRITE_FN is then set up to write the file header record to disk. The parameters for DK_SETST_FN are set up to set file length to 512 and disk is called.

If the request has been successful then the primary status IO_SUCCESSFUL_COMPLETION is returned unless the file was created in which case the error DM_FL_CREATED is returned.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.4 CLOSE FILE (DAM_CLOSE)

There are no functional differences between this command and the BTOS function CloseDaFile.

The parameter block consists of:

- DAM work area (pointer) : the address of a DAM work area assigned to the file to be closed.

The parameter block is checked as follows :

- DAM work area

  The DAM work area is checked to see if it is assigned to an open file. If the work area is invalid then the error message DM_INV_DWA is returned.

If the work area is valid then the parameters for DK_CLOSE_FN are set up and disk called.

If the request has been successful then the primary status IO_SUCCESSFUL_COMPLETION will be returned.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.5 READ RECORD (DAM_READ)

There are no functional differences between this command and the BTOS function ReadDaRecord.

The parameter block consists of:

- DAM work area (pointer) : the address of a DAM work area assigned to the file to be read from.

- Record data area (pointer) : the address of the description of a data buffer.

- Buffer length (16 bit unsigned integer) : the length of the data buffer supplied by the application.

- Data length (16 bit unsigned integer) : the number of bytes returned into buffer area, this will be set by DAM.

- Buffer area (pointer) : the address of an area to where the record is to be returned.

- Record number (32 bit unsigned integer) : the number of the record to be read.

The parameter block is checked as follows:

- DAM work area

  The DAM work area is checked to see if it is assigned to an open file. If the work area is invalid then the error message DM_INV_DWA is returned.

- Record number

  The offset of the record in the file is calculated and if the record does not physically exist within the file then the error DM_INV_REC_NO is returned.

- Buffer length

  The buffer length is checked to see if it is big enough to hold the record, that is buffer length >= record size. If it is not the error DM_BAD_BUFFER_SIZE is returned.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

The buffer bounds are then checked to see if the buffer contains the record. If it doesn't then the parameter block for DK_READ_FN is set up to read in the relevant sector(s) and disk called. The record is then checked as follows:

The offset field of the record header is checked to see if it contains the correct offset, the record size field must equal the record length plus 8, bcheck must equal dbcheck. If any of these conditions are not met then the error DM_MALFORMED_REC is returned. If bcheck is 1 then the error DM_REC_DELETED is returned. If bcheck is 0 then the error DM_REC_NOT_EXIST is returned. The record is then transferred from the disk buffer to the data buffer, even if the errors DM_MALFORMED_REC, DM_REC_DELETED or DM_REC_NOT_EXIST are returned.

If the request has been successful then the primary status IO_SUCCESSFUL_COMPLETION will be returned.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.6 WRITE RECORD (DAM_WRITE)

There are no functional differences between this command and the BTOS function WriteDaRecord.

The parameter block consists of:

- DAM work area (pointer) : the address of a DAM work area assigned to the file to be written to.
- Record data area (pointer) : the address of the description of a data buffer.
    - buffer length (16 bit unsigned integer) : the length of the data supplied by the application.
    - data length (16 bit unsigned integer) : the number of bytes to be written from the buffer.
    - buffer area (pointer) : the address of an area from where the record has to be written.
- Record number (32 bit unsigned integer) : the number of the record to be written.

The parameter block is checked as follows :

- DAM work area

  The DAM work area is checked to see if it is assigned to an open file. If the work area is invalid then the error message DM_INV_DWA is returned.

- Data length

  The data length is checked to see if it is the correct size to fill a record, that is data length = record size. If it is not the error DM_BAD_BUFFER_SIZE is returned.

The access mode is then checked and if access is read only then the error DM_FL_READ_ONLY is returned.

The record number is then checked to see if it physically exists within the file. If it doesn't exist within the file, the file is filled out to the required record number and all records added are set to be "non-existent". The following steps are required to do this:

- Set up the parameter block for DK_GETST_FN to read the number of sectors allocated to the file and call disk.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

- If there are not enough allocated sectors to hold the record then the parameter block for DK_CHNGFLEN_FN is set up and disk called to request enough sectors to hold the record. If the number of sectors required is greater than the number supplied then the error DM_DISK_FULL is returned.

- The parameter block for DK_SETST_FN is then set up for set file length and disk called to set the length of the file to the end of the new record.

- The following process is carried out for each new record to be created:

    - If the sector(s) that the record exists on are not resident in the disk buffer then set up the parameter block for DK_READ_FN to read in required sector(s) and call disk.

    - The record header and trailer are set up to show that the record does not logically exist.

    - If this is the last record to be created or the end of the next record to be created is outside the buffer area (i.e. the record will be "read in" and the data in the disk buffer will be overwritten) then the parameter block for DK_WRITE_FN is set up to write the required sector(s) to disk and disk called.

The buffer bounds are then checked to see if the buffer contains the record to be written. If it doesn't then the parameter block for DK_READ_FN is set up to read in the relevant sector(s) and disk called. The record is then checked as follows:

The offset field of the record header is checked to see if it contains the correct offset, the record size field must equal the record length plus 8 and bcheck must equal dbcheck. If any of these conditions are not met then the error DM_MALFORMED_REC is returned.

If the record is valid then the bcheck is incremented or set to Hex 10 if it is less than Hex 10 and the data is transferred from the data buffer to the disk buffer. The parameter block for DK_WRITE_FN is then set up to write the required number of sectors for the record and disk is called. Write through mode is assumed.

If the request has been successful then the primary status IO_SUCCESSFUL_COMPLETION will be returned.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.7 QUERY RECORD STATUS (DAM_Q_REC_STATUS)

There are no functional differences between this command and the BTOS function QueryDaRecordStatus.

The parameter block consists of:

- DAM work area (pointer) : the address of a DAM work area assigned to the file from which the record is to be read.

- Record number (32 bit unsigned integer) : the number of the record whose status is required.

- Record status (pointer) : the address of an 8 bit unsigned integer into which the record status is to be returned.

The parameter block is checked as follows:

- DAM work area

  The DAM work area is checked to see if it is assigned to an open file. If the work area is invalid then the error message DM_INV_DWA is returned.

- Record number

  The offset of the record in the file is worked out and if the record does not physically exist within the file then the record status DM_INV_REC_NO is returned.

The buffer bounds are then checked to see if the buffer contains the record. If it doesn't then the parameter block for DK_READ_FN is set up to read in the relevant sector(s) and disk called.

The offset field of the record header is checked to see if it contains the correct offset, the record size field must equal the record length plus 8, bcheck must equal dbcheck. If any of these conditions are not met then the error DM_MALFORMED_REC is returned. If bcheck is 1 then the record status DM_REC_DELETED is returned. If bcheck is 0 then the record status DM_REC_NOT_EXIST is returned.

If the request has been successful then the primary status IO_SUCCESSFUL_COMPLETION will be returned.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.8 QUERY LAST RECORD (DAM_Q_LAST_REC)

There are no functional differences between this command and the BTOS function QueryDaLastRecord.

The parameter block consists of:

- DAM work area (pointer) : the address of a DAM work area assigned to the file from which the last record is to be found.

- Last record number (pointer) : the address of a 32 bit unsigned integer into which the number of the last record is to be written.

The parameter block is checked as follows:

- DAM work area

  The DAM work area is checked to see if it is assigned to an open file. If the work area is invalid then the error message DM_INV_DWA is returned.

The following procedure is repeated for each record, from the last until the first or until a valid record is encountered:

- If the record is not in the disk buffer then set up the parameter block for DK_READ_FN and read in the required number of sectors.

- Check the validity of the current record. Record is valid if bcheck>16, offset = offset in file of record and record size = record size in DAM work area.

- The offset field of the record header is checked to see if it contains the correct offset, the record size field must equal the record length plus 8 and bcheck must equal dbcheck. If any of these conditions are not met then the error DM_MALFORMED_REC is returned.

If a valid record is found then store the record number in the application double word pointed to by Last record number otherwise set the double word to zero.

If the request has been successful then the primary status IO_SUCCESSFUL_COMPLETION will be returned.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.9 DELETE RECORD (DAM_DELETE)

There are no functional differences between this command and the BTOS function DeleteDaRecord.

The parameter block consists of:

- DAM work area (pointer) : the address of a DAM work area assigned to the file from which the record is to be deleted.

- Record number (32 bit unsigned integer) : the number of the record to be deleted.

The parameter block is checked as follows :

- DAM work area

  The DAM work area is checked to see if it is assigned to an open file. If the work area is invalid then the error message DM_INV_DWA is returned.

- Record number

  The offset of the record in the file is worked out and if the record does not physically exist within the file then the error DM_INV_REC_NO is returned.

The access mode is then checked and if access is read only then the error DM_FL_READ_ONLY is returned.

The buffer bounds are then checked to see if the buffer contains the record. If it doesn't then the parameter block for DK_READ_FN is set up to read in the relevant sector(s) and disk called. The record is then checked as follows:

The offset field of the record header is checked to see if it contains the correct offset, the record size field must equal the record length plus 8 and bcheck must equal dbcheck. If any of these conditions are not met then the error DM_MALFORMED_REC is returned. If bcheck is 1 then the error DM_REC_DELETED is returned. If bcheck is 0 then the error DM_REC_NOT_EXIST is returned.

If the record is valid then the bcheck is set to 1. The parameter block for DK_WRITE_FN is then set up to write the required number of sectors for the record and disk is called.

If the request has been successful then the primary status IO_SUCCESSFUL_COMPLETION will be returned.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.10 TRUNCATE FILE (DAM_TRUNCATE)

There are no functional differences between this command and the BTOS function TruncateDaFile.

The parameter block consists of :

- DAM work area (pointer) : the address of a DAM work area assigned to the file to be truncated.

- Record number (32 bit unsigned integer) : the number of the record after which the file is to be truncated.

The parameter block is checked as follows :

- DAM work area

  The DAM work area is checked to see if it is assigned to an open file. If the work area is invalid then the error message DM_INV_DWA is returned.

- Record number

  The offset of the record in the file is calculated and if the record does not physically exist within the file then the error DM_INV_REC_NO is returned.

The access mode is then checked and if access is read only then the error DM_FL_READ_ONLY is returned.

If there were no errors then the following procedure truncates the file:

Set up the parameter block for DK_GETST_FN to read the number of sectors allocated to the file and call disk.

- The parameter block for DK_SETST_FN is then set up for set file length and disk called to set the length of the file to the end of the required record.

- If there are too many allocated sectors (i.e. over thirty "spare" allocated sectors) then the parameter block for DK_CHNGFLEN_FN is set up and disk called to de-allocate the extra sectors.

If the request has been successful then the primary status IO_SUCCESSFUL_COMPLETION will be returned.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.11 TESTING

The following tests are development tests, not tests by independent groups i.e. DQA and PA&S.

The tests will be described, for each DAM function, as follows:

DAM function name

- Tests with valid input parameters. These are included to check behaviour under normal operating conditions.
- Boundary tests. These are included to check the reliabilty of the software.
- Efficiency tests. These are included to check the performance of the software.
- Controlled Error tests. These are included to test the design of the software and flow of control within the software.

11.11.1 OPEN FILE

Tests with valid parameters:

- Buffer Size : supply a buffer which is a multiple of 512 and is big enough for record size.
- Access Mode : pass an access mode which is either "mm" or "mr".
- DAM Work Area : supply the address of a DAM work area which is not already assigned to another open DAM file.
- Record Size : request a record size <= 63992 (max. allowable record size) and record size = DAM file record size.
- File Size : test with file size = 512 (record header exists but no records), file size > 512 (file contains records) and a non-existant file when access mode = "mr".
- File Header Block : use files with offset = 0, file header record size = 80, bcheck >= 16, signature = "am", file id = 4, file dependent information size = 0, dbcheck = bcheck, checksum = correct checksum, file maximum record size = file minimum record size.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Boundary tests:

- The boundary tests for this function will be devised after coding.

Efficiency tests:

- The Efficiency tests will be worked out along with the boundary tests and will include performance related data.

Controlled Error tests:

- Buffer Size : supply a buffer which is not a multiple of 512 or is too small for record size.

- Access Mode : pass an access mode which is neither "mm" nor "mr".

- DAM Work Area : supply the address of a DAM work area which is already assigned to another open DAM file.

- Record Size : request a record size > 63992 (max. allowable record size) and record size <> known DAM file record size.

- File Size : Test with file size = 0 (file empty), file size < 512 (no file header record).

- File Header Block : use files with offset <> 0, file header record size <> 80, bcheck < 16, signature <> "am", file id <> 4, file dependent information size <> 0, dbcheck <> bcheck, checksum <> correct checksum, file maximum record size <> file minimum record size.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.11.2 CLOSE FILE

Tests with valid parameters:

- DAM work area : pass a DAM work area that has been assigned to an open DAM file.

Boundary tests:

- The boundary tests for this function will be devised after coding.

Efficiency tests:

- The Efficiency tests will be worked out along with the boundary tests and will include performance related data.

Controlled Error tests:

- DAM work area : pass a DAM work area that has not been assigned to an open DAM file.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.11.3 READ RECORD

Tests with valid parameters:

- DAM work area : a DAM work area that is currently assigned to an open DAM file must be passed.

- Record number : the number of a valid record must be passed.

Boundary tests:

- The boundary tests for this function will be devised after coding.

Efficiency tests:

- The Efficiency tests will be worked out along with the boundary tests and will include performance related data.

Controlled Error tests:

- DAM work area : a DAM work area that is not currently assigned to an open DAM file must be passed.

- Record number : a record number for a record that does not physically exist within the file, the number of a deleted record, the number of a non-existant record and the number of a malformed record (a record with the wrong offset value, wrong record size or bcheck <> dbcheck) should all be tried.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.11.4 WRITE RECORD

Tests with valid parameters:

- DAM work area : a DAM work area that is currently assigned to an open DAM file whose access mode = "mm" must be passed.

- Record number : a valid record number, a record number for a record that does not physically exist within the file, the number of a deleted record and the number of a non-existant record should all be tried.

Boundary tests:

- The boundary tests for this function will be devised after coding.

Efficiency tests:

- The Efficiency tests will be worked out along with the boundary tests and will include performance related data.

Controlled Error tests:

- DAM work area : a DAM work area that is not currently assigned to an open DAM file, and a DAM work area that is assigned to a read only (access mode = "mr") file must be passed.

- Record number : a record number for a malformed record (a record with the wrong offset value, wrong record size or bcheck <> dbcheck) must be tried.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.11.5 QUERY RECORD STATUS

Tests with valid parameters:

- DAM work area : a DAM work area that is not currently assigned to an open DAM file must be passed.

- Record number : a valid record number, a record number for a record that does not physically exist within the file, the number of a deleted record and the number of a non-existant record should all be tried to check that the correct status is returned.

  Boundary tests:

  - The boundary tests for this function will be devised after coding.

  Efficiency tests:

  - The Efficiency tests will be worked out along with the boundary tests and will include performance related data.

  Controlled Error tests:

  - DAM work area : a DAM work area that is not currently assigned to an open DAM file must be passed.

  - Record number : the number of a malformed record (a record with the wrong offset value, wrong record size or bcheck <> dbcheck) should be tried to check that the correct error is returned.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.11.6 QUERY LAST RECORD

Tests with valid parameters:

- DAM work area : a DAM work area that is currently assigned to an open DAM file must be passed.

- File contents : a file with a valid record as last record in file, a file which only contains deleted or non-existant records, a file with no records and a file with a valid record within the file but not in last record position should all be tried to check that the correct record number is returned.

Boundary tests:

- The boundary tests for this function will be devised after coding.

Efficiency tests:

- The Efficiency tests will be worked out along with the boundary tests and will include performance related data.

Controlled Error tests:

- DAM work area : a DAM work area that is not currently assigned to an open DAM file must be passed.

- File contents : a file with a malformed record (a record with the wrong offset value, wrong record size or bcheck <> dbcheck) as 'last record' and a file with a malformed record within the file but not in 'last record' position but with no valid records after it should both be tried to check that the correct record number or error is returned.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.11.7 DELETE RECORD

Tests with valid parameters:

- DAM work area : a DAM work area that is currently assigned to an open DAM file whose access mode = "mm" must be passed.

- Record number : the number of a valid record must be passed.

Boundary tests:

- The boundary tests for this function will be devised after coding.

Efficiency tests:

- The Efficiency tests will be worked out along with the boundary tests and will include performance related data.

Controlled Error tests:

- DAM work area : a DAM work area that is not currently assigned to an open DAM file, and a DAM work area that is assigned to a read only (access mode = "mr") file must be passed.

- Record number : a record number for a record that does not physically exist within the file, the number of a deleted record, the number of a non-existant record and the number of a malformed record (a record with the wrong offset value, wrong record size or bcheck <> dbcheck) should all be tried.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

11.11.8 TRUNCATE FILE

Tests with valid parameters:

- DAM work area : a DAM work area that is currently assigned to an open DAM file whose access mode = "mm" must be passed.

- Record number : a record number for a record that exists within the file must be tried and a record number of 0.

Boundary tests:

- The boundary tests for this function will be devised after coding.

Efficiency tests:

- The Efficiency tests will be worked out along with the boundary tests and will include performance related data.

Controlled Error tests:

- DAM work area : a DAM work area that is not currently assigned to an open DAM file, and a DAM work area that is assigned to a read only (access mode = "mr") file must be passed.

- Record number : a record number for a record that does not physically exist within the file must be tried.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

APPENDIX A. RETURN STATUS CODES

The following return status values are returned in the secondary status byte, as a result of an error:

DK_ABM_INVALID - This error can arise for two reasons.

1. The number of free sectors marked in the allocation bit map does not correspond to the free pages count in the Vhb.

2. The bad sectors marked in the bad block table (BadBlk.Sys) are not all marked as allocated in the allocation bit map.

DK_ACCESS_DENIED - This error can arise for two reasons

1. The access authority provided to a function via the password was not sufficient to override the protection level of a file, directory or volume.

2. The open mode of a file disallowed the requested function.

DK_ACC_CODE_INVALID - The access code provided to a DK_CRTDIR_FN or DK_SETDS_FN call was not a valid access code for a directory.

DK_BAD_NAME_SPEC - The volume, directory, file name or password provided to a function was not in a valid format.

DK_DEV_NOT_PRESENT - This error is returned to the application if a working device is not connected to the S40. Possible causes are disk not powered on or not connected.

DK_DIR_ALREADY_EXISTS - A create directory request failed because the named directory was already present on the disk.

DK_DIR_FULL - An attempt to create a file failed because there was not enough room in the selected directory.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

DK_DIR_IS_NOT_EMPTY - An attempt was made to delete a directory which still contained files.

DK_DIR_NOT_FOUND - The request failed because the named directory does not exist.

DK_DISK_FULL - No space left on disk to allocate.

DK_EOM_ENCOUNTERED - An attempt was made to access a physical disk address, which was non-existant on the device addressed, or a logical file address which is beyond the end of the physical file.

DK_FHB_CHAIN_BROKEN - An attempt to allocate a file header block failed due to an inconsistency within the free file headers chain. If this error occurs the disk in question should be copied, file by file, and then re-initialised.

DK_FHB_INVALID - An inconsistency was detected in the file header block read in during the request.

DK_FINDEX_INVALID - The request failed because the file index supplied to identify the file was not in use, or non-existant.

DK_FL_ALREADY_EXISTS - A create file request failed because the named file was already present on the disk.

DK_FL_IN_USE - Cannot open file as file is already opened in an incompatible mode.

DK_FL_NOT_FOUND - The request failed because the named file does not exist.

DK_FN_NOT_IMPLEMENTED - The function code used in the request to the disk subsystem is not available in the disk subsystem in use. In a DK_SETDS_FN, DK_GETDS_FN, DK_SETST_FN or DK_GETST_FN function call this error could also indicate that the status code supplied was invalid.

DK_FRAGMENTED - From a DK_CHNGFLEN_FN call, a smaller area than requested has been allocated, probably due to fragmentation of the disk.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

DK_HARD_ERROR	- The disk driver reported a disk hardware failure, or communications protocol error.
DK_INV_VOL_NAME	- Volume name invalid in current system.
DK_IO_ERROR	- An IO request resulted in a hard I/O error being reported.
DK_MFD_FULL	- A DK_CRTDIR_FN function call failed because there was not enough space in the master file directoy.
DK_NO_FHB_FREE	- An attempt to create a new file, or extend an existing file failed because all the file header blocks are allocated.
DK_NO_SLOTS_FREE	- An attempt to open a file failed because all of the file indices were allocated.
DK_NON_ALIGNED_BUFF	- An i/o request was made, using a buffer which was not word aligned.
DK_NOT_READY	- A disk device returned a 'not ready' status in response to an i/o request. This could indicate that the door has been opened, or there is a hardware failure.
DK_PROTECTED	- An attempt was made to write on a disk which was write protected.
DK_STAM_FH_INVALID	- A call to the DK_GETSFH_FN function failed because the STandard Access Method (STAM) file header was not valid or the file was not a STAM file.
DK_VHB_INVALID	- Validation of the Initial Vhb, or Working Vhb failed. If the tertiary status byte is 00H then the Initial Vhb is invalid. If the tertiary status byte is 01H then the Working Vhb is invalid. If the initial VHB is corrupt then the disk requires reinitialisation. However, if the working VHB is corrupt, some data may be recoverable by use of the Reclaim Utility, see S40 System Software Utilities Functional Specification.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

DK_VOL_NOT_MOUNTED - Indicates that the request failed because the volume name supplied does not refer to the volume currently in the drive.

DK_WRONG_DISK - On recovering the disk drive after an error, the disk subsystem detected inconsistencies between the disk which was in the drive before the error and the disk which is in the drive after recovery.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Primary and Secondary Status Codes Table

IO_WARNING	
	DK_EOM_ENCOUNTERED
IO_RETRYABLE	
	DK_ACCESS_DENIED
	DK_ACC_CODE_INVALID
	DK_DEV_NOT_PRESENT
	DK_INV_VOL_NAME
	DK_IO_ERROR
	DK_NOT_READY
	DK_PROTECTED
	DK_VOL_NOT_MOUNTED
	DK_WRONG_DISK
IO_FATAL	
	DK_ABM_INVALID
	DK_DISK_FULL
	DK_FHB_CHAIN BROKEN
	DK_FHB_INVALID
	DK_FINDEX_INVALID
	DK_FN_NOT_IMPLEMENTED
	DK_NON_ALIGNED_BUFFER
	DK_STAM_FH_INVALID
	DK_VHB_INVALID
IO_RECOVERABLE	
	DK_BAD_NAME_SPEC
	DK_DIR_ALREADY_EXISTS
	DK_DIR_FULL
	DK_DIR_IS_NOT_EMPTY
	DK_DIR_NOT_FOUND
	DK_FL_ALREADY_EXISTS
	DK_FL_IN_USE
	DK_FL_NOT_FOUND
	DK_FRAGMENTED
	DK_HARD_ERROR
	DK_MFD_FULL
	DK_NO_FHB_FREE
	DK_NO_SLOTS_FREE

TABLE 41. Primary and Secondary Status Codes for Floppy Disk.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

APPENDIX B. GLOSSARY OF TERMS AND ABBREVIATIONS

The following terms and abbreviations are used in the above document.

Allocation Bit Map	-	A Volume Control Structure which contains one bit for every physical sector on the disk. The value of this bit indicates if a sector is available for allocation, or not.
BSWA	-	See "Byte Stream Work Area"
Byte Stream Work Area	-	To be defined
cylinder	-	To be defined.
Deb	-	See "Directory Entry Block"
Directory	-	A Volume Control Structure which contains an entry for every file associated with that directory.
Directory Entry Block	-	A data structure found within a Master File Directory Page
Disk Driver	-	This is a reference to the Mtos device driver which controls the communication link between the S40 and the disk hardware. For more details see ref 6
Disk Extent	-	A reference to a number of contiguous sectors on the disk. Every file is made up of a number of Disk Extents.
EOF	-	End Of File
Feb	-	See "File Entry Block"
Fhb	-	See "File Header Block"
File	-	To be defined
File Entry Block	-	A data structure found within a Directory Page.
File Header Block	-	A File Control Structure which is located with a system file 'FileHeaders.sys'. A File Header Block contains the information necessary to access data in that file.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

File Index - A File Index is an integer value, which is returned to the application, when a file is opened. This index must be supplied by the application, in order to identify the file, whenever the application requires access to that file.

lfa - See Logical File Address.

Logical File Address - A logical file address gives the number of bytes offset from byte zero of the file.

Magic Number - To be defined.

Master File Directory - A Volume Control structure which contains an entry for every directory on the disk. The Master File Directory is made up of number of Directory Entry Blocks. The Master File Directory resides in a system file called 'Mfd.sys'.

Mfd - See "Master File Directory"

Page - See "Sector."

pfa - See Physical file address.

Physical File Address - The Physical file address is the physical address on disk, of a sector of a file. The Physical file address is calculated by the volume control routines based on file header information, and the Logical file address.

Sector - A term used to describe a 512 byte data block on the disk.

Surface - To be defined

Track - To be defined.

Volume - To be defined.

Driver The MTOS [1,6,7] device driver whose design is specified in section 10.

Device The floppy disk drive and Controller attached to the communication link.

The information contained within this document is confidential and proprietary to Burroughs Corporation.

Controller	The firmware executing in the disk device to control its operation and communicate with the Driver.
Control Task	The task in the Operating System which manages disk access.
Command	A command that is issued by the Driver to the Controller.
RIO	Request for Input/Output in MTOS [1,6,7].
RIO Function	An IO function requested by the Control Task. The request is made to the Driver.
RIO Status	The four byte status returned to the requesting task on completion of the RIO function.
Parameter Data	The parameters required by a disk command [3].
Status Data	The ten byte block returned by the Controller on completion of a command [3].
Communication Link	The communication cable and controlling line drivers between the S40 main unit and the floppy disk drive. All communication takes place using this cable.
SCC	Is a Serial Communications Controller (Zilog Z8530) used to to drive the high speed serial communication link to the disk [8].
SDLC	Synchronous Data Link Control is the name of the protocol used to perform data transfer on the communication link. Only a subset of the SDLC protocol is used.
CRC	Cyclic Redundancy Check is a polynomial function applied to all bytes of data transferred using SDLC which can be verified to ensure data integrity.
Disk Extent	BTOS term refering to a contiguous area of disk.
Sector Address	4 byte disk address. This address specifies a particular sector on the disk. All operations are performed on sector boundaries.

The information contained within this document is confidential and proprietary to Burroughs Corporation.