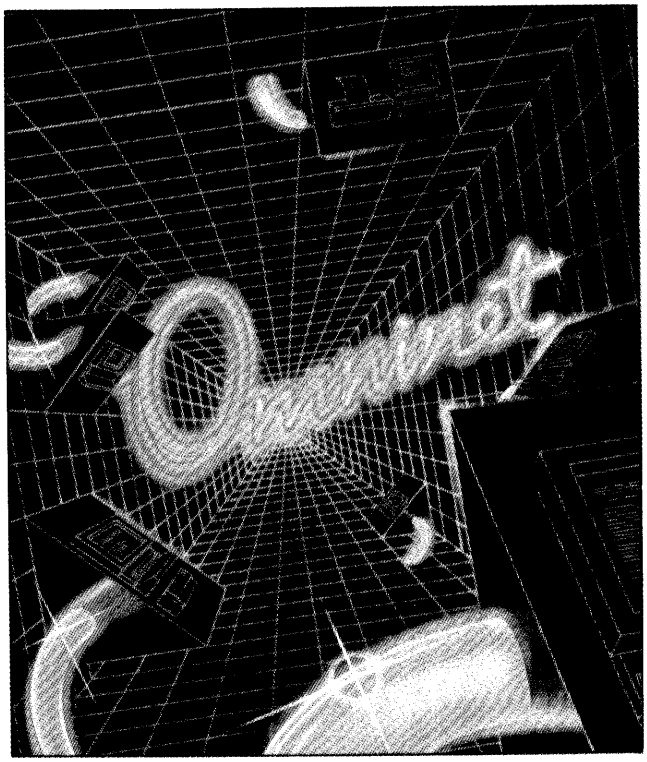
General Technical Information

Constellation® Software



CORVUS

LIMITED WARRANTY

Corvus warrants its hardware products against defects in materials and workmanship for a period of 180 days from the date of purchase from any authorized Corvus Systems dealer. If Corvus receives notice of such defects during the warranty period, Corvus will, at its option, either repair or replace the hardware products which prove to be defective. Repairs will be performed and defective parts replaced with either new or reconditioned parts.

Corvus software and firmware products which are designed by Corvus for use with a hardware product, when properly installed on that hardware product, are warranted not to fail to execute their programming instructions due to defects in materials and workmanship for a period of 180 days. If Corvus receives notice of such defects during the warranty period, Corvus does not warrant that the operation of the software, firmware or hardware shall be uninterrupted or error free.

Limited Warranty service may be obtained by delivering the product during the 180 day warranty period to Corvus Systems with proof of purchase date. YOU MUST CONTACT CORVUS CUSTOMER SERVICE TO OBTAIN A "RETURN AUTHORIZATION CODE" PRIOR TO RETURNING THE PRODUCT. THE RAC (RETURN AUTHORIZATION CODE) NUMBER ISSUED BY CORVUS CUSTOMER SERVICE MUST APPEAR ON THE EXTERIOR OF THE SHIPPING CONTAINER. ONLY ORIGINAL OR EQUIVALENT SHIPPING MATERIALS MUST BE USED. If this product is delivered by mail, you agree to insure the product or assume the risk of loss or damage in transit, to prepay shipping charges to the warranty service location and to use the original shipping container. Contact Corvus Systems or write to Corvus Customer Service, 2100 Corvus Drive, San Jose, CA, 95124 prior to shipping equipment.

ALL EXPRESS AND IMPLIED WARRANTIES FOR THIS PRODUCT, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO A PERIOD OF 180 DAYS FROM DATE OF PURCHASE, AND NO WARRANTIES, WHETHER EXPRESS OR IMPLIED, WILL APPLY AFTER THIS PERIOD. SOME STATES DO NOT ALLOW LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY LASTS, SO THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU.

IF THIS PRODUCT IS NOT IN GOOD WORKING ORDER AS WARRANTED ABOVE, YOUR SOLE REMEDY SHALL BE REPAIR OR REPLACEMENT AS PROVIDED ABOVE. IN NO EVENT WILL CORVUS SYSTEMS BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF OR INABILITY TO USE SUCH PRODUCT, EVEN IF CORVUS SYSTEMS OR AN AUTHORIZED CORVUS SYSTEMS DEALER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES FOR CONSUMER PRODUCTS, SO THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH MAY VARY FROM STATE TO STATE.

Constellation Software

General Technical Information

Corvus Part #: 7100-05944-01
Release date: 22 February 1985

Table of Contents

Figures

Scope

This manual describes the Constellation software provided by Corvus. It covers the Constellation I/II data structures, Constellation drivers, booting and logon procedures, and the Constellation II management utilities. It also contains an appendix which covers the operating system specific details of each implementation supported by Corvus.

Related documents

Corvus Mass Storage Systems General Technical Information
Part Number: 7100-05945-01

Omninet General Technical Information
Part Number: 7100 06614-01

You should also be familiar with the System Manager Guide for your particular system.

Conventions

Hexadecimal values are suffixed with an **h**. For example, FFh, 02h.

A **block** is always 512 bytes.

Lsb means least significant byte or least significant bit, depending on context. Similarly, **msb** means most significant byte or most significant bit.

Data structures are explained by describing each field of the structure, as shown below:

```
Field name | Off/Len | Type | Description
=============================================================
```

Field name is the name commonly used, either in the code or in this manual, to refer to the piece of data being discussed. Field names, when used in this manual, appear in **bold**.

Off is the offset of the start of this field, and **len** is the length of the field. **off** and **len** are always in bytes, unless otherwise specified; **off** is zero-based, unless otherwise specified.

Type is the data type of the field. The meaning of each type is given in the table below:

Type	Meaning
BYTE	An unsigned 8-bit value.
WORD	An unsigned 16-bit value; msb, lsb format.
FWRD	An unsigned 16-bit value; lsb, msb format; a byte-flipped word.
ADR2	An unsigned 16-bit value; msb, lsb format.
ADR3	An unsigned 24-bit value; msb..lsb format.
ADR4	An unsigned 32-bit value; msb..lsb format.
BSTR	A string of 1 or more characters, padded on the right with blanks (20h).
NSTR	A string of 1 or more characters, padded on the right with NULs (00h).
FLAG	A BYTE with bits numbered 7..0; msb..lsb format.
ARRY	An array of 1 or more BYTEs.

Description is a one or two line description of the field.

Table entries are described in the following format:

Entry	Byte	Content	(Meaning)	Field name

Entry is the offset of this entry, zero-based unless otherwise specified.

Byte is the byte offset of this field, zero-based.

Content is the actual content of this field, given either in ASCII or hexadecimal.

(Meaning) gives further explanation of the content, if necessary.

Field name is the name of this field, and refers back to the entry description.

Chapter 1: Introduction

Constellation software is written specifically for the care and management of the Corvus hard disk system. The Constellation software system allows you to create and maintain a multi-user system consisting of one or more computers in conjunction with one or more Corvus mass storage devices.

Constellation I software supports Rev B/H drives, MUX networks, and single disk server Omninet networks. Constellation II software supports these, as well as Omnidrives, Banks, and multiple disk server Omninet networks.

Access to the network is managed through user accounts. An account consists of a name, password, and a list of volumes to which the user is given access. Users must "log-on" to the network in order to start accessing their volumes. Once a user has logged-on, most applications software written for single-user systems may be used with no modification.

Constellation software differs from other networking software in two key respects:

1) Constellation software requires a system manager, who is responsible for disk space allocation and assigning user names and volume access.

2) Transparent volume sharing is NOT supported. That is, Constellation software does not detect the case where two users attempt to write to the same volume at the same time. The application must explicitly coordinate such access.

Constellation software is divided into two parts: management utilities and user utilities. Management utilities are operating system independent and are used to maintain the disk resident data structures that define the network configuration. User utilities are operating system dependent, and include a boot or logon program, a disk driver, a mount utility, and a spooler.

Overview

This manual discusses the following topics:

Data structures

Several data structures are maintained on the disks comprising the network. These include the Network User Table, the System Boot Table, and the Drive User, Volume, and Access Tables.

The data structures are described in Chapter 2.

Disk Drivers

For each supported operating system, code is provided to support the Corvus disk as a file-structured device under the host operating system. The operating systems supported are listed in Appendix B.

A typical disk driver is described in Chapter 3.

Boot and logon

For each operating system, code is provided which installs the disk driver into the host operating system, and initializes the mount table for a given user.

The boot and logon process is described in Chapter 4, with specific details for each operating system given in Appendix B.

Utilities

Constellation software also encompasses a set of user utilities and a set of management utilities which are used to access and maintain the data structures. User utilities must run under the host operating system, whereas management utilities may be run from any available computer.

The utilities are described in Chapters 5 and 6.

How to use this manual

In this manual, each chapter discusses the functionality of the subject piece of software. For instance, the chapter on disk drivers covers the functions a disk driver must provide. The chapters are written for the ideal world, where space and time are not limitations. In reality, not all functions can be implemented for each operating system. Specific implementation details, showing some of the limitations that must be acknowledged, are given in Appendix B.

This manual should be useful to several kinds of readers:

- Software developers - Appendix B gives specific implementation details about memory locations used and volume formats. Software developers who are having trouble getting their applications to run with the Corvus software installed may consult the appropriate section of Appendix B to see where conflicts might exist.

  Additionally, the implementation of Constellation software uses several of the features of Corvus mass storage systems, including semaphores, the boot protocol, and the Active User Table (These features are described in the *Corvus Mass Storage System General Technical Information Manual*). You can refer to this manual for examples of how to use these features in your own applications, such as electronic mail or a network data base.

- Implementing support for a new operating system - Chapters 3, 4, and 5 describe the pieces of software which must be written: a disk driver, boot or logon code, and user utilities. You will need to refer to Chapter 2 for descriptions of data structures.

- Implementing or modifying the management utilities - Chapter 6 gives descriptions of the management utilities. You will need to refer to Chapter 2 for descriptions of data structures.

Chapter 2: Constellation II data structures

This chapter describes the data structures used by Constellation II.

Overview

Each network, which consists of one or more disk servers, has a Network User Table. The Network User Table contains entries for all users of the network, and is stored at each disk server. Also stored at each disk server is the System Boot Table. The System Boot Table points to the location of the boot code for each computer type attached to the network. The user information and boot code is stored at all disk servers so that it can be obtained from any disk server on the network, as required by the cold boot procedure (see Chapter 4).

One or more disk drives is attached to each disk server. Information local to each disk drive is stored only on the particular drive. This information includes the following:

- Drive Information Block - drive name and password
- Drive Volume Table - location of each volume
- Volume Access Table - volume access information
- Drive User Table, Drive Access Table - user access information

Rather than reserve a large fixed area for the Constellation II tables, we chose instead to reserve 1 block, through which all of the other tables may be located. Block 8, the Drive Information Block, is this reserved block. All other tables are located in a volume known as the Corvus volume. This volume is like any other volume: it has an entry in the Drive Volume Table, and it may be located anywhere on the drive. The only identifier of the Corvus volume is a pointer to it in the Drive Information Block.

The Corvus volume is a UCSD-format volume (see Appendix B for details on UCSD volume format). All of the tables in the Corvus volume are treated as data files, and each has an entry in the volume directory.

Figures 2.1a and 2.1b show where the data structures are located on each of the disk(s) comprising a network.

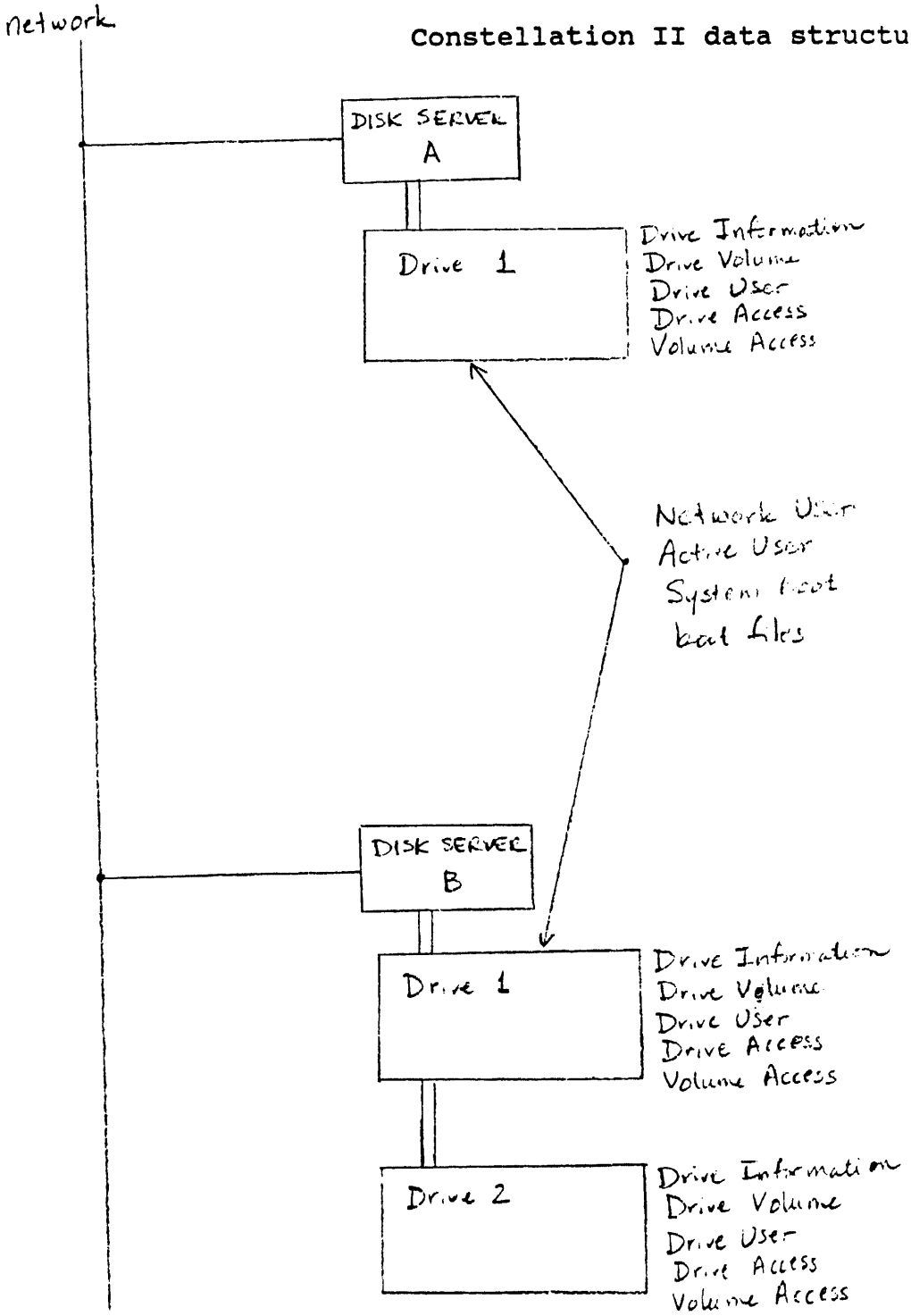


Figure 2.1a: Constellation II network layout

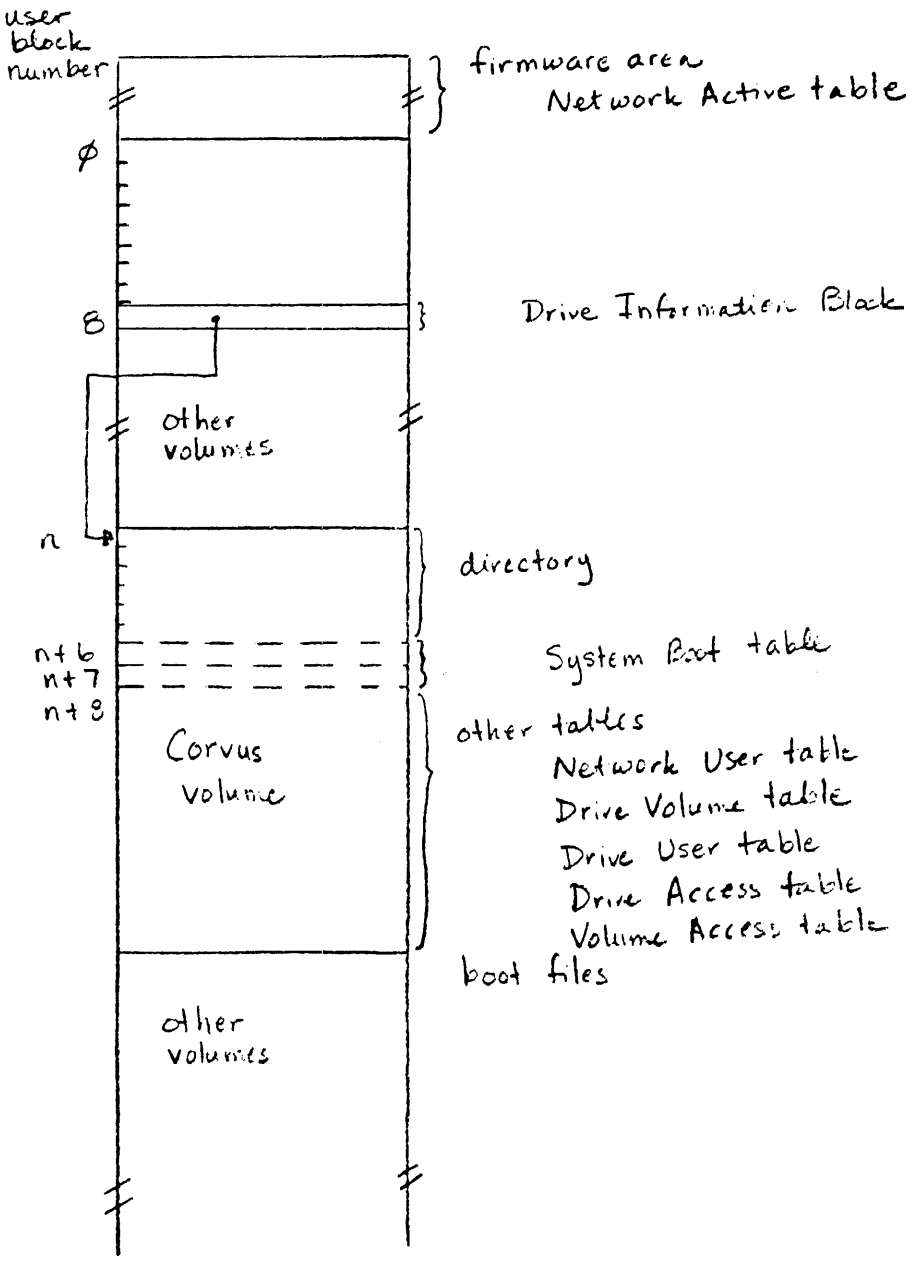


Figure 2.1b: Constellation II disk layout

Detailed table descriptions

Each structure is described in detail in Figures 2.3 through 2.10. You should probably skip to the next section, *Table manipulation*, on first reading.

There are implicit links among the Constellation II tables; that is, for each entry in a particular table, a corresponding entry may occur in another table. Figure 2.2 shows these implicit links. Required links are indicated by a double arrow, and optional links are indicated by a single arrow. Field names appear in **bold**.

For instance, the double arrow pointing at Drive User indicates that for each entry in Drive Access, there must be an entry in Drive User that has the same value in the **Userid** field. In other words, if there is an entry in Drive Access with a **Userid** of 1, then there must be an entry in Drive User with a **Userid** of 1, and these two entries refer to the same user.

As another example, the single arrow pointing from Drive Volume to Volume Access indicates that for each entry in Drive Volume, there may be an entry in Volume Access that has the same value in the **Startblk** field. In other words, if there is an entry in Drive Volume with a **Startblk** of 1024, then there may be an entry in Volume Access with a **Startblk** of 1024 as well. If there is, then the two entries refer to the same volume.

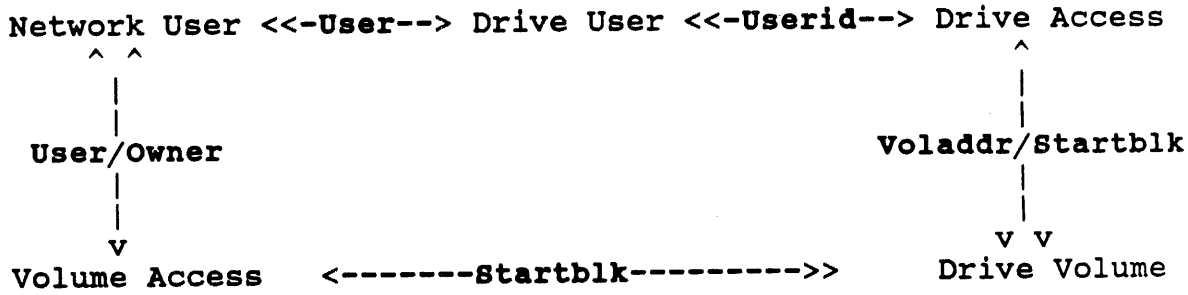


Figure 2.2: Implicit links among Constellation II tables

File name: N/A

Location: Block 8 of each drive.

Description: The Drive Information Block occupies a fixed location on each disk on a Corvus network. This is the table used to find everything else.

Access: All Corvus utilities accessing the Corvus volume make use of this table since the address of the Corvus volume is located here. Only SYSGEN and FIXIT modify the information contained here.

Format: 1 entry of 512 bytes

Field Name	Off/Len	Type	Description
Drive	0/10	BSTR	Drive name (unencrypted)
Drivepass	10/8	BSTR	Drive password (encrypted)
Server	18/10	BSTR	Server name (unencrypted)
Serverpass	28/8	BSTR	Server password (encrypted)
Crvsaddr	36/4	ADR4	Block address of Corvus volume
Size	40/4	ADR4	Size of Corvus volume, in blocks
Bootaddr	44/4	ADR4	Block address of System Boot table (SYSTEM.BOOT)
Unused0	48/4	ARRY	Reserved.
Driveinit	52/1	BYTE	Drive initialized flag (see Notes) 0 = not initialized 1 = initialized
Unused1	53/1	BYTE	Reserved.
Driveno	54/2	WORD	Drive number (1 to 7)

Figure 2.3: Drive Information Block (block 8)
(continued on next page)

Field Name	Off/Len	Type	Description
Online	56/1	BYTE	Drive online indicator (see Notes) 0 = not online 1 = online
Unused2	57/1	BYTE	Reserved.
NUblks	58/2	WORD	NETWORK.USER table size, in blocks
DVblks	60/2	WORD	DRIVE.VOLUME table size, in blocks
DUblks	62/2	WORD	DRIVE.USER table size, in blocks
DAblks	64/2	WORD	DRIVE.ACCESS table size, in blocks
SBblks	66/2	WORD	SYSTEM.BOOT table size, in blocks
Special	68/2	WORD	Version code = 1982 (see Notes)
NUaddr	70/4	ADR4	NETWORK.USER table block address
DVaddr	74/4	ADR4	DRIVE.VOLUME table block address
DUaddr	78/4	ADR4	DRIVE.USER table block address
DAaddr	82/4	ADR4	DRIVE.ACCESS table block address
SBaddr	86/4	ADR4	SYSTEM.BOOT table block address
DIversion	90/1	BYTE	Drive Information version (0 or 1)
*Unused3	91/1	BYTE	Reserved.
*VAblks	92/2	WORD	VOLUME.ACCESS table size, in blocks
*VAaddr	94/4	ADR4	VOLUME.ACCESS table block address
Reserved	98/414	ARRY	Reserved -- use 00h.

Figure 2.3: Drive Information Block (block 8)
(continued from previous page)

Notes are located on the next page.

Notes: The fields **driveinit**, **online**, and **special** can be examined to determine whether a Drive Information Block is valid. I.e., if **driveinit** = 1 and **online** = 1 and **special** = 1982, then the Drive Information Block is probably valid.

In Drive Information Blocks with **DIversion** of 0, the directory of the Corvus volume was written incorrectly; each file length was 1 block shorter than it should have been, indicating unused areas where data was actually located. This is not a problem unless the user "crunches" the Corvus volume.

* These fields have been added for the Macintosh implementation.

File name: N/A

Location: A Corvus volume is located on each drive of the network, at the location indicated by the **Crvsaddr** field of the Drive Information Block.

Description: The Corvus volume contains the following tables, each embedded within the indicated file:

	Table	File name
*(1)	System Boot Table	SYSTEM.BOOT
(2)	Network User Table	NETWORK.USER
(3)	Drive Volume Table	DRIVE.VOLUME
(4)	Drive User Table	DRIVE.USER
(5)	Drive Access Table	DRIVE.ACCESS
*(6)	boot files	BOOT.xxx
*(7)	Volume Access Table	VOLUME.ACCESS

*These tables are optional.

The Corvus volume is a standard UCSD format volume. Volume size may range from 64 to 600 blocks.

Access: Only Constellation II utilities modify the data within the Corvus volume. The tables are normally found via the directory, although some utilities use the addresses found in block 8 instead.

Format: Offset and length are in blocks.

Field Name	Off/Len	Type	Description
-	0/2	-	Unused.
-	2/4	-	UCSD format directory (see Appendix B)
-	6/2	-	SYSTEM.BOOT file
-	8/ end	-	Other files, always in the following order: NETWORK.USER, DRIVE.VOLUME, DRIVE.USER, DRIVE.ACCESS, then zero or more BOOT.xxx files.

Figure 2.4a: Corvus Volume

File name	EOF	Date	Off:	Len	Type
SYSTEM.BOOT	512	1 Sep 84	6:	2	data
NETWORK.USER	512	1 Sep 84	8:	32	data
DRIVE.VOLUME	512	1 Sep 84	40:	32	data
DRIVE.USER	512	1 Sep 84	72:	32	data
DRIVE.ACCESS	512	1 Sep 84	104:	98	data
BOOT.IBM	512	1 Sep 84	202:	10	data
<...>			212:	88	

Figure 2.4b: Example Corvus volume directory

Notes: Although the Corvus volume is a standard UCSD volume, and it can be mounted under either the p-system or CCOS operating systems, you should **never** crunch the Corvus volume. If you do crunch the Corvus volume, you will invalidate the addresses in the Drive Information Block, and any utilities, including the boot code, which use these addresses will no longer work correctly. If you do inadvertently crunch the Corvus volume, you can try to recover by executing the management utility FIXIT.

File name: SYSTEM.BOOT

Location: Corvus Volume. The contents of the System Boot Table should be the same on each drive on the network.

Description: The System Boot Table is always the first file of the Corvus volume. It is used by the Corvus Read Boot Block command described in the *Corvus Mass Storage System GTI*. Each entry in the table is the relative block address of the start of the boot code file for a particular computer.

Access: The table is manipulated by the Corvus BOOT MANAGER utility.

Format: 2 bytes per entry; space for 256 entries. Unused entries contain FFh, FFh. Entries are ordered by boot code number (see Table A.4).

Field Name	Off/Len	Type	Description
-	0/2	WORD	Block offset to start of boot file, relative to start of Corvus volume.

Figure 2.5: System Boot Table

Notes: The SYSTEM.BOOT file is always allocated 2 blocks; the second block is not used.

Entry	Byte	Contents
0	0	FFh
	<	<
	>	>
	17	FFh
9	18	00h
	19	CAh
10	20	FFh
	<	<
	>	>
	511	FFh

Figure 2.5b: Example System Boot Table

Notes: In the example above, there is one entry in the System Boot table, entry number 9. Entry number 9 is assigned to the IBM PC (see Table A.4 for computer number assignments). The entry for the IBM PC, 00CAh, is equal to 202 (decimal), and indicates that the boot code file for the IBM PC starts at block 202 of the Corvus volume. If everything is valid, then the Corvus volume should have a directory entry for the file BOOT.IBMPC with a starting block address of 202.

File name: NETWORK.USER

Location: Corvus volume. The contents of the Network User Table should be the same on each drive on the network.

Description: The Network User Table identifies each user of the network. Each valid user of the network must have an entry in the user table.

Access: Entries are modified by the USER MANAGER utility.

Format: Table size ranges from 4 to 32 blocks (a maximum of 511 users requires 32 blocks). Each entry is 32 bytes long, allowing 16 entries per block. The first 32 bytes of the first block are reserved. Unused entries contain all 00h; end of table is indicated by an entry containing all FFh; see next section, Table manipulation, for more information.

Sort field: **User**

Field Name	Off/Len	Type	Description
User	0/10	BSTR	User or device name (encrypted)
Userpass	10/8	BSTR	User password (encrypted)
Server	18/10	BSTR	Home disk server name (encrypted)
Opsys	28/2	WORD	Operating system type (see Notes)
Hostkind	30/1	BYTE	Type of user or device (see Notes)
Hoststa	31/1	BYTE	Network address (devices only)

Figure 2.6a: Network User Table

Notes: Values for **Opsys** are positive integers; meanings are given in Table A.3. Values for **Hostkind** are whole numbers; meanings are given in Table A.1; only generic meanings are used for this table. If **Hostkind** = 0, the entry is for a user, and **Hoststa** is ignored. If **Hostkind** <> 0 and **Hoststa** <> 0, then the boot code will perform an automatic logon for this user; see Chapter 4, *Special versions of boot code*.

Entry	Byte	Contents	(Meaning)	Field name
0	0	FFh		Reserved
	<		<	
	>		>	
	31	FFh		
1	32-41	g..C.%.N}b	(DENISE)	User
	42-49	yPI.t=cs	(PITSCH)	Userpass
	50-59	... i..T.h	(S)	Server
	60-61	0010h	(CCOS)	Opsys
	62	00h	(User)	Hostkind
	63	00h	(Unused)	Hoststa
2	64-73	... i..T.h	(S)	User
	74-81	.$z.+.'p	( )	Userpass
	82-91	....c..N}b	(S)	Server
	92-93	0000h	(Unused)	Opsys
	94	01h	(Disk)	Hostkind
	95	00h	(00h)	Hoststa
	96	FFh		Unused
	<		<	
	>		>	
	511	FFh		

Figure 2.6b: Example Network User Table

Notes: In the entries above, **User**, **Userpass**, and **Server** are encrypted; the unencrypted values are given in parentheses.

File name: DRIVE.VOLUME

Location: Corvus Volume

Description: The Drive Volume Table contains an entry for each volume located on the drive. All operating system dependent information must be embedded in the volume itself. The Drive Volume Table contains an entry for the Corvus volume.

Access: All modifications to the Drive Volume Table are performed by the Corvus VOLUME MANAGER utility.

Format: 4 - 64 blocks (maximum allows 511 volumes/drive); 32 bytes per entry; 16 entries per block; the first 32 bytes of the first block are reserved. Unused entries contain all 00h; end of table is indicated by an entry containing all FFh; see next section, Table manipulation, for more information.

Sort field: **Startblk**

Field Name	Off/Len	Type	Description
Volname	0/10	BSTR	Volume name (encrypted)
Startblk	10/4	ADR4	Address of first block of volume
Endblk	14/4	ADR4	Address of last block of volume
Voltype	18/1	BYTE	Directory type (see notes)
Writeable	19/1	BYTE	Write flag 0 = volume is read only 1 = volume is read/write
Glbaccess	20/1	BYTE	Global access flag 0 = volume is not accessible 1 = volume is accessible
Voloffset	21/1	BYTE	Beginning block offset of volume
Reserved	22/4	NSTR	Reserved.
ImgDate	26/2	ADR2	Image date (see notes)
ImgBlock	28/4	ADR4	Image volume block number

Figure 2.7a: Drive Volume Table

Notes: **Voloffset** is normally 0, but can be used to reserve some blocks at the start of a volume for additional volume information. When mounting a volume, the starting block address should be calculated as **startblk + voloffset.**

Voltype is a value between 1 and 255 which indicates the volume directory type. Values are given in Table A.2.

ImgDate and **ImgBlock** are only significant if voltype is 18 or 19, which means that this is an Image volume.

Entry	Byte	Contents	(Meaning)	Field name
0	0	FFh		Reserved
	<		<	
	>		>	
	31	FFh		
1	32-41	e..X.;.V.j	(CORVUS)	Volname
	42-45	00000A00h	(1024)	Startblk
	46-49	00000B2Bh	(1223)	Endblk
	50-53	01h, 01h, 01h, 00h	(UCSD, yes, yes, 0)	Voltype, Writeable, Glbaccess, Voloffset
	54-57	00h, 00h, 00h, 00h		Reserved
	58-63	00h	(Unused)	ImgDate, ImgBlock
2	64-73	e..[...V.j	(CCSYS)	Volname
	74-77	00000B2Ch	(1224)	Startblk
	78-81	00000B2Bh	(....)	Endblk
	82-85	01h, 01h, 01h, 00h	(UCSD, yes, yes, 0)	Voltype, Writeable, Glbaccess, Voloffset
	86-89	00h, 00h, 00h, 00h		Reserved
	90-95	00h	(Unused)	ImgDate, ImgBlock
	96	FFh		
	<		<	
	>		>	
	511	FFh		

Figure 2.7b: Example Drive Volume Table

Notes: In the entries above, **Volname** is encrypted.

File name: DRIVE.USER

Location: Corvus Volume

Description: The Drive User Table is used to map user names to volumes accessible on a particular drive. Each user with access to a volume on the drive has an entry in the Drive User Table. The **userid** field of each entry corresponds to the **userid** field in the Drive Access Table.

Access: All table additions are made via the Corvus ACCESS MANAGER utility. Both the ACCESS MANAGER and the USER MANAGER utilities may delete entries.

Format: 4 - 32 blocks (maximum of 511 entries requires 32 blocks); 32 bytes per entry; 16 entries per block; first 32 bytes of the first block are reserved. Unused entries contain all 00h; end of table is indicated by an entry containing all FFh; see next section, Table manipulation, for more information.

Sort field: **User**

Field Name	Off/Len	Type	Description
User	0/10	BSTR	User name (encrypted)
Userpass	10/8	BSTR	User password (encrypted)
Userid	18/2	ADR2	User number on this drive
Mounted	20/2	ADR2	Number of user volumes mounted
Access	22/2	ADR2	Number of user volumes accessible
Bootinfo	24/2	ADR2	User boot volume location flag 0 = not on this drive 1 = on this drive
Reserved	26/6	NSTR	Reserved.

Figure 2.8a: Drive User Table

Entry	Byte	Contents	(Meaning)	Field name
0	0	FFh		Reserved
	<			
	>			
	31	FFh		
1	32-41	g..C.%.N}b	(DENISE)	User
	42-49	yPi.t=cs	(PITSCH)	Userpass
	50-51	0002h	(2)	Userid
	52-55	0002h, 000Eh	(2, 14)	Mounted, Access
	56-57	0001h	(yes)	Bootinfo
	58-63	00h	(Unused)	Reserved
2	64-73	...Ri..T.h	(SMGR)	User
	74-81	iIu./.dt	(HAI)	Userpass
	82-83	0001h	(1)	Userid
	84-87	0003h, 0005h	(3, 5)	Mounted, Access
	88-89	0001h	(yes)	Bootinfo
	90-95	00h	(Unused)	Reserved
3	96	FFh		Unused
	<			
	>			
	511	FFh		

Figure 2.8b: Example Drive User Table

Notes: In the entries above, **User** and **Userpass** are encrypted.

File name: DRIVE.ACCESS

Location: Corvus Volume

Description: The Drive Access Table identifies each volume on the drive to which the user has access. Each entry contains the **userid** of a user identified in the Drive User Table, the physical volume location, and user access information. The mount information **mntinfo** indicates which drive specifier a volume is associated with on boot.

Access: The Drive Access Table is modified by the Corvus ACCESS MANAGER. The USER MANAGER, VOLUME MANAGER, and ACCESS MANAGER may delete entries.

Format: 4 - 256 blocks (512 users, with access to 256 volumes each, on a single drive requires 256 blocks); 8 bytes per entry; 64 entries per block; first 32 bytes of first block are reserved. Unused entries contain all 00h; end of table is indicated by an entry containing all FFh; see next section, Table manipulation, for more information.

Sort field: **Userid** concatenated with **voladdr**

Field Name	Off/Len	Type	Description
Userid	0/2	ADR2	User number on this drive
Voladdr	2/4	ADR4	Starting block of volume
Mntinfo	6/1	BYTE	OS dependent mount information
Readonly/ Bootflag	7/1	BYTE	Readonly flag - bits 7-4 0 = volume is read-write 1 = volume is read-only Boot volume flag - bits 3-0 0 = not user's boot volume 1 = is user's boot volume

Figure 2.9a: Drive Access Table

Entry	Byte	Contents	(Meaning)	Field name
0	0	FFh		Reserved
	<		<	
	>		>	
	31	FFh		
4	32-33	0001h	(User 1)	Userid
	34-37	00000A00h	(1032)	Voladdr
	38-39	xxxxh	(..)	Mntinfo, Readonly, Bootflag
5	40-41	0001h	(User 1)	Userid
	42-45	00000A00h	(1032)	Voladdr
	46-47	xxxxh	(..)	Mntinfo, Readonly, Bootflag
6	48-49	0002h	(User 2)	Userid
	50-53	00000A00h	(1032)	Voladdr
	54-55	xxxxh	(..)	Mntinfo, Readonly, Bootflag
	56	FFh		Unused
	<		<	
	>		>	
	511	FFh		

Figure 2.9b: Example Drive Access Table

Notes: In the example above, user 1 has access to 2 volumes: the volume starting at block 1032, and the volume starting at block 2048. To find the name of the volume starting at block 1032, you must search the Drive Volume Table for the entry that has a value of 1032 in the field **startblk**. To find the name of user 1, you must search the Drive User Table for the entry that has a value of 1 in the field **userid**.

File name: VOLUME.ACCESS

Location: Corvus Volume

Description: The Volume Access Table contains an entry for each volume located on the drive, giving access information for the volume.

Access: All modifications to the Volume Access Table are performed by the Corvus VOLUME MANAGER utility.

Format: 4 - 64 blocks (maximum allows 511 volumes/drive); 32 bytes per entry; 16 entries per block; the first 32 bytes of the first block are reserved. Unused entries contain all 00h; end of table is indicated by an entry containing all FFh; see next section, Table manipulation, for more information.

Sort field: **Startblk**

Field Name	Off/Len	Type	Description
Startblk	0/4	ADR4	Address of first block of volume
Owner	4/10	BSTR	Name of user with R/W access (encrypted)
Volpsw	14/8	BSTR	Volume password (encrypted)
AccType	22/2	WORD	Access type 0 = uncontrolled; 2 = public; 1 = controlled; 3 = private
Case AccTyp	24/8	ARRY	Access information
Lock	24/1	BYTE	If AccType = 1 (controlled), then 0 = Volume is not locked 1 = Volume is currently locked

Figure 2.10a: Volume Access Table

Notes: **Acctype** specifies the type of access allowed for this volume. If no entry exists for a volume in the Volume Access table, or if no Volume Access table exists, then the volume is mounted using information from the Drive Access table.

The mount algorithm for each type of access is given below. The field **Writeable** is from the corresponding entry in the Drive Volume table, and denotes either RO (**writeable**=0) or RW (**writeable**=1); the variable Username is the user name the user is logged in under. The parameter on the call to MountIt is either RO (read-only) or RW (read-write).

```
{ start of mount algorithm }

Psw2Str( volpsw, pswd );  { decrypt volume password }
NeedPswd := LENGTH(Pswd>0); { does volume have a password? }
IF NeedPswd THEN GetPswd(pass);  { prompt user for volume password }

CASE Acctype OF

Uncontrolled: IF ((NeedPswd) AND (Pswd=pass)) OR (not NeedPswd)
                 THEN MountIt(Writeable)

  Controlled: IF ((NeedPswd) AND (Pswd=pass)) OR (not NeedPswd)
                 THEN MountIt(RO)

      Public: IF ((NeedPswd) AND (Pswd=pass)) OR (not NeedPswd)
                 THEN BEGIN
                   IF Username = Owner THEN MountIt(Writeable)
                      ELSE MountIt(RO)
                   END;

     Private: IF Username = Owner THEN MountIt(Writeable)
END;
```

Entry	Byte	Contents	(Meaning)	Field name
0	0	FFh		Reserved
	<		<	
	>		>	
	31	FFh		
1	32-35	00000A00h	(1024)	Startblk
	36-45	.....	(SMGR)	Owner
	46-53	.....	(XYZ)	Volpsw
	54-55	0000h	(Uncontrol)	Acctype
	56-63	00h	(Unused)	Filler
2	64-67	00000B2Ch	(1224)	Startblk
	68-77	.....	(SMGR)	Owner
	78-85	.....	(ZZZZ)	Volpsw
	86-87	0001h	(Controlle)	Acctype
	88-95	00h	(Varies)	Lock
	96	FFh		
	<		<	
	>		>	
	511	FFh		

Figure 2.10b: Example Volume Access Table

Notes: In the entries above, **Owner** and **Volpsw** are encrypted.

Table manipulation

A common set of routines is used to manipulate those tables which have a defined sort field. Each such table consists of one or more table blocks, where each table block is 2 disk blocks or 1024 bytes. Each table block contains an integral number of entries; the entry size for each table was given in the previous figures.

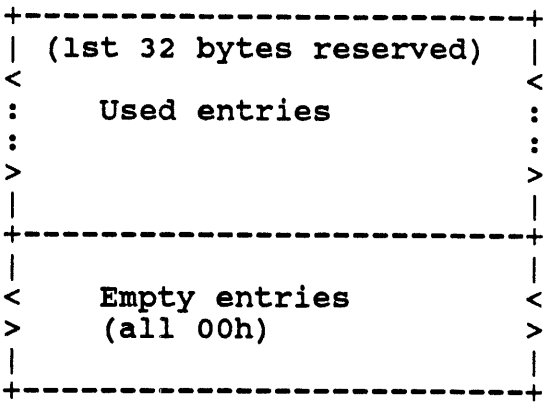
Within a table block, some entries are used, and some are empty. All used entries are grouped together at the beginning of the block, and all empty entries are always grouped together at the end of the block. Empty entries contain either 00h or FFh in all bytes of the entry. An entry containing 00h indicates that an entry has been deleted; an entry containing FFh indicates that the entry has never been used. The illustration on the next page will help you picture the table organization.

This concept of empty entries within a block was used to keep insertion and deletion times down, the idea being that table blocks would tend to have one or more empty entries.

1st table block:

The first 32 bytes of the first table block contain FFh, and should be ignored.

Once you see an entry containing all 00h's, you can skip to the next table block.



Used table block:

An empty entry always indicates that a deletion has occurred.

A table block may contain all 00h's, if many deletions have occurred.

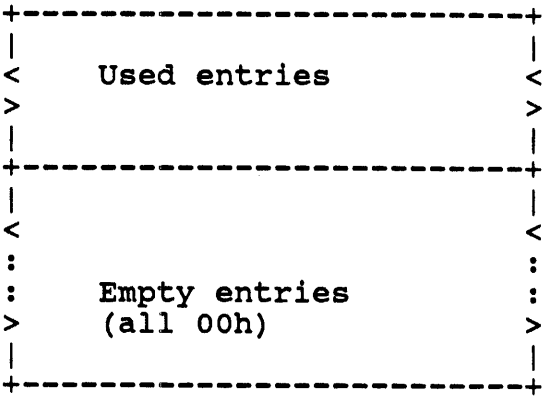
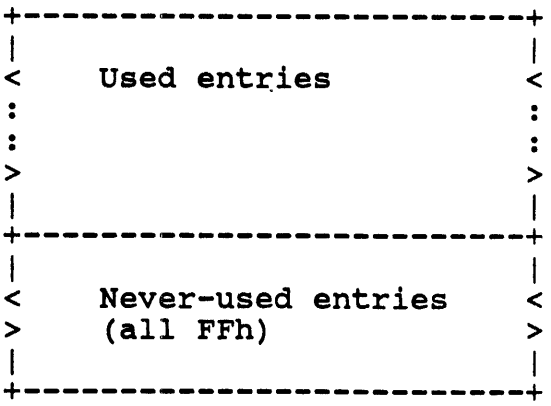


Table block with never-used entries:

Once you see a table block with an entry of FFh, all succeeding table blocks should contain only FFh.



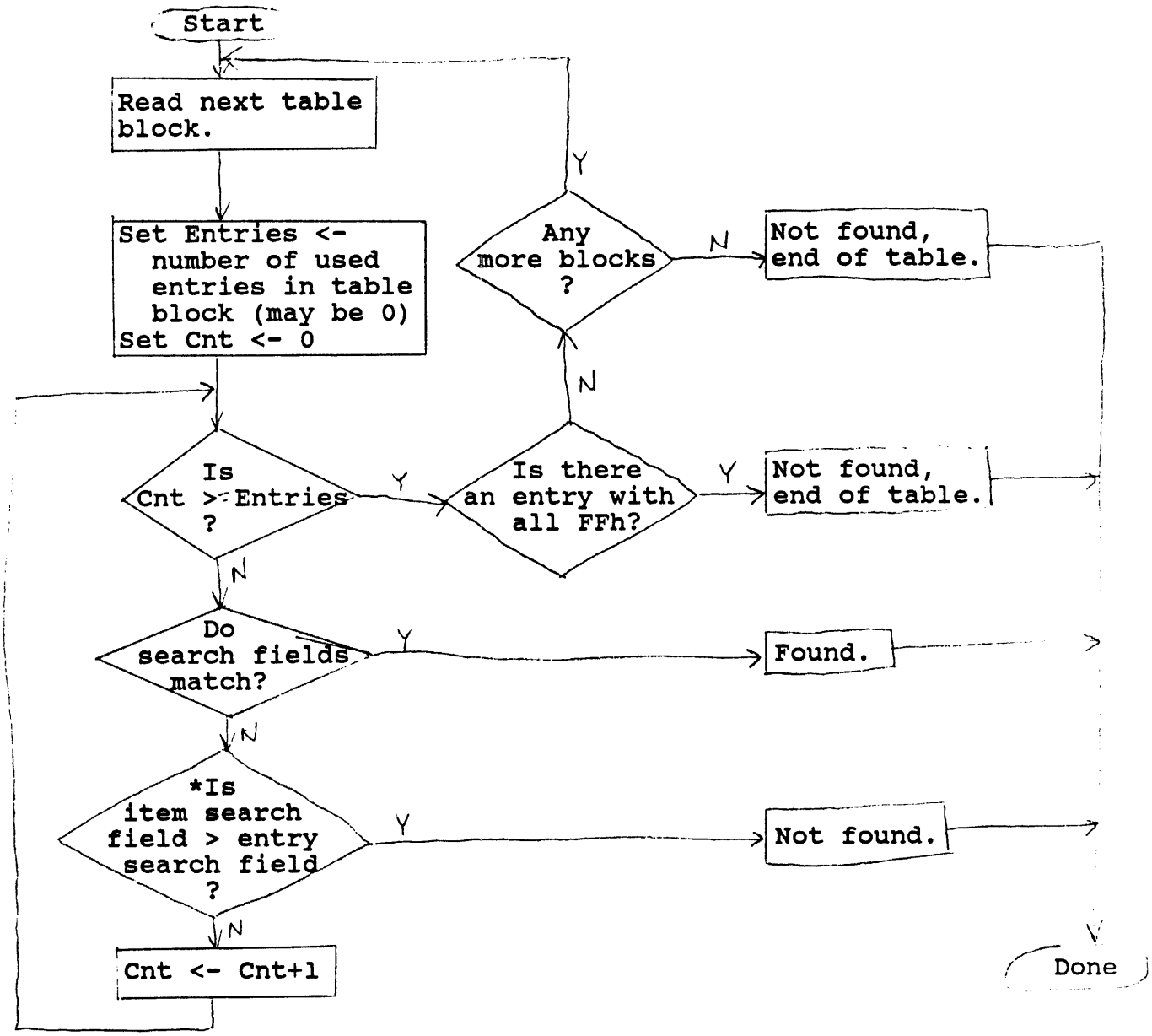
Four operations are allowed these tables: search, insert, delete, and replace. The routines used by the Constellation II management utilities to manipulate these tables are described in Chapter 6. The algorithms used for these operations are described here in order to give you a better feel for how the tables are organized.

Each table has a defined sort field. Table entries are ordered alphabetically by this sort field. The search algorithm allows you to specify which field to base the search on; the search field may or may not be the sort field. If the search field is the sort field, the search stops when an entry greater than the entry being searched for is found. Otherwise, the search stops when a never-used entry is found.

The search and insert algorithms are described in detail by Figures 2.10a and 2.10b.

Table deletion is handled by finding the desired entry and moving all following entries in the table block back one entry. The last bytes in the table block are overwritten with 00h or FFh, indicating an empty entry. FFh is used only if other never-used entries are in this block.

Replacement is handled by finding the old entry and overwriting it with the new entry.



* Test only done if search field is the sort field.

Figure 2.10a: Table search algorithm

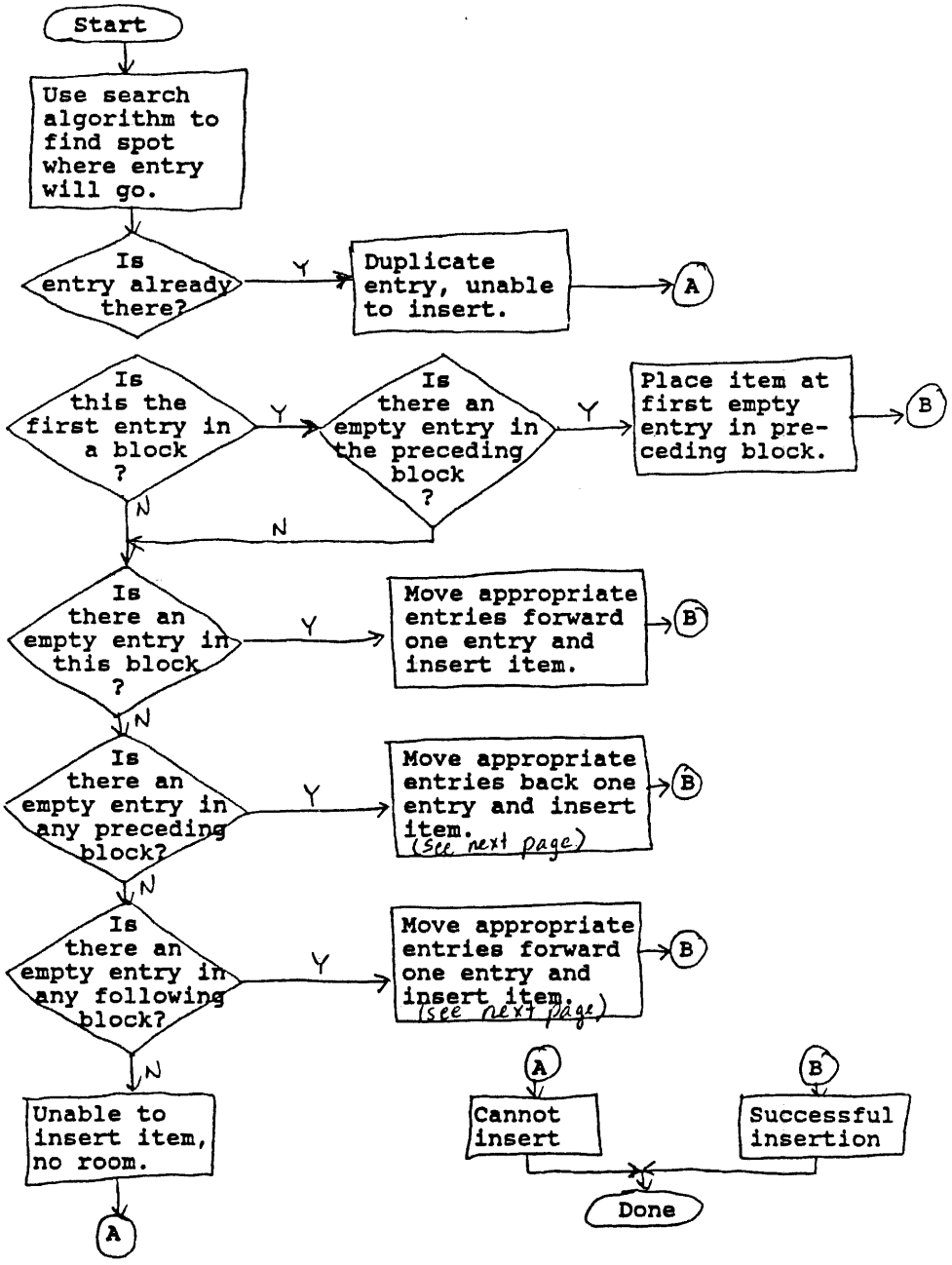
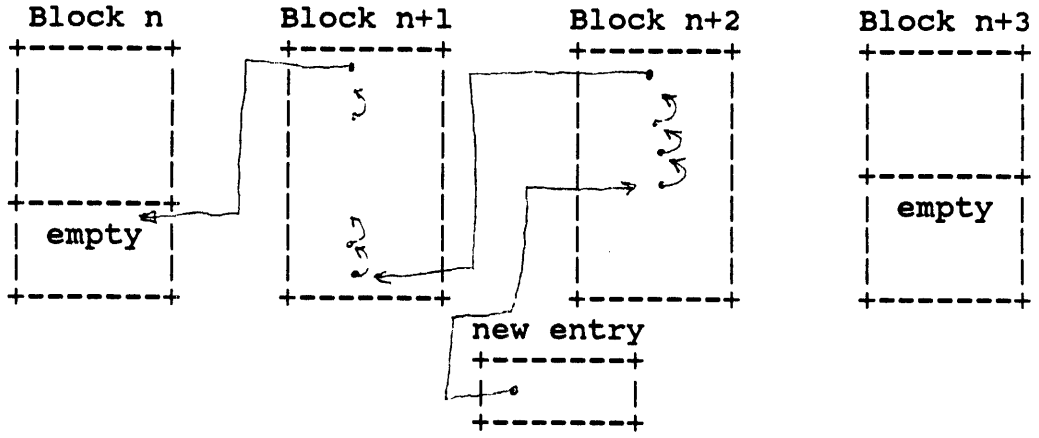


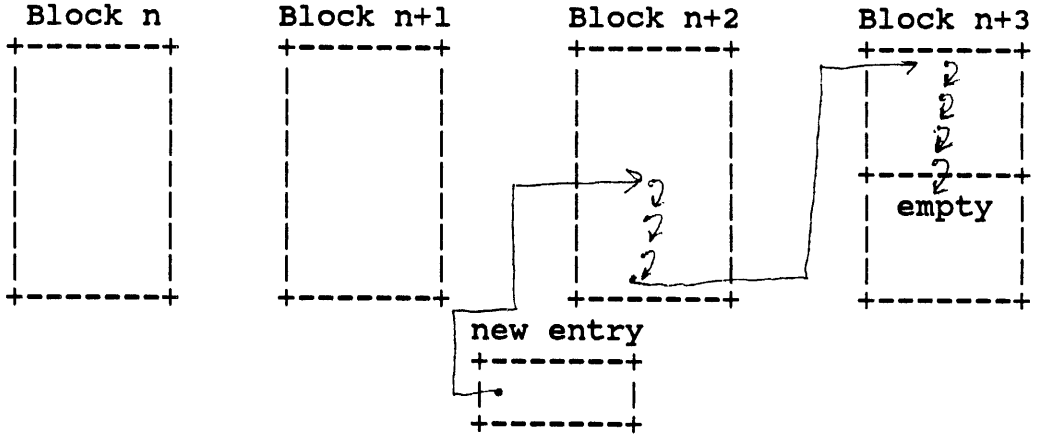
Figure 2.10b: Table insertion algorithm

The insertion algorithm for inserting into a full table block is shown below. In each case, the new entry is to be inserted into block n+2, which is full.

Case 1: a preceding block has empty entries



Case 2: All preceding blocks are full, but a following block has empty entries.



Encryption

All names and passwords in the tables are encrypted. Encrypting the names makes it harder for a malicious user to locate the Constellation II tables. Even if a user does locate the tables, figuring out how to decrypt the information is not easy, since each byte is encrypted based on its location within a name. Additionally, a different algorithm is used for encrypting names than is used for encrypting passwords.

The name encryption algorithm preserves alphabetic ordering, so that a list of encrypted names and a list of the corresponding unencrypted names will sort in the same order.

A set of procedures which performs encryption and decryption is described in Chapter 6.

Active User Table

The Active User table contains a list of those users and servers currently active on, or "logged-on" to, the network. Thus, it is a dynamic table which can be used to determine the current state of the network. As stated in the *Corvus Mass Storage Systems GTI* however, the Active User Table is not always up-to-date. The Active User table provides a rudimentary name service, and is intended for use by those applications which cannot utilize the Constellation name protocol.

File name: N/A

Location: In firmware area

Description: The Active User Table contains entries for all users or servers currently active on the network. Each new user of the network is added to the table during cold boot or via the server logon protocol. An Active User Table entry consists of the user or server name, and the host address of the user or server. This table can be read by utility programs to find the addresses of various users or servers on the network.

Access: This table is initialized by the disk server. It is modified by the firmware in response to commands from a user logon or server power-up.

Format: 4 blocks (maximum 128 entries - there are more than 64 to allow for multiple users per host)
16 bytes per entry; 32 entries per block

Field Name	Off/Len	Type	Description
Host	0/10	BSTR	User or device name
Hostno	10/1	BYTE	Network address
Hostkind	11/1	BYTE	User or device type (see Table A.1)
Reserved	12/4	BSTR	Reserved

Figure 2.12a: Active User Table

Entry	Byte	Contents	(Meaning)	Field name
0	0-9	SERVER1		Host
	10	01h	(1)	Hostno
	11	00h	(disk)	Hostkind
	12-15	00h	(Unused)	Reserved
1	16	20h		Unused
	<			
	>			
	31	20h		
2	32-41	DENISE		Host
	42	13h	(19)	Hostno
	43	20h	(Concept)	Hostkind
	44-47	00h	(Unused)	Reserved
	48	20h		Unused
	<			
	>			
	511			

Figure 2.12b: Example Active User Table

There is an Active User Table on each disk server on the network. See the *Corvus Mass Storage Systems GTI*, for descriptions of the commands which access the Active User table.

The Active User Table can be updated by choosing the List Drives On-line option of the Constellation II management utilities. The algorithm used to update the Active User table is shown in Figure 2.13.

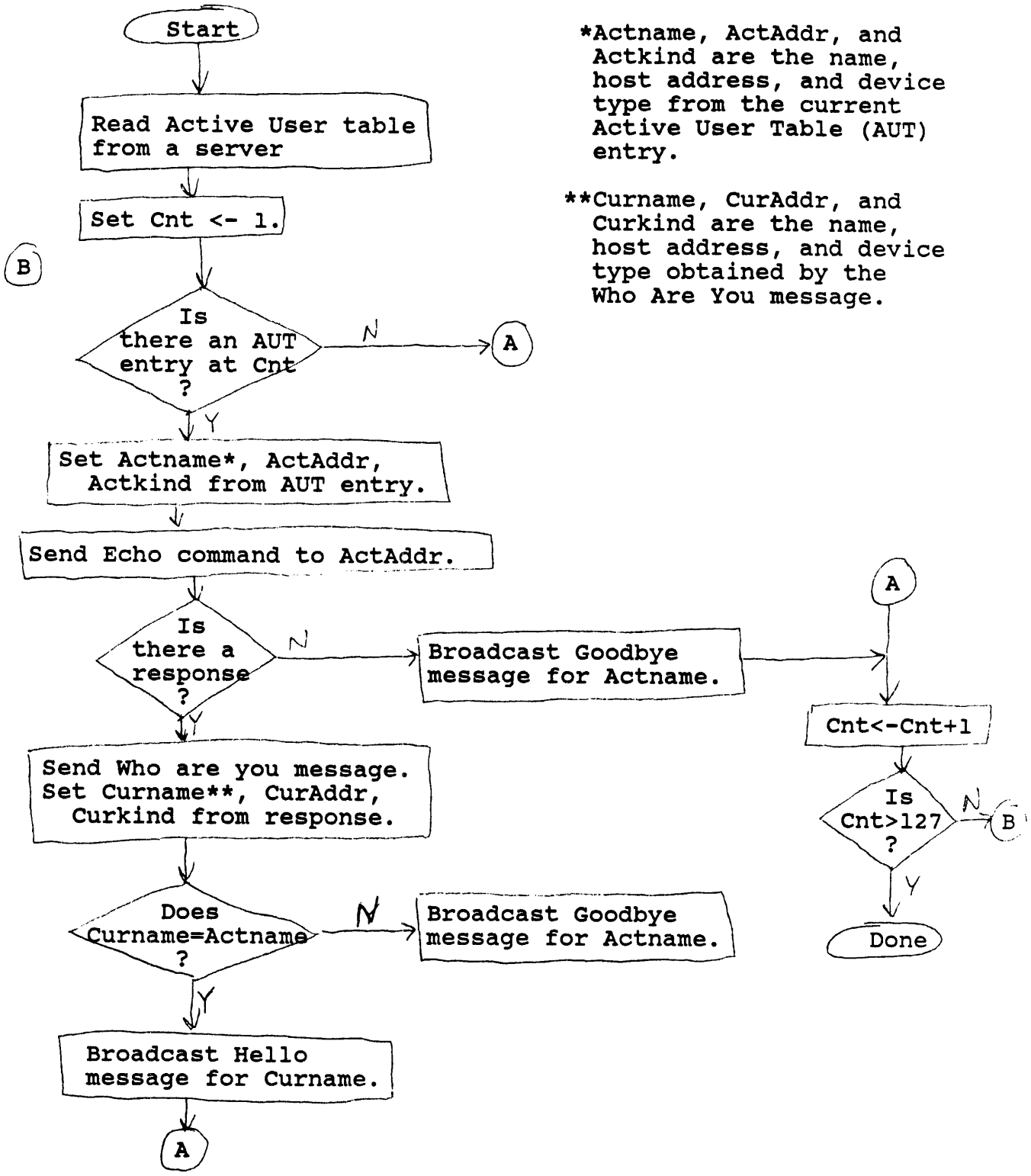


Figure 2.13: Flowchart of update Active User Table

Chapter 3: Constellation Disk drivers

A simple disk driver was described in Chapter 3 of the *Corvus Mass Storage Systems GTI*. You will have to refer to that chapter for complete information on writing a disk driver; this chapter discusses only the particular functions of a Constellation disk driver.

Figure 3.1 summarizes the disk driver functions required of a Constellation disk driver. These functions are the ideal set that a driver should provide. In reality, it is not always possible to provide all of these functions, due to limftations of the operating system interface, or space limitations. The actual functions provided by the drivers supplied by Corvus are described in Appendix B.

Function	How used
Driver initialization	Standard BIOS call
Device read	"
Device write	"
Send Corvus drive command	Utilities, user applications
Send Omninet command	"
Mount	Mount utility, user applications
Unmount	"
Get mount information	"

Figure 3.1: Disk driver functions

A Constellation disk driver maintains a structure known as the mount table. The mount table serves as a map between an operating system device number and a volume on a Corvus drive. The driver will normally support a fixed sized mount table, supporting between 6 and 10 mount table entries. We have found through experience that while 2 or 3 disk devices are sufficient for most applications, there are occasions where 6 are not enough. On the other hand, most drivers have limited data space, so you don't want to make the mount table too large.

Each entry in the mount table must contain the following information:

- Corvus disk server address - 1 byte
- Corvus disk drive number - 3 bits
- Corvus disk block address (starting block address of volume) - 3 or 4 bytes
- Corvus volume length, or ending address (2 or 3 bytes)
- Read-write flag - 1 bit

Therefore, each device entry requires at least 7 bytes. In addition, the driver must store the user's name, and may also store the user's password, home disk server address, and transporter address. Figure 3.2 shows a typical mount table description. See Appendix B for the exact format of each operating system's mount table.

```
VAR MT: RECORD
        Devices:  ARRAY [1..10] OF RECORD
           Diskserver:  BYTE;
           DriveNo:     BYTE;
           StartBlk:    PACKED ARRAY [1..4] OF BYTE;
           VolLen:      INTEGER;
           Writeflag:   BOOLEAN;
           END;
        User:           PACKED ARRAY [1..10] OF CHAR;
        Userpass:       PACKED ARRAY [1..8] OF CHAR;
        HomeDS:         BYTE;
        UserAddr:       BYTE;
        END;
```

Figure 3.2: Sample mount table declaration

Each function listed in figure 3.1 is now discussed in detail.

Driver initialization

This call is a standard BIOS function in most operating systems. It is used to perform whatever initialization is required. In the case of a Constellation disk driver, this call can be used to initialize the mount table, if this has not been done previously (e.g., during boot).

Device read, device write

The usual parameters to device read or write include the following:

Device number
Sector number
Number of sectors
Data buffer
Result code

The driver must first of all check that the device number is valid, and that a mount table entry exists for this device number. If the device number is invalid, the driver should return a result code indicating "device not on-line".

If the request is a device write, then the driver must validate that the write flag is set in this device's mount table entry. If not, the driver should return a result code indicating "write protected".

Next, the driver must validate that the sector number is valid, by comparing it to the volume length field in the mount table. If the sector number is out of range, the driver should return a result code indicating "invalid sector address".

Next, the driver must build the appropriate read or write command to be sent to the Corvus disk. Refer to Chapter 3 of the *Corvus Mass Storage Systems GTI* for a thorough description of this step, which includes building a command vector, sending it to the proper disk, and receiving the results. Remember that the sector size is operating system dependent, and that the appropriate Corvus read or write sector command must be used. The sector number specified in the driver call must be converted to an absolute disk sector address by adding it to the starting block address from the mount table. The drive number and disk server address must also be obtained from the mount table.

The read or write command must be repeated for the appropriate number of sectors, as specified in the driver call.

Send Corvus drive command (CDSEND, CDRECV)

The driver should provide an interface for sending arbitrary disk commands, such as pipes or semaphores commands. Refer to Chapter 4 of the *Corvus Mass Storage Systems GTI* for a detailed description of this function.

Mount, unmount, get mount information

The driver should also provide an entry point that allows for changing the mount table. This function can be used by applications which require certain volumes to be mounted, and by a user who has access to more volumes than the driver supports.

In Corvus implementations, these functions are often provided by two low-level functions: Read mount table and Write mount table. As their names imply, these functions simply move the entire mount table between the driver's memory and a user-supplied buffer. Since these functions require a detailed knowledge of the structure of the mount table, it is desirable to provide a user library which implements the higher-level functions mount, unmount, and get mount information. These functions are described in Chapter 5, *User Utilities*.

The Read Mount Table entry point has two parameters, both output parameters. The first parameter is an integer which contains the length of the mount table; the second parameter is a buffer which contains the mount table itself. The caller must ensure that the buffer is large enough to accomodate the mount table.

The Write Mount Table entry point has one parameter, a buffer which contains the new mount table.

Extreme caution must be used in calling these entry points, since writing incorrect information to the mount table could cause data on the Corvus drive to be overwritten.

Send Omninet command

The driver may also provide an interface for sending arbitrary Omninet messages, such as Send message or Setup Receive message. The details of this interface vary widely among implementations; you should look at the individual descriptions in Appendix B for information on this function.

Ideally, the Omninet functions should be supported by a transporter driver, and the disk driver should call on the transporter driver when sending commands. In reality, however, only the Concept Operating System currently provides both a transporter driver and a disk driver. Since this manual is intended to describe the current state of Constellation II, it does not describe a separate transporter driver. Documentation on transporter drivers will be published by Corvus as soon as it is available.

Although this manual does not describe a transporter driver, it

is important to realize that providing a separate transporter driver and disk driver is much superior to providing only a disk driver. With only a disk driver, certain sockets and data buffers of a buffered transporter must be reserved for exclusive use by the disk driver. Applications which desire to use the transporter directly must be coded so as not to interfere with the disk driver. If the disk driver changes at all, then such applications may no longer work properly. Having a transporter driver which controls access to sockets and buffers will resolve these potential conflicts. Therefore, it is recommended that all future development of disk drivers implement both a transporter driver and a disk driver.

Chapter 4: Boot and logon procedure

This chapter describes the tasks of a typical boot from the Corvus drive. As mentioned for disk drivers, the set of tasks described here is the ideal; in reality, it is not always possible to provide all of these functions, due to limitations of the operating system interface, or space limitations. The actual functions provided by the various boot and logon programs supplied by Corvus are described in Appendix B.

This chapter discusses both boot and logon, which are two separate functions. Boot refers to the process of loading and launching the operating system; logon refers to the process of validating a user's name and mounting the user's volumes. How boot and logon are accomplished is operating system dependent. In some cases logon is performed after the boot has been completed, and in other cases it is performed as part of the boot.

There are several reasons why it is desireable to boot from the Corvus disk, rather than from some local device such as a floppy. First, one of the reasons for buying a network is to minimize hardware cost per user; if each user can boot from the Corvus disk, then each computer does not need to have a floppy drive attached. Second, it is usually faster to boot from a hard disk than it is to boot from a floppy disk. Thirdly, when a new version of the operating system is released, only the copy on the hard disk needs to be updated; with a floppy disk boot, each user's boot floppy needs to be updated.

Despite the many advantages of booting from the Corvus disk, there are computers for which Corvus does not support booting. Some computers do not have the capability to boot from an external device. In other cases, the operating system supplier does not make available the information which Corvus must have in order to develop boot code.

Corvus boot protocol was established to allow many types of computers to boot from one disk. We didn't want to reserve disk sectors for each computer type, which would be wasteful of space since most networks have only one or two computer types connected. We also chose not to reserve a large fixed area. Instead, we chose to reserve one block (block 8), to be used as an indirect pointer to where the boot code is located. A second indirect pointer indicates where, within the boot blocks, a particular computer's boot code is located.

Each computer that is supported by Corvus is assigned a number which is its "boot" number. (See Figure 4.1 for boot number

assignments.) The boot number is used to indicate the location of that computer's boot code. The Read Boot Block command, a firmware command to the disk controller, is used to read the boot blocks. Thus, any computer can do a cold boot from the Corvus drive by issuing the Read Boot Block command to load its boot code and then using that code to bootstrap itself up. You should refer to the description of the Read Boot Block command given in the *Corvus Mass Storage Systems GTI*.

The BOOT MANAGER program is provided to allow easy update of the boot blocks. The BOOT MANAGER maintains the boot blocks as files within the Corvus volume. For further information on BOOT MANAGER, see Chapter 6.

Figure 4.2 shows a drive which is set up to boot 2 computer types: IBM PC and TI Pro. You may wish to refer back to the description of the System Boot table in Chapter 2.

boot number	computer type	boot file name
0, 1, 2, 3	Apple II	BOOT.APPLE2
4, 5, 6, 7	Concept	BOOT.CONCEPT
9	IBM	BOOT.IBMPC
10	Xerox 820	BOOT.XR820
11	Zenith H89	BOOT.HZ89
12	NEC PC8000	BOOT.NC
13	Pet	BOOT.PET
14	Atari 800	BOOT.ATARI800
15	TRS-80 MOD I	BOOT.TRSI
16	TRS-80 MOD III	BOOT.TRSIII
17	LSI-11	BOOT.LSI11
18	Printer server	BOOT.PRINTSRV
19	Apple ///	BOOT.APPLE3
20	DEC Rainbow	BOOT.RAINBOW
21	TI Pro	BOOT.TIPRO
22	Z-100	BOOT.Z100
23	Concept2	BOOT.CONCEPT2
24	Companion	BOOT.COMPANION
25	Macintosh	BOOT.MAC
26	Sony SMC-7086	BOOT.SONY

Figure 4.1: Boot number assignments and file names

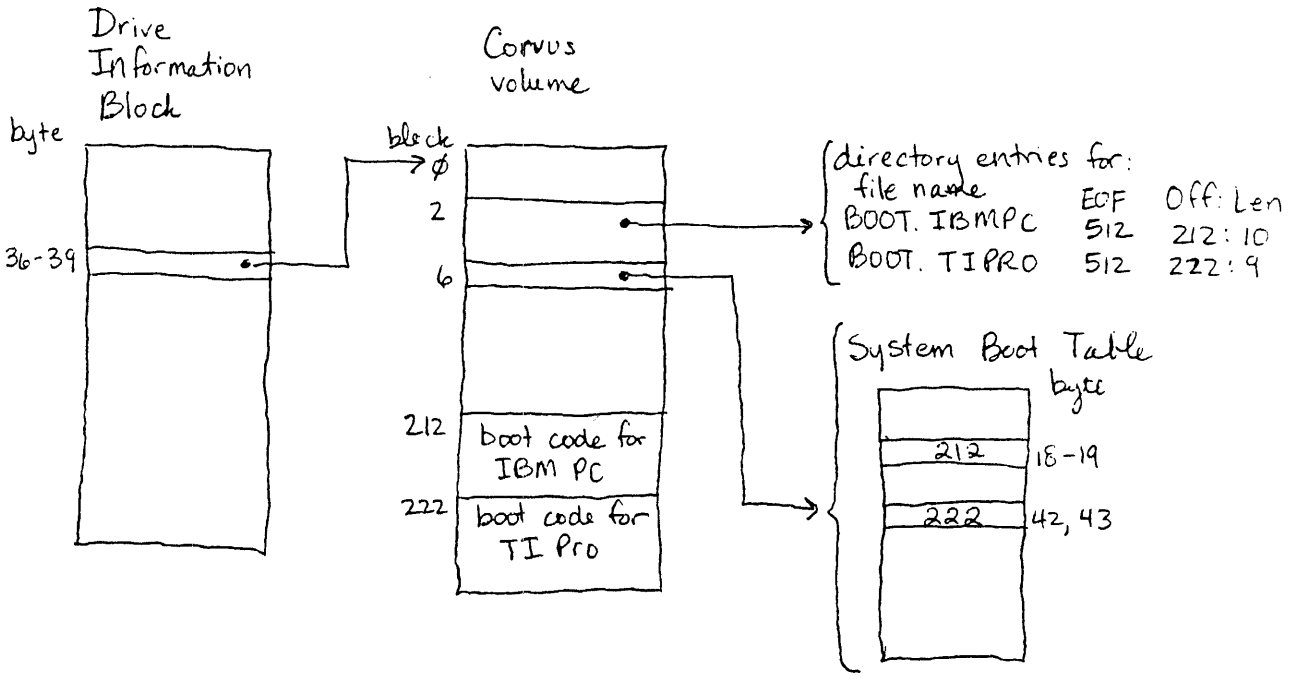


Figure 4.2: Boot tables for 2 computers

The boot code is generally divided into two parts. The first part, referred to as the first stage boot, is generally resident in a boot ROM, and must perform the following steps:

1) Locate a disk server.

2) Load the second stage boot code.

The second stage boot code is located in the boot file in the Corvus volume. It must perform the remainder of the boot and logon procedures:

3) Load the rest of the boot code (if necessary).

4) Prompt for user name and password. Read the NETWORK.USER table and validate the user name and password. Obtain the home disk server name and operating system type from the user's entry in the NETWORK.USER table.

5) Locate the user's home disk server.

6) Update the Active User table with the user information.

7) Locate the user's boot volume and load the operating system.

8) Build the user's mount table.

Once all this has been accomplished, the boot code jumps to the operating system, allowing it to complete the boot.

It is also possible that a computer will boot from some local device, and logon to the network after booting. This logon does not need to load any boot code or operating system code (steps 2, 3, and 7 above); however, it must perform all the other steps.

Detailed description of boot process

Each of the steps 1 through 8 listed above is now described in more detail. A text explanation is given, followed by a box containing pseudo-code that performs the task being discussed.

The format of pseudo-code used is similar to that used in the *Corvus Mass Storage Systems GTI* in describing disk drivers. It is assumed that you are familiar with the Omninet command vectors and Omninet Protocols described in that manual.

The declarations used by the pseudo-code are given on the next page.

```
; Sample data structures
;
; First the data structure is declared, then a list of offsets
; into the structure are declared.
;
;
; Transporter command vector (see Omninet GTI, pgs. 32,33)
;   It is not necessary to have more than one command record,
;   although it is sometimes more convenient to use separate
;   records which are preinitialized as Send and Setup receive
;   commands.

TCmd    .BYTE 0          ; OpCode - command code
        .BYTE 0          ; ResAdr  - high order byte of result address
        .WORD 0          ;         - low order word of result address
        .BYTE 0          ; Sock - socket number
        .BYTE 0          ; DatAdr  - high order byte of data address
        .WORD 0          ;         - low order word of data address
        .WORD 0          ; DataLen - data length
        .BYTE 0          ; CrtlLen - user control length
        .BYTE 0FFh       ; Dest - destination host number
                         ; offsets
OpCode .EQU 0            ; offset to OpCode
ResAdr .EQU 1            ; offset to ResAdr
Sock   .EQU 4            ; offset to socket number
DatAdr .EQU 5            ; offset to DatAdr
DataLen.EQU 8            ; offset to data length
CrtlLen.EQU 10           ; offset to user control length
Dest   .EQU 11           ; offset to destination host number (Send only)
Destecho.EQU 4           ; offset to destination host number (Echo only)

; Result record definitions
                         ; Send result record
SndRes .BYTE 0           ; transporter return code
       .BYTE 0           ; unused
       .WORD 0           ; unused
                         ; offsets
RCode  .EQU 0            ; offset to transporter return code

                         ; Receive result record
Rcv80  .BYTE 0           ; transporter return code
       .BYTE 0           ; Src - source host number
       .WORD 0           ; unused
                         ; offsets
Src    .EQU 1            ; offset to Src
```

```
; Sample data structures (cont.)
;
; Data area buffers
;
                          ; Send buffer
MsgBuf .WORD 0            ; ProtoID
       .WORD 0            ; MsgTyp
       .WORD 0            ; Source
       .WORD 0            ; DevType
       .WORD 0            ; Name (10 bytes)
       .WORD 0
       .WORD 0
       .WORD 0
       .WORD 0
                          ; offsets
ProtoID.EQU 0             ; offset to ProtoID
MsgTyp .EQU 2             ; offset to MsgTyp
Source .EQU 4             ; offset to Source
DevType.EQU 6             ; offset to DevType
Name   .EQU 8             ; offset to Name

                          ; Receive buffer
S80Msg .WORD 0            ; ProtoID
       .WORD 0            ; MsgTyp
       .WORD 0            ; Source
       .WORD 0            ; DevType
       .WORD 0            ; Name (10 bytes)
       .WORD 0
       .WORD 0
       .WORD 0
       .WORD 0

; local variables

TimeOut.WORD 0            ; used to control disk server wait loop

MyAddr .BYTE 0            ; this computer's transporter address

Bootsrv.BYTE 0            ; transporter address of boot server
```

1. Locate a disk server:

   First, you should make sure that there are no other nodes with your address on the network. This is accomplished by issuing an Omninet Echo command to your address. If you get a response, then report an error.

   Send a WhoamI command to get your address:

```
+-------------------------------------------+
|TCmd+Opcode  <- 01h (WhoamI command code)  |
|TCmd+ResAdr  <- address of SndRes          |
|                                           |
|SndRes+Rcode <- FFh (initialize result code)|
+-------------------------------------------+
```

   Wait for result code to change. Store result in MyAddr:

```
+-------------------------------------------+
|MyAddr       <- SndRes+RCode               |
+-------------------------------------------+
```

   Send an Echo command:

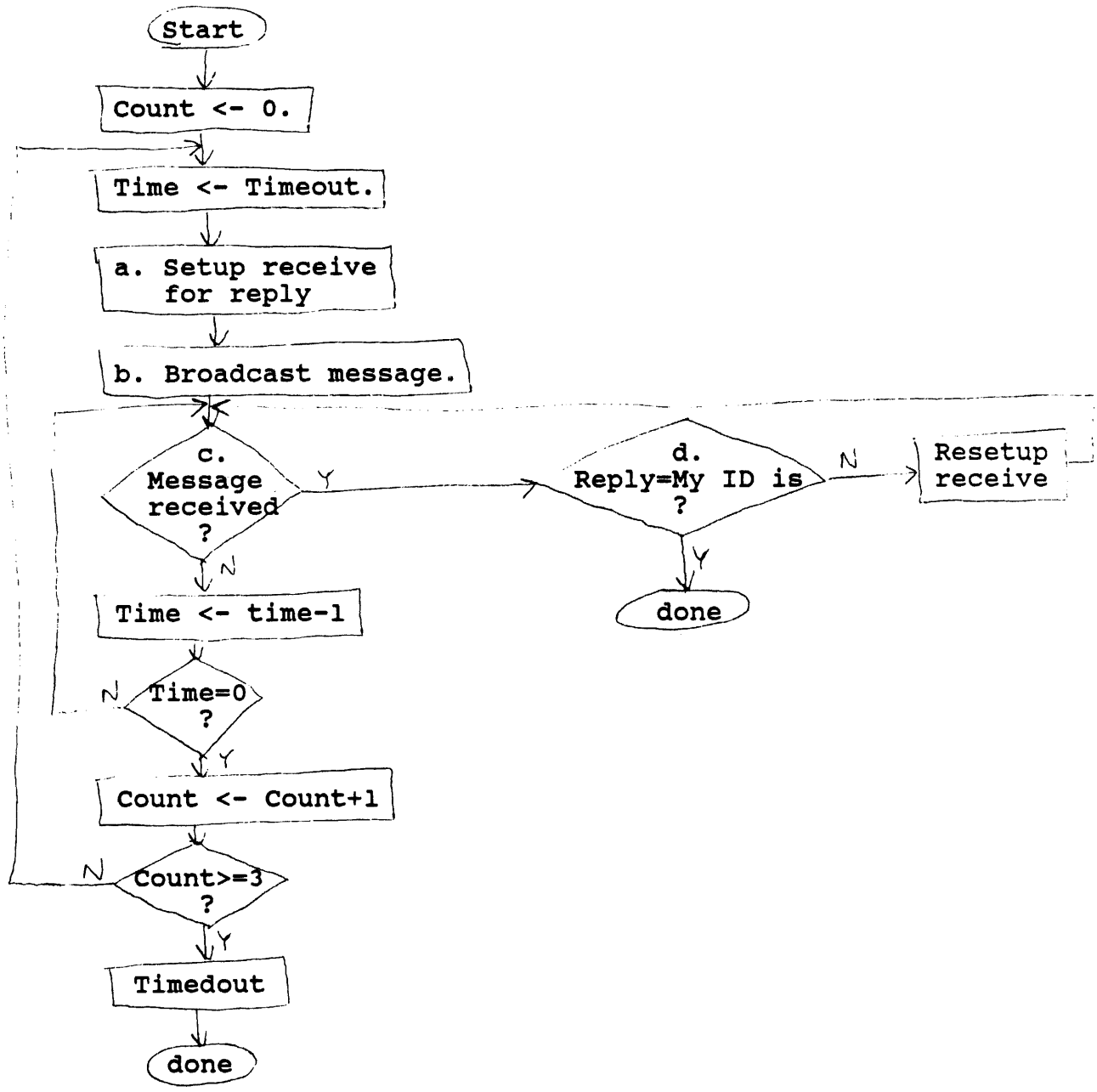
```
+-------------------------------------------+
|TCmd+Opcode  <- 02h (Echo command code)    |
|TCmd+ResAdr  <- address of SndRes          |
|TCmd+Destecho<- MyAddr                     |
|                                           |
|SndRes+Rcode <- FFh (initialize result code)|
+-------------------------------------------+
```

   Check result:

```
+-------------------------------------------+
|If SndRes+RCode = C0h THEN error.          |
+-------------------------------------------+
```

   Once you have made sure that no other transporters with your address are present on the network, you can proceed to to look for a disk server. This procedure involves using the Name lookup protocol. The procedure is summarized below, and is shown in detail in the flowchart in figure 4.3.

   Broadcast 'Who are you, any disk server' command. If no response in 5 seconds, try again; retry 3 times. If still no response, broadcast illegal command (Old Disk server Protocol). If no response in 5 seconds, try again; retry 3 times. If still no response, report error.



Note: the lower case letters in some of the boxes refer to descriptions on the next pages.

Figure 4.3: Flowchart of find a disk server

a. EndRecv on socket 80h.

```
+-----------------------------------------------+
|TCmd+Opcode   <- 10h (EndRecv command code)    |
|TCmd+ResAdr   <- address of SndRes             |
|TCmd+Sock     <- 80h                           |
|                                               |
|SndRes+Rcode  <- FFh (initialize result code)  |
+-----------------------------------------------+
```

Setuprecv on socket 80h. The expected response is a 'My ID is' message.

```
+-----------------------------------------------+
|TCmd+Opcode   <- F0h          (Setuprecv)      |
|TCmd+ResAdr   <- address of Rcv80              |
|TCmd+Sock     <- 80h                           |
|TCmd+DatAdr   <- S80Msg                        |
|TCmd+DataLen  <- 12h          (18 bytes)       |
|TCmd+CrtlLen  <- 0                             |
|                                               |
|Rcv80+Rcode   <- FFh                           |
+-----------------------------------------------+
```

b. Broadcast a 'Who are you' message

```
+-----------------------------------------------+
|TCmd+Opcode   <- 40h    (Send)                 |
|TCmd+ResAdr   <- address of SndRes             |
|TCmd+Sock     <- 80h                           |
|TCmd+DatAdr   <- address of MsgBuf             |
|TCmd+DataLen  <- 08h    (8 bytes)              |
|TCmd+CrtlLen  <- 0                             |
|TCmd+Dest     <- FFh    (Broadcast)            |
|                                               |
|SndRes+Rcode  <- FFh                           |
|                                               |
|MsgBuf+Pid    <- 01FEh                         |
|MsgBuf+MsgTyp <- 0200h (Who are you)           |
|MsgBuf+Source <- MyAddr                        |
|MsgBuf+DevType<- 01h   (Generic disk server)   |
+-----------------------------------------------+
```

c. Check for message received, when result code goes to 0.

```
+-------------------------------------------------------+
|IF Rcv80+Rcode=0 THEN check message ELSE increment time|
+-------------------------------------------------------+
```

d. If Rcv80+Rcode is 0, then check that it is a My Id Is message that has been received.

```
+-----------------------------------------------+
|IF (S80Msg+Pid=01FEh) AND (S80Msg+MsgTyp=1000h)|
|   THEN Bootsrv <- S80Msg+Source               |
|   ELSE ignore message                         |
+-----------------------------------------------+
```

If no disk server is found, then you must repeat the entire sequence, this time using the Old Disk Server Protocol which broadcasts an illegal command in place of the 'Who are you' command. The expected response is an error code.

a. The Old Disk Server Protocol will receive a Results message on socket B0h, so issue an End Receive followed by a Setup Receive for socket B0h.

```
+-----------------------------------------------+
|TCmd+Opcode   <- 10h          (End receive)    |
|TCmd+ResAdr   <- address of SndRes             |
|TCmd+Sock     <- B0h                           |
|                                               |
|SndRes+Rcode  <- FFh (initialize result code)  |
|                                               |
|                                               |
|TCmd+Opcode   <- F0h          (Setup Receive)  |
|TCmd+ResAdr   <- address of Rcv80              |
|TCmd+Sock     <- B0h                           |
|TCmd+DatAdr   <- S80Msg                        |
|TCmd+DataLen  <- 0                             |
|TCmd+CrtlLen  <- 3                             |
|                                               |
|Rcv80+Rcode   <- FFh                           |
+-----------------------------------------------+
```

Now broadcast the Find a server message:

```
+-------------------------------------------+
|TCmd+Opcode   <- 40h          (Send)       |
|TCmd+ResAdr   <- address of SndRes         |
|TCmd+Sock     <- 80h                       |
|TCmd+DatAdr   <- address of MsgBuf         |
|TCmd+DataLen  <- 08h    (8 bytes)          |
|TCmd+CrtlLen  <- 0                         |
|TCmd+Dest     <- FFh    (Broadcast)        |
|                                           |
|SndRes+Rcode  <- FFh                       |
|                                           |
|MsgBuf+Pid    <- 01FEh                     |
|MsgBuf+2      <- 01h    (Find a server)    |
|MsgBuf+3      <- 0001h                     |
|MsgBuf+5      <- 0000h                     |
|MsgBuf+7      <- FFh    (Illegal command)  |
+-------------------------------------------+
```

If a message is received, then set Bootsrv.

```
+-------------------------------------------+
|Bootsrv <- Rcv80+Src                       |
+-------------------------------------------+
```

If still no disk server is found, then a message should be displayed on the screen, 'No disk server found.'

If a disk server is found, then this server becomes the boot server used in the steps below.

2. Get first block of boot code:

   Send directed Read Boot Block command to the boot server identified in step (1). Jump to the code that is read in.

   If there is an error reading the boot code, then (if possible) put a message to the screen and ask the user to enter the boot server name or number. This situation would occur when the boot code has been placed on some disk servers, but not on all.

For flat cable networks, the ROM simply sends a Read Boot Block command. If there is no response in 5 seconds, an error is reported.

3. Get rest of boot code:

   Send the appropriate number of Read Boot Block commands (the first boot block should tell you how many more to read). Timeout and retry as for normal disk reads.

4. Get user name and password:

   Display the Constellation II message (see Figure 4.6). Prompt for the user name.

   If the user name entered is <esc>, then boot with no volumes mounted, and a blank user name and password. This feature allows a user to boot or logon with the Corvus driver installed, but no access to the network. The user could still execute programs which use the driver to send disk or Omninet commands, such as the disk diagnostic.

   Read the user table (NETWORK.USER) from the boot server. Encrypt the user name. Search the user table for the user; if not found, report an error.

   If user name is found, see if a password is required. If so, prompt for password, encrypt it, and validate. If password is invalid, reprompt.

   Get user's home disk server name and operating system type from the NETWORK.USER table entry. Decrypt the home disk server name.

5. Locate user's home disk server:

   See if user's home server is the one found in step 1. If not, broadcast 'Where are you' to find user's home disk server; retry 3 times. If no response in 5 seconds, report error, and prompt for a server name to boot from. This algorithm is very similar to the one in Figure 4.3, except for box b.

b. Broadcast 'Where are you?', name = home server name

```
+-----------------------------------------------+
|TCmd+Opcode    <- 40h    (Send)                |
|TCmd+ResAdr    <- address of SndRes            |
|TCmd+Sock      <- 80h                          |
|TCmd+DatAdr    <- address of MsgBuf            |
|TCmd+DataLen   <- 12h    (18 bytes)            |
|TCmd+CrtlLen   <- 0                            |
|TCmd+Dest      <- FFh    (Broadcast)           |
|                                               |
|SndRes+Rcode   <- FFh                          |
|                                               |
|MsgBuf+Pid     <- 01FEh                        |
|MsgBuf+MsgTyp  <- 0300h  (Where are you)       |
|MsgBuf+Source  <- MyAddr                       |
|MsgBuf+DevType<- 01h     (Generic disk server) |
|MsgBuf+Name    <-        (home disk server name -|
|                          unencrypted)         |
+-----------------------------------------------+
```

If the home disk server is not found (i.e., timed out waiting for reply), then prompt the user to enter a server name or number:

> Home disk server not found. Enter the name of the home disk server, or press <enter> to use the default disk server [name].

The default disk server is the boot server. You could also accept a number as input, and use the number as the address of the home disk server.

6. Tell network you're here.

Delete any entries for this host number from the Active User Table. This is essentially a clean up step, in case previous users have not logged off. This step has not yet been implemented in any versions of Corvus boot code.

See if there are duplicate users, and warn the user if there are. This check is done to protect users from inadvertently sharing access to volumes. The check is accomplished by issuing the FindActive drive command using the (unencrypted) user name. If an entry is found, and the Omninet address is not the same as MyAddr, then you should send an Omninet Echo command to the Omninet address. If you get a reply, then warn the user that there may be another user with the same name already logged on.

The last step is to tell the network you're here. This is done by broadcasting a Hello message. You should broadcast twice, just to be more sure of the message being received.

```
+-------------------------------------------+
|TCmd+Opcode    <- 40h    (Send)            |
|TCmd+ResAdr    <- address of SndRes        |
|TCmd+Sock      <- 80h                      |
|TCmd+DatAdr    <- address of MsgBuf        |
|TCmd+DataLen   <- 12h    (18 bytes)        |
|TCmd+CrtlLen   <- 0                        |
|TCmd+Dest      <- FFh    (Broadcast)       |
|                                           |
|SndRes+Rcode   <- FFh                      |
|                                           |
|MsgBuf+Pid     <- 01FEh                    |
|MsgBuf+MsgTyp  <- 0000h  (Hello)           |
|MsgBuf+Source  <- MyAddr                   |
|MsgBuf+DevType<-         (your device type - |
|                          see Table A.1)   |
|MsgBuf+Name    <-        (user name -      |
|                          unencrypted)     |
+-------------------------------------------+
```

7. Load the operating system from the user's boot volume.

   This code searches the DRIVE.ACCESS entries on the user's home server for the user's boot volume. If no boot volume is found, an error is reported.

   The operating system is normally located in a file with a special name, and is found by searching the volume directory of the boot volume.

8. Build the user's mount table (see next section).

Building the mount table

Volumes from the user's home server should be mounted first, then volumes from other servers. Only volumes from disk servers (i.e., not Banks) should be mounted, unless booting from the Bank.

The algorithm for building the mount table is given in Figure 4.4.

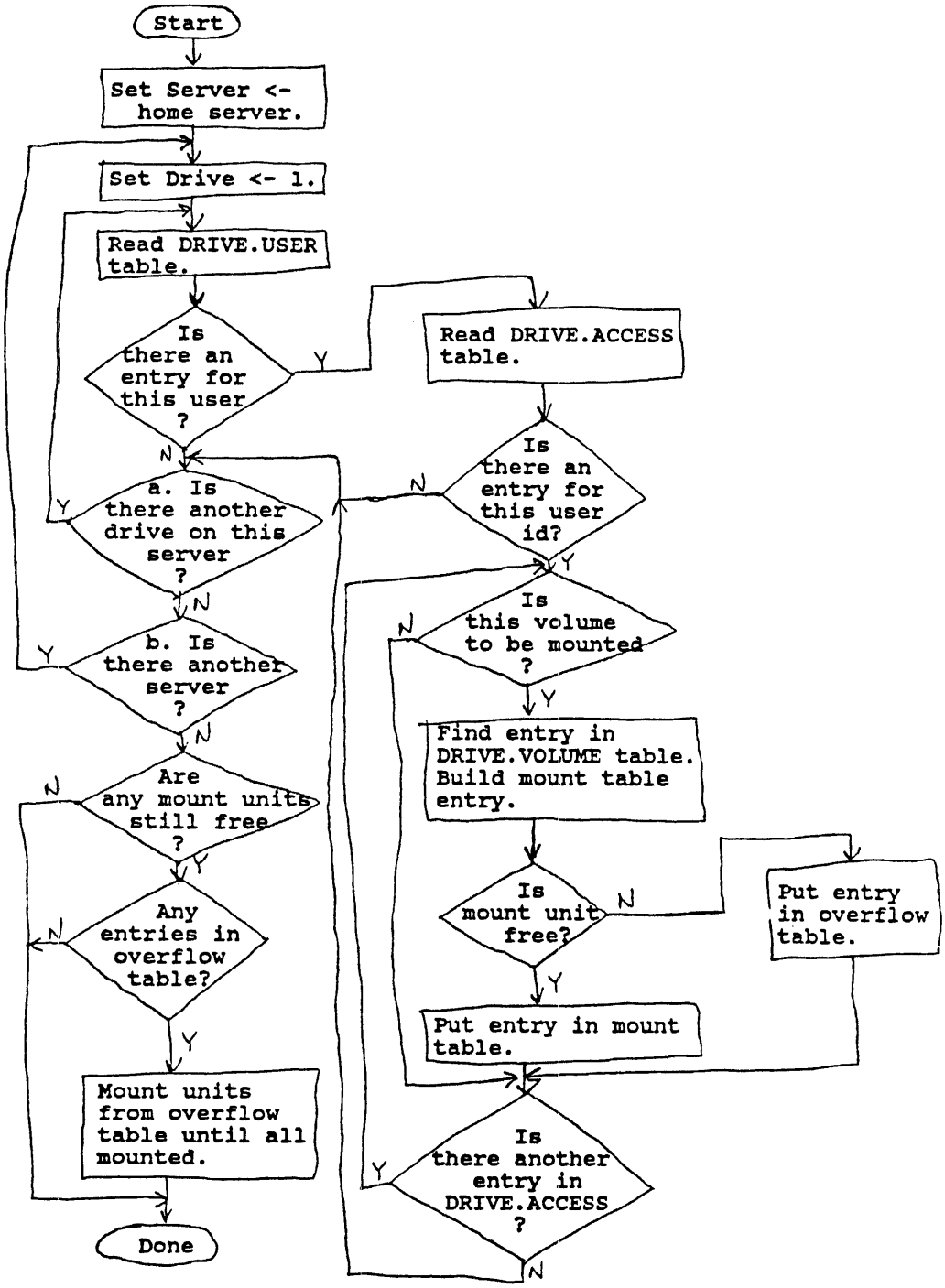


Figure 4.4: Flowchart of building the mount table

a. To see if there is another drive on this server, you should bump the current drive number by 1, and send a Get Drive Parameters command to the new drive number. If you get a reply, and the drive is a physical drive, then continue the loop with the new drive number. If you get a reply, and the drive is a virtual drive, bump the drive number again, and send another Get Drive Parameters command.

b. To see if there are any more servers, you should start by setting the current server address to 0. Send a 'Who are you' message to the current server address. If no reply, or if the reply indicates this is not a disk server, then go on the next address. If the reply indicates a disk server, then check to see if this is the home server. If so, skip it. Exit when the Omninet address reaches 64.

Setting read and write access

There are several levels of read and write protect scattered throughout the Constellation II tables. These are summarized below:

1) **DVRec.Glbaccess** (set by VOLUME MANAGER)
   0 => volume is "invisible". No one can mount this volume.
   1 => volume is "visible".

2) **DVRec.writeable** (set by VOLUME MANAGER)
   0 => volume is read only. No one can write to this volume.
   1 => volume is read/write.

3) **DARec.readonly** (set by ACCESS MANAGER)
   0 => volume is read/write.
   1 => volume is read only. This user cannot write to this volume.

When mounting a volume, either at boot time or with the MOUNT MANAGER, the following algorithm is used:

1) If **DVRec.Glbaccess** = 0, the volume cannot be mounted.

2) If **DVRec.writeable** = 0, the volume is mounted with read only capabilities.

3) If **DARec.readonly** = 1, the volume is mounted with read only capabilities.

4) If none of the above conditions is true, the volume is mounted with read/write capabilities.

GlbAccess and **writeable** are initialized when the volume is created. Both are set to 1. Either can be changed with the Change option of VOLUME MANAGER.

Readonly is set when the ACCESS MANAGER is used to grant access to a volume. It can be changed with the Change option of ACCESS MANAGER.

Special versions of boot code

There are two additional functions which the boot code can perform. One is **single-user boot**, and the other is **no logon boot**.

Single-user boot is a special version of the boot code which allows only one user on the network at a time. This code reads the Active User Table before prompting for the user name. For each entry in the Active User Table which has a host type signifying workstation, the boot code sends an Echo command to the workstation's address. If there is a response, the boot code assumes that another user is currently logged-on, and it refuses to boot the new user.

No logon boot is also a special version of the boot code. With this version, the boot code first looks for an entry in the Network User table which has a value equal to its host transporter address in the **Hoststa** field. If such an entry is found, then the user name is taken from this entry, instead of being obtained from a prompt. If no entry is found with the current transporter address, then the boot code prompts for a user name. Regardless of how the user name is obtained, the boot code then proceeds, as described previously, to find the home server, the boot volume, and build the mount table.

No logon boot is used on networks which are configured such that certain stations always perform dedicated functions. For example, you may have an installation in which you wish to dedicate one computer to executing your accounting package. That computer can be assigned a particular Omninet address, and an entry made in the Network User table for that address. Then whenever that computer is booted, it will have a fixed user name, and that user can be configured to run only the accounting package.

Message	How determined
Duplicate host number.	Use Echo command
No disk server found.	Broadcast of 'Who are you?' or illegal command.
User not found.	Search NETWORK.USER table
User's operating system type not supported.	From NETWORK.USER table and boot code
Incorrect password.	From NETWORK.USER table
Home server not found.	From NETWORK.USER table and 'Where are you?' command
Boot volume not found.	From DRIVE.USER table, DRIVE.ACCESS table
Unable to load operating system.	Directory of boot volume
Disk read error.	Drive off-line, invalid block address.

Figure 4.5: Summary of error conditions during boot

```
        *
          *
                C O R V U S   S Y S T E M S
             *          CONSTELLATION II
           *    *       V x.x  (yy/zz)

Please enter your name:
Please enter your password:
<<Error messages appear in this line...>>
```

Figure 4.6a: Screen formats for boot messages (old logo)

```
             *
           * *
        C  O  R  V  U  S
        CONSTELLATION II

Please enter your name:
Please enter your password:
<<Error messages appear in this line...>>
          Version x.xx     Boot server:  zz     Station:  yy
```

Figure 4.6b: Screen formats for boot messages (new logo)

Please enter your name:

Please enter your password:

<<Error messages appear in this line...>>

Figure 4.6c: Screen formats for boot messages (new logo)

Notes for figures 4.6a through 4.6c:

These are three screen formats currently used by different versions of Corvus boot code. You should always try to include the following information on the boot screen:

x.x:	the version number of the boot code.
yy:	this computer's Omninet address.
zz:	server number of boot server.

The boot code should be outputting some sort of message or heartbeat (dots across screen) at least every 2 seconds.

Chapter 5: User utilities

Mount manager

The Mount Manager is a utility which a user can run to change which volumes are currently on-line. Functions included are listed below:

Mount a volume, specifying volume name, unit number, and write protection
Unmount a volume, specifying volume name or unit number
Display the currently mounted volumes
Display the accessible volumes (i.e., those that can be mounted)
Display user name and transporter address

In order to provide these functions, the Mount Manager must read the Constellation II disk tables, including the Network User table, the Drive User table, the Drive Volume table, and the Drive Access table. The Mount Manager must also use the Read Mount Table and Write Mount Table entry points of the operating system driver.

Mount a volume

1) Validate the unit number.

2) Search all Drive Volume Tables for the named volume. Note that since the Drive Volume Table is sorted by volume address, this must be a sequential search through the entire table.

3) Find the Drive User entry for the user.

4) Using the volume address from the Drive Volume Table, and the user id from the Drive User Table, find the Drive Access entry for this user and volume.

5) Select the appropriate write protection, as described below under the heading *Setting read and write access*.

5) Call Read Mount Table to get the current mount table; modify the appropriate entry; call Write Mount Table to update the driver with the new mount table.

Unmount a volume

1) Search all Drive Volume Tables for the named volume.

2) Call Read Mount Table to get the current mount table. Search the mount table for the entry whose mount information matches the address of the named volume. Overwrite this entry with zeroes to indicate an unused mount entry.

3) Call Write Mount Table to update the driver.

Display mount information

1) Call Read Mount Table to get the current mount table and user information.

2) Display the appropriate information. For currently mounted volumes, you will have to find the volume names by searching the appropriate Drive Volume tables. For accessible volumes, you will have to search all Drive User tables and appropriate Drive Access and Drive Volume tables.

Setting read and write access

There are several levels of read/write protect scattered throughout the Constellation II tables. These are listed below:

1) DVRec.Glbaccess (set by VOLUME MANAGER)
   0 => volume is invisible. No one can mount this volume.
   1 => volume is visible.

2) DVRec.writeable (set by VOLUME MANAGER)
   0 => volume is read only. No one can write to this volume.
   1 => volume is read/write.

3) DARec.readonly (set by ACCESS MANAGER)
   0 => volume is read/write.
   1 => volume is read only. This user cannot write to this volume.

When mounting a volume, either at boot time or with the MOUNT MANAGER, the following algorithm is used:

1) If DVRec.Glbaccess = 0, the volume cannot be mounted.

2) If DVRec.writeable = 0, the volume is mounted with read only capabilities.

3) If DARec.readonly = 1, the volume is mounted with read only capabilities.

4) If none of the above conditions is true, the volume is mounted with read/write capabilities.

GlbAccess and Writeable are initialized when the volume is created. Both are set to 1. Either can be changed with the Change option of VOLUME MANAGER.

Readonly is set when the ACCESS MANAGER is used to grant access to a volume. It can be changed with the Change option of ACCESS MANAGER.

Library routines

(This section lists the ideal set of library routines. See Appendix B for a list of the routines provided by Corvus for a particular operating system.)

Library routines should provide the following functions:

Semaphore operations

See *Corvus Mass Storage Systems GTI*, Chapter 5.

Pipes operations

See *Corvus Mass Storage Systems GTI*, Chapter 6.

Mount table operations

```
FUNCTION UnitAdr( unit: INTEGER;
                  VAR voladr: CDAddr;
                  VAR read_only BOOLEAN): BOOLEAN;
```

Returns the location and read-write information for the volume mounted on the specified unit. Definition of CDAddr is:

```
TYPE CDAddr = RECORD
   Slotno:  INTEGER;
   serverno: INTEGER;
   driveno:  INTEGER;
   Addr:     LONGINT;
   END;
```

```
FUNCTION GetMntInfo( VAR MntInfo: MntStuff ): BOOLEAN;
```

Returns the mount information as follows:

```
TYPE MntStuff = RECORD
   username:  string[10]; { unencrypted }
   userpass:  string[8];  { encrypted }
   homesrvr:  INTEGER;
   homeslot:  INTEGER;
   cardtype:  INTEGER; (1=flat cable, 2=OmniNet)
   useraddr:  INTEGER; (user's transporter address )
   END;
```

```
FUNCTION UnMount( unit: INTEGER ): BOOLEAN;
```

Unmounts the volume on the unit specified;
returns TRUE if successful.

```
FUNCTION Mount( volume: String; unit: INTEGER ): BOOLEAN;
```

Mounts the volume named, if user has access;
returns TRUE if successful.

Some convention for specifying server and drive number must be used. Either the volume name string must include a server and drive name, or another function must be called to specify server and drive.

Send Corvus disk command

See *Corvus Mass Storage Systems GTI*, Chapter 4.

Spooler and despooler

See *Corvus Mass Storage Systems GTI*, Chapter 6.

Spool driver

For those operating systems which support attachable drivers, it is possible to write a spool driver which will redirect output sent to the printer to the Corvus pipes area. The despooler will then print the data on the shared network printer. This section summarizes the functions provided by a spool driver.

There are usually two entry points to a printer driver: device initialization and character write. Device initialization simply sets up all the default parameters used by the driver. The write call does all the work:

1) If a pipe is not currently open, then open a pipe with the default name. Write the preamble block. Set the pipe open flag, and initialize the output buffer.

2) Put the character in the output buffer.

3) If the output buffer is full (512 bytes), then write the buffer to the disk with a Pipe Write command, and reinitialize the output buffer.

The main problem encountered in writing a spooler driver is figuring out when to close the pipe. Corvus has used two approaches which both seem to be acceptable in practice: one is to use a timeout (that is, if no output has occurred in a fixed amount of time, then flush the buffer and close the pipe); the second is to monitor the output for a defined string; when that string is seen, flush the buffer and close the pipe.

All of the parameters used by the spooler driver are stored in a table which can be modified through an entry point in the driver. Parameters and suggested defaults are listed below:

1) pipe name (default "PRINTER");
2) disk server address of disk containing pipes area (default is user's home disk server);
3) contents of preamble block message (default is user name concatenated with date and time);
4) timeout value to use in closing the pipe (default is 15 seconds);
5) string to look for in closing a pipe (default "::").

Chapter 6: Management utilities

Overview

The Constellation II management utilities consist of several separate programs which are controlled from a single management program known as the Constellation II manager (CC.C2MGR on Concept; C2MGR.CODE on p-system).

The Constellation II manager program presents the user with several sets of menus; when a user chooses an option, either another menu is provided, or a second program is loaded. When execution of the second program is completed, control is returned to the manager program.

On Concept, control is transferred to other programs with the CALL intrinsic. Under CCOS, when the CALLed program terminates, control is returned to the statement following the CALL.

On p-system, control is transferred to other programs with the CHAIN procedure. When an option is selected which requires another program to execute, the CHAIN procedure is called twice, once to CHAIN to the new program, and once to CHAIN back to C2MGR. Then an EXIT(program) statement is executed, which transfers control to the new program. When the new program terminates, control is transferred to the next statement in the CHAIN, which is C2MGR.

(Note to implementers: All this chaining under p-system is VERY slow. There are two alternatives to using CHAIN: one is to use the CALLER unit available from NCI, and the other is to make most of the programs into units rather than programs. Both of these alternatives have been tried, and appear to work. The only problem is in phasing in the new versions of the utilities, since the old CHAIN versions will not work with versions using either CALLER or units.)

The main menu of C2MGR currently contains these options:

Menu option	Action
Drive Management	Display Drive Management menu
Backup Utilities	Display Backup menu
Maintenance Utilities	Display Maintenance menu
Utility Server	Display Utility Server menu
Transfer Manager	Invoke Transfer Manager program
Initialize Drive	Invoke System Generation program
List Drives	Update and display Active User Table
Exit	Terminate program

Each sub-menu is described below:

Drive Management

Menu option	Action
User/Device Manager	Invoke User Manager program
Volume Manager	Invoke Volume Manager program
Access Manager	Invoke Access Manager program
Boot Manager	Invoke Boot Manager program
Select Drive	Prompt for drive name and password
List Drives	Update and display Active User Table
Exit	return to outer menu

Backup Utilities

Menu option	Action
Mirror	Invoke Mirror program
Archival Storage	Invoke Mirror program, remote option
Remote Backup	Invoke Net Backup program
Select Drive	same as above
List Drives	same as above
Exit	return to outer menu

Maintenance Utilities

Menu option	Action
Parameter Manager	Invoke Parameter Manager program
Disk Diagnostic	Invoke Rev B/H Diagnostic program
OmniDrive Diagnostic	Invoke Omnidrive Diagnostic program
Bank Diagnostic	Invoke Bank Diagnostic program
Omninet Test	Invoke Omninet Test program
Fixit (hidden option)	Invoke Const II Utility program
Update Utilities	Invoke Utility Update program
Select Drive	same as above
List Drives	same as above
Exit	return to outer menu

Utility Server Manager

Menu option	Action
Printer Service	Invoke Print Server Manager program
Time Service	Invoke Time Server Manager program
Exit	return to outer menu

Each program is described in detail later in this chapter, except for the Utility Server Manager options which are covered in Chapter 7.

Menu file

The menus actually displayed and the actions taken for each option are controlled by a text file called C2.MENU.TEXT. Each line in this file has the following format:

Column 1-2:	Menu number on which this option is displayed.
Column 4-5:	Relative order of this option in menu.
Column 6:	"b" indicates to print a blank line before printing this option. "q" indicates that this is a hidden option, and the option label should not be displayed.
Column 7-34:	Option label. The first character (column 7) must be the character that the user presses to select this option.
Column 35-36:	Softkey number.
Column 37-44:	Softkey label.
Column 45-46:	Softkey number.
Column 47-54:	Softkey label.
Column 56-57:	Number of the menu to be displayed if this option is selected.
Column 58-71:	Program to be invoked if this option is selected.
Column 75:	"n" indicates that the current disk server information should NOT be displayed on the screen.
Column 76:	"Y" indicates that the selected drive information should be passed to the invoked program; "-" indicates that the selected drive information should NOT be passed to the invoked program. Selected drive information is discussed below.
Column 77-86:	Additional parameters to be passed to program.
Column 90:	"/" to indicate end-of-line.

Here is a portion of a C2.MENU.TEXT file, with column numbers indicated. It shows the file contents for the main menu and for the Drive Management sub-menu.

```
         1    1    2    2    3    3    4    4    5    5    6    6  7
1...5....0....5....0....5....0....5....0....5....0....5....0....5//5..
01 10bD - Drive Management          10Drive   00Manager  02x           -
01 20 B - Backup Utilities                    01Backup   03x           -
01 30 M - Maintenance Utilities               02Maint    04x           -
01 40qC - Configure System                                05x         n-
01 50bU - Utility Server Manager    13Utility 03Server   06x         n-
01 55 T - Transfer Manager          14Transfer04Manager  00XFRMGR      -
01 60bI - Initialize Drive          15Init    05Drive    00SYSGEN      -
01 80 L - List Drives               17List    07Drives   00x           -
01 85 H - Help                                08Help     00x           -
01 90 E - Exit                                19Exit     00x           -
02 10bU - User/Device Manager       10User    00Manager  00USERMGR     Y
02 20 V - Volume Manager            11Volume  01Manager  00DRIVEMGR    Y
02 30 A - Access Manager            12Access  02Manager  00ACCMGR      Y
02 40bB - Boot Manager              14Boot    04Manager  00BOOTMGR     Y
02 70bS - Select Drive              15Select  05Drive    00x           -
02 80 L - List Drives               17List    07Drives   00x           -
02 90 E - Exit                                09Exit     01x           -D
```

On startup, C2MGR reads the entire C2.MENU.TEXT file, and builds an internal data structure. This structure allows a maximum of 10 options per menu, and a maximum of 10 menus.

Menu 1 is displayed, and the user is prompted to enter a character. The character is compared to each of the possible choices for the current menu (column 7); if it matches, then one of the following actions is taken:

(a) Display new menu (menu number > 0).
(b) Execute predefined option (menu number=0, program name="x")
(c) Invoke new program (menu number=0, program name<>"x")

The predefined options include:

L - List Drives: Updates the Active User Table using the algorithm described in Figure 2.13; then displays, for each disk server entry, information on each drive connected to the server.

S - Select Drive: Prompts user for a server name, drive name, and drive password. This information will be passed as a parameter to any invoked programs.

E - Exit: Exits the program.

H - Help: Displays the Help file, C2.HELP.TEXT.

If a new program is to be invoked, then a parameter string is built, consisting of the selected drive information, if required (column 76="Y"), followed by any additional parameters indicated in columns 77-86. If selected drive information is required, and no drive has been selected, then the user is prompted to select a drive before the new program is invoked.

For p-system, the selected drive information is also included on the second CHAIN, which will reexecute C2MGR. A second parameter specified on this CHAIN tells the C2MGR program which menu it should display on start-up. A "D" tells it to display the Drive Management menu; an "X" tells it to display the main menu. This information is passed to C2MGR so that the effect of the CHAIN is similar to that obtained using CALL on CCOS: the user is returned to the point in C2MGR where the second program was invoked.

The selected drive information is passed in the following format:

3 digits indicating slot number (use 001 for machines with only one slot).
3 digits of zero.
3 digits indicating server address.
3 digits indicating drive number.
8 character drive password (unencrypted).

Each field is separated by a "/". For instance, the string

001/000/000/001/S

indicates slot 1, server 0, drive 1, with password "S".

Note that while the user selects the drive by indicating the server and drive names, the corresponding addresses are passed to the program.

Programs may be invoked from the command line by entering the appropriate parameter string. For instance, the following command executes the User Manager program on CCOS:

```
USERMGR 005/000/000/001/S
```

The following command executes the User Manager program on p-system:

```
XUSERMGR I="001/000/000/001/S,"
```

Options file

Several of the Constellation II programs present a list of alternatives when prompting for specific input. The contents of these lists are read from an options file called C2.MISC.TEXT. Currently, the following options are supported:

Device type, from Table A.1 (generic types only)
Volume type, from Table A.2
Operating system type, from Table A.3
Boot type, from Table A.4

The options file contains a text description of the option and its corresponding integer indicator. It usually contains a subset of the information included in Tables A.1 through A.4. On program start-up, an internal data structure is built reflecting the contents of this file; the structure allows 18 options per table. Whenever the user is prompted to specify a volume type, for instance, the appropriate text descriptions are displayed, and the user may enter a substring which maps uniquely to one of the choices. The corresponding integer is then used in the Drive Volume table entry. Similarly, whenever a Drive Volume table entry is displayed, the integer from the entry is mapped to the appropriate text description, and the text is displayed.

Each line from the options file has the following format:

Column 1: Table indicator:
- D - device type
- V - volume type
- O - operating system type
- B - boot type

1 or more blanks, and then an integer value

1 or more blanks, and then either an "x" or a "." If its an "x", then the next 20 characters are a text description.

For operating system type, additional information is included, which indicates what the drive specifiers are for the particular operating system. This information includes two integers, as described below:

First integer, preceded by a ".", is drive specifier type:
1: Letters "A", "B", "C", ...
2: p-system units: 4, 5, 9, 10, ...
3: Units numbered from 0: 0, 1, 2, ...
4: Apple DOS slot and drive numbers: 01, 02, 11, 12, ...

Second integer: maximum number of unit specifiers

Then there are 1 or more blanks, followed by an "x", followed by a text description.

Here is a sample of the C2.MISC.TEXT file:

```
B      4       xConcept
B      9       xIBMPC
B     18       xPrintsrv
H      6       xBank
H      2       xDiskServer
H      3       xPrintServer
H      1       xUserWorkstation
O     16       .2        29      xCCOS
O      4       .3        10      xMSDOS
O     17       .2        12      xUCSDIV.0
V     14       xCCOS
V      3       xMSDOS
V      1       xUCSD
```

Update file (install new versions of utilities)

For CCOS, new versions of the utilities are installed either manually, or by executing an EXEC file which invokes the filer.

For p-system, Corvus does not license the Filer, so we have provided our own Update program to copy and remove files. This program is driven by a text file called SYSTEM.CONTENT. Each line in the SYSTEM.CONTENT file is a command; the commands and their format are listed below:

Command formats (showing columns):

```
         1         2         3         4         5    5    6
1...5....0.........0.........0.........0.........0....5....0

C   <Fname1>                 <Fname2>                   /

D   <Fname>                  /

K   <Volname>                /

L   <Volname>                /

M   <Fname1>                 <Fname2>                   /

X   <Fname>                  /

N   <Volname>                /
```

Command definitions:

C: Copy file; <Fname1> = source; <Fname2> = destination

D: Delete file <Fname>

K: Krunch volume <Volname>

L: List volume <Volname>

M: Merge <Fname1> = <Fname1>+<Fname2>

X: Execute program <Fname>

N: Prompt user to insert floppy called <Volname>

Commands can come in any order, except the last command in a file must be either an "N" command or an "X" command. If there are no more floppies in the set, then <Volname> is blank.

Blank lines between commands are allowed. There must always be a character in (or past) the column marked with '/'; this character marks the end of the command.

<Fname> is a fully qualified file name, i.e., it can include a volume name. <Volname> is a volume name, including the '/' or ':' required by Concept and UCSD respectively.

The Merge command is currently unimplemented, but was intended to be used to merge new lines into the menu file or the options file. It assumes that both files are sorted, and merges the new file <Fname2> into the old one <Fname1>, discarding duplicate lines.

The Copy command compares the dates of the source and destination files; if the source file is older than the destination file, the copy is not performed. If there is no room to copy a file, the volume is krunched, and the user is prompted to reboot. The user can then execute the update again.

The Update Utility also states that a copy command may have a * in column 2. This feature was used to indicate that the program should reboot after the copy and was used for replacing the UPDATE program itself, or other critical programs. However, since the Update Utility was changed to copy a source file if the source date was equal to the destination date, this "*" feature should no longer be used, since it will prevent the update from ever completing successfully.

Here is a listing of a SYSTEM.CONTENT file:

```
 ...5....0.........0.........0.........0.........0....5....0
C   CORMS22:ddiag.code       IBMSYS:DDIAG.CODE          /
C   CORMS22:DIAG.DATA        IBMSYS:DIAG.DATA           /
C   CORMS22:CF18.4ap.DATA    IBMSYS:CF18.4AP.DATA       /
C   CORMS22:OTEST.CODE       IBMSYS:OTEST.CODE          /
C   CORMS22:odiag.code       IBMSYS:ODIAG.CODE          /
C   CORMS22:OD.DIAG.DATA     IBMSYS:OD.DIAG.DATA        /
C   CORMS22:FODR1.6.DATA     IBMSYS:FODR1.6.DATA        /
N   CORMS23:                                            /
```

Program descriptions

The next few pages describe each of the current Constellation II management programs. Each description gives the name of the program code file for Concept and p-system, and a brief summary of what the program is used for. Then each option is described by listing the internal steps performed by the program.

	Concept	p-system
User Manager	USERMGR	USERMGR.CODE

Used to add, delete, and change users. Updates entries on the first drive of each disk server on the network.

Add user..... Prompt for user name, password, home server, and OS type. Build a NETWORK.USER entry. Add the entry to the selected drive. Check all other disk servers on network; if entry does not exist, add it; if entry does exist, replace it.

Remove user.. Prompt for user name. Find the NETWORK.USER entry on the current server, and display its contents. For each drive on the current server, find the DRIVE.USER entry for the specified user; if found, delete all DRIVE.ACCESS entries with the user's **userid**. Remove the NETWORK.USER entry. Repeat for each disk server on the network.

Change user.. Prompt for user name. Find the NETWORK.USER entry, and display its contents. Prompt for new password, home server, and OS type. Replace the entry with the new information. Check all other disk servers on network; if entry exists, replace it; if entry does not exist, add it.

List users... Dislay each entry in the current NETWORK.USER table.

	Concept	p-system
Volume Manager	DRIVEMGR	DRIVEMGR.CODE

Used to add, delete, and change volumes. Works on selected drive only.

Add volume... Build a table of free spaces on drive. Prompt for and validate volume name, length, starting location, volume type. Build a DRIVE.VOLUME table entry, with RW access; add the entry. Format the volume. Build a DRIVE.ACCESS table entry for **userid** 1 and the new volume; add the entry.

Remove volume Prompt for volume name. Find the DRIVE.VOLUME entry, and display its contents. Remove all DRIVE.ACCESS entries with that volume's address. Remove the DRIVE.VOLUME entry.

Change volume Prompt for volume name. Find the DRIVE.VOLUME entry, and display its contents. Prompt for new name, RW protection. Change the entry. Note that nothing in the volume itself is modified.

List volumes. Display each entry in the current DRIVE.VOLUME table.

Extended list Prompt for a volume type. Display each entry with that volume type in the current DRIVE.VOLUME table, along with pertinent operating system dependent information extracted from the volume itself.

	Concept	p-system
Access Manager	ACCMGR	ACCMGR.CODE

Used to specify which users have access to which volumes, and to set default boot conditions. Works on one drive at a time.

Next user.... Prompt for user name. Find the NETWORK.USER entry. Find the DRIVE.USER entry; if not there, add one.

Grant access. Prompt for volume name. Find the DRIVE.VOLUME entry. Prompt user for additional information. Build a DRIVE.ACCESS entry; add the entry.

Remove access Prompt for volume name. Find the DRIVE.ACCESS entry; delete it.

Change access Prompt for volume name. Find the DRIVE.ACCESS entry. Prompt user for new information; replace DRIVE.ACCESS entry.

List access.. Find all DRIVE.ACCESS entries. For each entry, find the matching DRIVE.VOLUME entry. Print the appropriate information.

Help..... Display Help options:

View volume For each DRIVE.VOLUME entry, print the volume name. Find the matching DRIVE.ACCESS entry; if one exists, print an "x" indicating access for this user.

User list.. For each NETWORK.USER entry, print the user name. Find the matching DRIVE.USER entry; if one exists, print the **userid** number.

	Concept	p-system
Boot Manager	BOOTMGR	BOOTMGR.CODE

Used to add, delete, and display the table of boot information used by the boot command. Updates information on the first drive on each disk server on a network.

Add file..... Copy a boot file to the Corvus volume. Update the appropriate entry in the SYSTEM.BOOT table to point to the new boot file. Repeat for all servers on network.

Remove file.. Delete the appropriate file from the Corvus volume. Update the appropriate entry in the SYSTEM.BOOT table to FFh. Repeat for all servers on network.

List files... Print each non-empty entry from the SYSTEM.BOOT table. Print the computer type associated with that entry. Print the file name from the directory entry with the address indicated in the SYSTEM.BOOT entry.

	Concept	p-system
System Generation	SYSGEN	SYSGEN.CODE

To create the Corvus volume, and to optionally set up for operation system installation. Works mainly on selected drive, but adds users and boot files to the first drive on each disk server on the network.

Init..... Prompt for location and length of Corvus volume. Decide on sizes for SYSTEM.BOOT, NETWORK.USER, DRIVE.USER, DRIVE.VOLUME, and DRIVE.ACCESS tables, based on length of Corvus volume.
Build DRIVE.INFO block; write it. Copy the boot code to the proper location in the Corvus volume. Build a directory for the Corvus volume; write it. Write FFh to all table entries (initial value). Create NETWORK.USER entries for all default users. Create DRIVE.USER entries for all users. Create a DRIVE.VOLUME entry for Corvus volume, and any operating system volumes. Create a DRIVE.ACCESS entry for each default user-volume access.

Modify..... Display additional options:

- Mix..... Find Corvus volume. Add entries for new users, volumes and accesses. Add new boot code.
- Add..... Find an unused area of the drive. Proceed as for Init.
- Upgrade.... Same as for Add.
- Install.... Copy operating system dependent files to appropriate volume.

	Concept	p-system
Mirror	MIRROR	MIRROR.CODE
Archival storage	MIRROR "R"	MIRROR.CODE "R"
Network Backup	MBACKUP	MBACKUP.CODE

Utilities for backing up and restoring data with Corvus Mirror(TM) hardware.

	Concept	p-system
Parameter Manager	PARMGR	PARMGR.CODE

Used to maintain the pipes and semaphore areas of the specified drive.

Pipes..... Display additional options:

- Init..... Find a volume called PIPES. Send a Pipe Area Initialize command using the (starting address+6) and (length-6) of the PIPES volume.
- Clear..... Send a Pipe Status 2 command to get the pipe pointer table. Send a Pipe Area Initialize command using the data received from the Pipe Status command.
- Close..... Send a Pipe Close read command. If error, send a Pipe Close write command.
- Purge..... Send a Pipe Close purge command.
- List..... Send a Pipe Status 1 command to get the pipe name table. Send a Pipe Status 2 command to get the pipe pointer table. Format and print the results.

Semaphores... Display additional options:

- Init..... Send a Semaphore Initialize command.
- Lock..... Send a Semaphore Lock command.
- Unlock..... Send a Semaphore Unlock command.
- Status..... Send a Semaphore Status command. Print the results.

MUX..... Put drive in prep mode. Send a Read Corvus Firmware command. Modify MUX table and parameters as specified by user. Send a Write Corvus Firmware command. Take drive out of prep mode.

	Concept	p-system
Rev B/H Diagnostic	DDIAG	DDIAG.CODE
Omnidrive diagnostic	ODIAG	ODIAG.CODE
Bank diagnostic	BDIAG	BDIAG.CODE
Combined diagnostics	MDIAG	MDIAG.CODE

Utility programs used to format the drive, update firmware, modify drive or tape parameters, etc. MDIAG is a later version which combines the functions of DDIAG, ODIAG, and BDIAG.

	Concept	p-system
Omninet Test	OTEST	OTEST.CODE

Test program used to verify reliable operation of Omninet hardware.

Start..... Prompt for destination node address and number of times to send. Use Omninet Send command to send a fixed message, and wait for a reply. Count number of retries and number of replies. Display results.

Who..... Send Omninet Echo command to every possible node address (0 to 63). Print the node number if you get a response, otherwise print a "."

Additionally, the program uses the Omninet Setup Receive command to set up a socket for receiving messages from other computers which may be running the Omninet Test program. Each time a message is received, a count is incremented, and the socket is set up again.

	Concept	p-system
Const II Utility	FIXIT	FIXIT.CODE

Utility program used to rewrite server and drive name and password, in case you forget it. The program reads the Drive Information Block, displays the old information, prompts for new information, and rewrites the Drive Information Block.

	Concept	p-system
Update Utilities	-	UPDATE.CODE

Reads SYSTEM.CONTENT file from specified floppy and executes commands contained in it. This option is used to install new or updated copies of the Constellation II management programs. This is a batch program, controlled entirely by the SYSTEM.CONTENT file, except when it prompts the user to insert a new floppy.

	Concept	p-system
Transfer Manager	XFRMGR	XFRMGR.CODE

Used primarily to backup drives to Bank, and to restore.

Drive to Image Copy Create a DRIVE.VOLUME entry for the image. Add the entry. Copy all blocks up to the last used block from the drive to the image volume, using absolute block read and write commands.

Image to Drive Copy Locate the image volume. Copy the entire image to the destination drive, starting at block 0, using absolute block read and write commands.

Remove an Image.... Prompt for image name. Find the DRIVE.VOLUME entry, and display its contents. Remove the DRIVE.VOLUME entry.

List Images..... Read DRIVE.VOLUME table and display all image volumes. If specified, read the DRIVE.VOLUME table from selected image, and display all volumes.

Volume Copy..... Prompt for source volume location. Prompt for destination volume location. Locate DRIVE.VOLUME entry for source; create and add DRIVE.VOLUME entry for destination. Use absolute block read and write to copy volume contents.

Media Copy..... Use Get Drive Parameters command to get source drive size and destination drive size. If source drive is Constellation II format, then read DRIVE.VOLUME table to get size of data on source drive. If source size is less than or equal to destination size, then copy all blocks from source to destination, using absolute block read and write commands.

Show Results..... Display contents of specified results file.

Constellation II library routines

Both CCOS and p-system support the concept of units, which are separately compiled procedures and functions. Many of the units used by Constellation II are of general use to software developers. The units used by Constellation II are described in this section.

Error reporting (ERRORS, ERRREPORTS)

These two units are used for all error reporting. Unit Errors contains only constant declarations, and details all of the error messages produced by Constellation II utilities. Unit ErrReports contains the procedures used to print these error messages to the screen.

Table manipulation (CVDEFN, CVTABS, TBLRTNS, CVTUTILS)

These units are used for Constellation II table manipulation. The units Cvdefn and CVtabs contain declarations for all the table structures. Unit Tblrtns contains low-level routines for table search, insertion, deletion, replacement, and listing. Unit CVTUtils contains higher level routines for operations on specific tables. All of the Constellation II programs use CVTUtils to access the tables; CVTUtils in turn calls the routines in TblRtns. The procedures in TblRtns are never called directly, except by CVTUtils.

CVdefn and CVTabs are separate units for CCOS; for p-system, the declarations contained in CVdefn and CVTabs are included in unit CVTUtils instead.

Encryption (C2Util1, CVCRYPT)

All name and password encryption and decryption is handled by these routines. There are also routines in C2Util1 which convert from string data type to the internal Name and Password data types.

For CCOS, there is only one unit, C2Util1, for encryption.

Drive selection (NIUTIL, C2UTIL3, C2U3)

These routines handle the menu functions Select Drive and List Drives.

For CCOS, units NIUtil and C2U3 are used. For p-system, units NIUtil and C2Util3 are used.

Options file (C2UTIL2, C2U2)

These routines handle reading the options file, prompting the user with appropriate options, and converting table information into text.

For CCOS, the unit is called C2U2. For p-system, the unit is called C2Util2.

Volume formatting (FMTERS)

This unit handles all operating system specific volume formatting. Whenever a new volume type is defined, the code to format the volume must be added to this unit.

Console I/O (CCCRTIO, UCCRTIO)

These routines handle all i/o to the screen. CCCrtIO is for CCOS; UCCrtIO is for p-system.

Drive communication (CCDRVIO, UCDRVIO)

These routines are used to send drive commands directly to a drive. In addition to absolute disk reads and writes, these commands include the pipes and semaphores commands, and drive status commands. CCDrvIO is used for CCOS; UCDrvIO is used for p-system.

Omninet operations (CCOTCIO, CCOMNIIO, UCOMNIIO)

These routines are used to issue Omninet commands, such as Send and Setup Receive. Units CCOtcIO and CCOmniIO are used by CCOS; unit CCOtcIO supercedes CCOmniIO; unit UCOmniIO is used by p-system.

Chapter 7: Utility server

This chapter describes the original version of the Utility Server software. This software consists of two pieces: the utility server software which runs on the Utility server hardware, and the management programs which run on some host machine on the network. It is assumed that you have read the *Utility Server Manager's Guide* for your computer.

Utility server software

The Utility server software provides two functions: printer despooling and time stamping. The server loads and executes a program on power-up; this program simply loops, looking for pipes to print, and updating its internal timer variable periodically. The despooler function assumes that any pipes it is to print contain the preamble block described in the *Corvus Mass Storage System GTI*, Chapter 6.

Corvus has chosen to use the Version II p-system as the operating system executing on the Utility server. The Utility server software therefore includes the following files:

File name	Description
P.SERVER.BOOT	Boot code
SYSTEM.COMSRV	p-system interpreter with custom BIOS
SYSTEM.PASCAL	Version II p-system operating system
SYSTEM.MISCINFO	Version II miscinfo file
SYSTEM.STARTUP	Utility server program

The Utility server uses Constellation II boot protocol. That is, the ROM on the utility server broadcasts a Constellation II Read Boot Block command on power up. The Utility Server boot code must be installed on all disk servers on the network, as a boot file in the Corvus volume and with an entry in the System Boot table.

All of the other Utility server files are located in a volume pointed to by the boot code file. This volume is a standard UCSD volume, and must contain all of the files listed above. In addition, the volume contains printer configuration files used by the despooling function. These files consist of an index file P.SERV.TBL, and one or more data files. Each data file has a name of the form P.CONFG.xxxxxxx, where xxxxxxx is a name assigned by the user who set up the configuration file.

The P.SERV.TBL file is one block long, and consists of 64

8-byte entries. Each of the 64 entries maps to the Omninet addresses 0 to 63, and contains the suffix of the configuration file assigned to the Utility server at the corresponding Omninet address. For instance, if entry number 10 contains the characters ".UTIL1 ", then the Utility server located at Omninet address 10 will look for its configuration information in the file named P.CONFG.UTIL1.

Each configuration file contains information for each of the three printers that can be connected to the utility server, in the following format:

P.CONFG.xxxxxxx file
(1 block)

byte	Description	
0	Info for serial printer 1	The format of the printer information is shown below.
39		
40	Info for serial printer 2	
79		
80	Info for parallel printer	
119		
120	Timer information	The format of the timer information is shown below.
143		
144	Unused	
511		

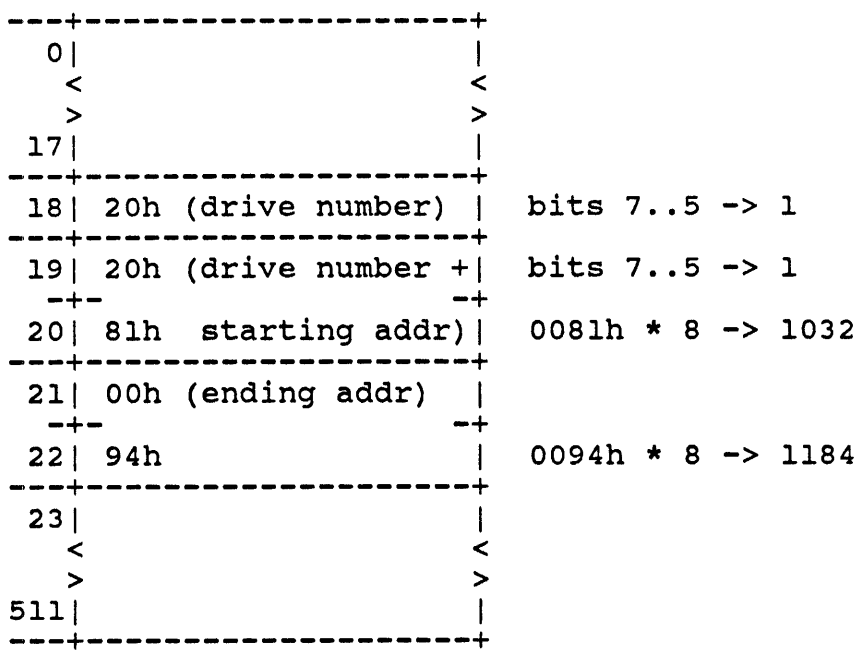
Printer info entry
(40 bytes)

byte	Description		
0	Pipe name (first byte is length; maximum 20 chars)		
20			
21	Unused		
22	Device type	lsb	Device type: 1 -> console 2 -> line printer 3 -> none 4 -> file 5 -> modem
23		msb	
24	Device active flag 0 -> not active	lsb	
25	1 -> active	msb	
26	Line feed flag 0 -> line feeds off	lsb	
27	1 -> line feeds on	msb	
28	Max lines per page	lsb	
29		msb	
30	Tab spacing	lsb	Baud rate: 0 -> 300 baud 1 -> 600 baud 2 -> 1200 baud 3 -> 2400 baud 4 -> 4800 baud 5 -> 9600 baud
31		msb	
32	Baud rate	lsb	
33		msb	
34	Character size 0 -> 7 bits	lsb	Parity: 0 -> parity disabled 1 -> odd 2 -> even
35	1 -> 8 bits	msb	
36	Parity	lsb	
37		msb	Handshake: 0 -> line/cts/inverted 1 -> line/cts/normal 2 -> line/dcd/inverted 3 -> line/dcd/normal 4 -> xon/xoff
38	Handshake	lsb	
39		msb	

Timer information

byte	Description		Further explanation
120	02h		
121	hour - 1st digit		ASCII: 00, 01, ..., 23
122	hour - 2nd digit		
123	Unused		
124	02h		
125	minute - 1st digit		ASCII: 00, 01, ..., 59
126	minute - 2nd digit		
127	Unused		
128	Day of week	lsb	integer 0..6
129		msb	
130	Day	lsb	integer 1..31
131		msb	
132	03h		
133	Month	1st char	ASCII: JAN, FEB, MAR, ...
135		3rd char	
136	04h		
137	Year	1st char	ASCII: 1983, 1984, ...
140		4th char	
141	Unused		
142	Valid flag	lsb	1 -> timer information is valid 0 -> timer information is not valid
143		msb	

The file P.SERVER.BOOT is modified on installation to include the location of the Utility server volume. This information is located in block 0 of the file, in bytes 18 through 22. Included are the drive number on which the volume is located, the starting address of the volume, and the (ending address+1) of the volume. This information is in the old Constellation I format; that is, the upper 3 bits of the msb of the address contain the drive number, and both addresses are divided by 8. For example, if the Utility server volume is located on drive 1, starting at block 1032, and ending at block 1183, then the contents of bytes 18 through 22 would be as shown below:



Note that the disk server address of the Utility server volume is not included. For this reason, the Utility server volume must always be on disk server 0.

Management Utilities

The Management utilities include two programs: Print Server Manager and Time Server Manager.

	Concept	p-system
Print Server Manager	PSMGR	PSMGR.CODE

Used to install the Utility server boot code, and to maintain the printer configuration tables.

Install... Prompt for name of volume containing Utility server files. Read file P.SERVER.BOOT and write volume location information into it. Write out file BOOT.PRINTSRV. Invoke the Boot Manager program so that the user can install the BOOT.PRINTSRV file on all disk servers.

Config.... Prompt for name of configuration file. Prompt for printer port. If configuration file already exists, and there is an entry for the specified port, then display the information. Prompt user for new information. Build printer information entry. Write P.CONFG.xxxxxx file. Prompt user for Utility server address. Update entry in P.SERV.TBL file.

	Concept	p-system
Time Server Manager	TSMGR	TSMGR.CODE

Used to reset the Utility server clock.

Read P.CONFG.xxxxxxx file for specified server.
Update timer information as specified by user.
Write out P.CONFG.xxxxxxx file.

Table A.1
Constellation Device Types

Specific types are indented below their generic type.

Value	Meaning
01	Generic disk device, booting; Corvus disk server
02	Generic Print Server
03	Reserved
04	Mirror Server
05	Bank
06	Omnidrive (generic type = 01)
07-0Fh	Reserved.
10h	Generic disk device, non-booting
11h-1Fh	Reserved for future mass storage devices.
20h-3Fh	Workstations. Workstations are Constellation Boot number plus 20.
20h	Generic Workstation Device Type
21h	Apple II
25h	Corvus Concept
29h	IBM/PC or IBM/XT
2Ah	Xerox 820
2Bh	Zenith H89
2Ch	NEC PC8000
2Dh	Commodore PET
2Eh	Atari 800
2Fh	TRS-80 Model I
30h	TRS-80 Model II
31h	LSI-11
33h	Apple ///
34h	DEC Rainbow
35h	TI Professional
36h	Zenith Z-100
37h	Corvus Concept Plus
38h	Corvus Companion
39h	Apple MacIntosh
3Ah	Sony SMC-7086
40h-5Fh	Reserved for future workstations.

60h-7Fh Operating system types. Operating system types are Constellation operating system number plus 60h.

61h	Apple Pascal
62h	Apple DOS 3.3
63h	UCSD Pascal version 2.x
64h	MS-DOS 1.x
65h	Apple SOS
66h	Apple Pascal Runtime
67h	CP/M 80
68h	RT-11
69h	RSX-11
6Ah	PET DOS
6Bh	NEWDOS (TRS-80 Mod I/III)
6Ch	NEWDOS-80 (TRS-80 Mod I/III)
6Dh	Atari DOS 2.0
6Eh	UNIX System 3
6Fh	CP/M 86
70h	CCOS (Corvus Concept)
71h	Constellation II Pascal IV.x
72h	CP/M 68
73h	NCI p-system
74h	Softech p-system IV.1
75h	Apple ProDOS
76h	Apple MacIntosh
77h	UNIX System 5
78h	Apple II CP/M

80n-8Fh Gateways

80h	Generic gateway
81h	SNA gateway

90h-9Fh Reserved.

A0h-A8h Z80 based utility servers

A0h	Generic Utility Server II server
A1h	Enhanced print service
A2h	Simple pipes bridge

A9h-AFh Reserved for future servers

B0h-FEh Reserved for future use

FFh Any device.

Table A.2
Volume types

Code	Directory type	Mnemonic
0	Invalid	INVALID
1	UCSD	UCSD
2	CP/M	CPM
3	MSDOS	MSDOS
4	Pet	Pet
5	Apple DOS 3.3	DOS3.3
6	Atari DOS 2.0	ATARI
7	RT-11	RT11
8	RSX-11	RSX11
9	NEWDOS	NEWDOS
10	NEWDOS-80	NEW80
11	UNIX	UNIX
12	Apple III SOS	A3SOS
13	unused	
14	Concept OS	CCOS
15	unused	
16	reserved (blocks 0-8)	RESERVED
17	drive image (normal)	IMAGE
18	drive image (invalid)	INVIMAGE
19	drive image (condensed)	CNDIMAGE
20	Softech AFS	AFS
21	Apple PRODOS	PRODOS
22	Macintosh	MAC

Table A.3
Operating System Types

Code	Operating system	Mnemonic
1	Apple Pascal	A2Pascal
6	Apple run-time	A2Runtime
3	UCSD II.0	UCSDII
17	ConstII IV.0	C2IV.0
19	NCI IV.0	NCIIV.0
20	Softech IV.0	SOFTECHIV0
16	Concept OS	CCOS
7	CP/M	CP/M
15	CP/M-86	CP/M-86
18	CP/M-68	CP/M-68
24	A2CP/M	A2CP/M
4	MSDOS	MSDOS
2	Apple DOS 3.3	A2DOS3.3
5	Apple SOS	A3SOS
21	Apple ProDOS	PRODOS
22	Macintosh	MAC
8	RT-11	RT11
9	RSX-11	RSX11
10	Pet	?
11	NEWDOS	NEWDOS
12	NEWDOS-80	NEW80
13	Atari DOS 2.0	ATARI
14	UNIX System 3	UNIX3
23	UNIX System 5	UNIX5

Table A.4
Boot Types

boot number	computer type	boot file name
0, 1, 2, 3	Apple II	BOOT.APPLE2
4, 5, 6, 7	Concept	BOOT.CONCEPT
9	IBM	BOOT.IBMPC
10	Xerox 820	BOOT.XR820
11	Zenith H89	BOOT.HZ89
12	NEC PC8000	BOOT.NC
13	Pet	BOOT.PET
14	Atari 800	BOOT.ATARI800
15	TRS-80 MOD I	BOOT.TRSI
16	TRS-80 MOD III	BOOT.TRSIII
17	LSI-11	BOOT.LSI11
18	Printer server	BOOT.PRINTSRV
19	Apple ///	BOOT.APPLE3
20	DEC Rainbow	BOOT.RAINBOW
21	TI Pro	BOOT.TIPRO
22	Z-100	BOOT.Z100
23	Concept2	BOOT.CONCEPT2
24	Companion	BOOT.COMPANION
25	Macintosh	BOOT.MAC
26	Sony SMC-7086	BOOT.SONY

Appendix B: Operating system specifics

Each section covers the implementation details of the specified operating system. This information includes the following topics:

- Overview.
- Volume format.
- Driver calls.
- Description of mount table.
- Memory locations used.
- Boot/logon process.
- Utilities.

B.1 Corvus Concept Operating System

You should also consult the following manuals:

Corvus Concept Operating System Reference Manual, Corvus P/N: 7100-02825. Describes how to write drivers; includes listing of system common.

Corvus Concept System Library User Guide, Corvus P/N: 7100-03295. Describes units CCDIRIO (directory i/o), CCOMNIO and CCOTCIO (transporter interfaces), CCDRVIO (Corvus drive interface).

Overview

There are 4 disk drivers for the Concept Operating System: two floppy disk drivers and 2 hard disk drivers. The floppy disk drivers (DRV.FDISK and DRV.ADISK) are loadable; the hard disk drivers (Local and Omninet) are built into the CCOS kernel. Drivers are associated with unit numbers ranging from 1 to 36. By convention, disk drivers are only associated with units 4, 5, and 9 through 29. Unit 4 is always the system boot volume, and must contain certain operating system files.

In addition to the disk drivers, there are two other drivers used by Constellation II. One is called SLOTIO and is used for sending arbitrary disk commands. The other is called OMNINET and is used for sending arbitrary Omninet commands. Both of these drivers are built into the CCOS kernel.

Volume format

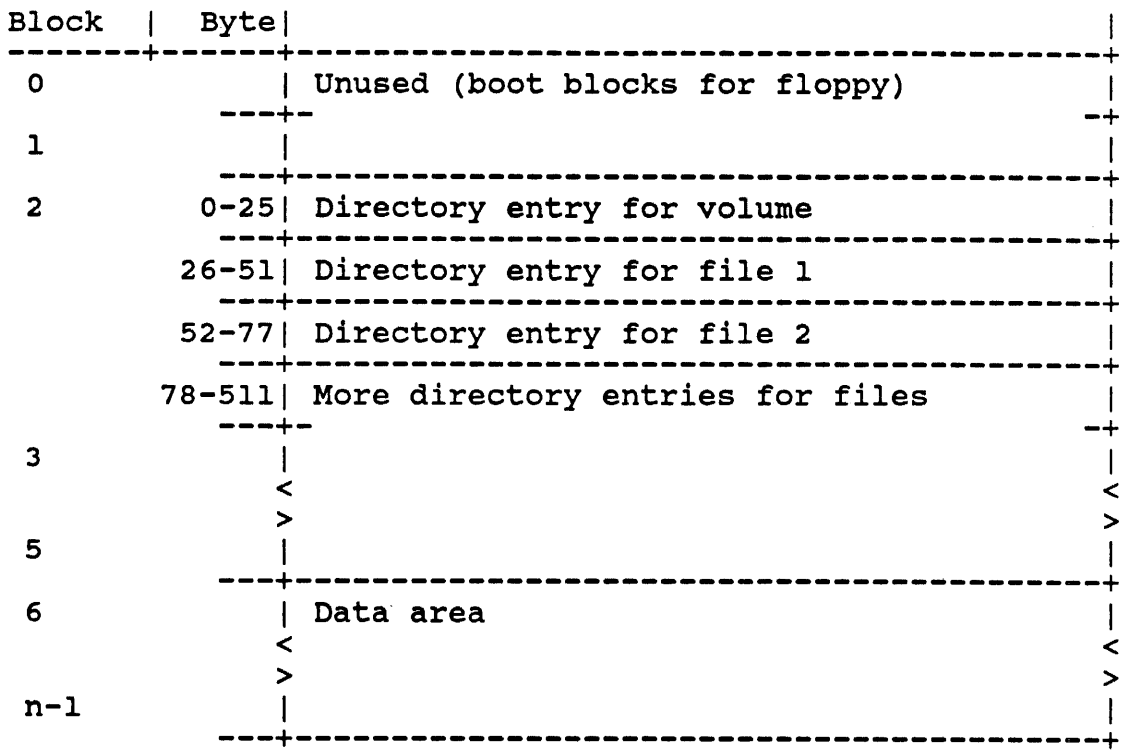
The volume format used by CCOS is identical to that used for UCSD p-system volumes. By convention, volumes whose directory is stored with most significant byte first are called CCOS format, and volumes whose directory is stored with least significant byte first (a "flipped" directory) are called UCSD format.

Refer to the Corvus Concept System Library User Guide, CCDIRIO unit, for more information. Constellation II software also uses the unit C2UTIL1 in library CIILIB.OBJ for directory i/o.

The first two blocks of each volume are currently unused; these blocks are used for boot code on a floppy. The next 4 blocks are the directory. Each directory entry is 26 bytes; there is space for 78 entries in the 4 blocks (78*26=2028 bytes; the remaining 20 bytes are unused). The first entry in the

directory always describes the volume itself, giving volume size and number of files. The rest of the entries describe the individual files. Only the first n+1 entries are valid, where n is the number of files indicated in the volume entry. Unused entries may appear valid, since previously used entries are not zeroed out.

CCOS Volume layout



n is the volume length, as specified by the user when the volume was created.

When formatting a volume with the Constellation II VOLUME MANAGER program, the user can choose 1 of 3 options. These have the following effect:

1) Initialize volume: Read blocks 2 through 5. Create directory entry for volume, truncating volume name to 7 characters, and indicating 0 files. Fill remainder of block 2 with 00h. Write blocks 2 through 5. Note that only block 2 is actually changed by this option.

2) Write volume header: Read blocks 2 through 5. Put new volume name and volume length into volume entry. Write blocks 2 through 5. Note that only the volume name and volume length are actually changed by this option.

3) Zero directory: Same as option (1).

Directory entry for volume

Byte		
0	Firstblock - first block number of volume (usually 0000h)	(msb)*
1		(lsb)
2	Nextblock - first block number of data area (usually 0006h)	(msb)
3		(lsb)
4	Fkind (low order 4 bits) - kind of file (0000h for a volume)	(msb)
5		(lsb)
6	Dtid[0] - length of volume name	
7	Dtid - volume name (7 bytes max)	
< >		< >
13		
14	Deovblock - volume size	(msb)
15		(lsb)
16	Dnumfiles - number of files in volume	(msb)
17		(lsb)
18	Dloadtime - date of last access	(msb)
19		(lsb)
20	Dlastboot - date of last boot	(msb)
21		(lsb)
22	Memflipped (see below)	(msb)
23	Dskflipped	(lsb)
24	Unused	
25		

* All entries marked with (msb) and (lsb) are flipped for UCSD format volumes.

Memflipped and **Dskflipped** are only valid when the directory has been read into memory by the GetDir routine.

The following test determines whether a directory is flipped or not:

```
IF Nextblock = 3 OR Nextblock = 6 OR Nextblock = 10 THEN
   directory is not flipped
 ELSE
   IF Flip(Nextblock) = 3 OR Flip(Nextblock) = 6 OR
      Flip(Nextblock) = 10 THEN
      directory is flipped
   ELSE
      invalid directory.
```

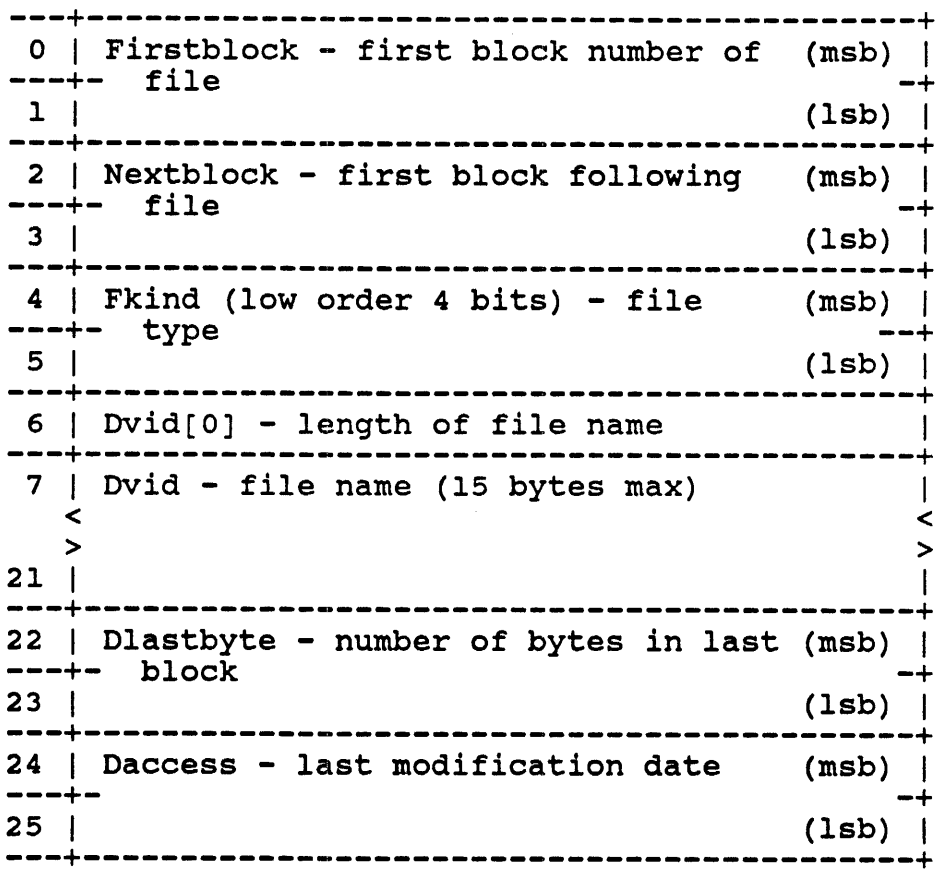
Values for FKind are given below:

Fkind	Meaning
0	Volume
2	Code file
3	Text file
5	Data file
8	Volume

The date field has the following format:

```
Bits 0..6 (7 bits):    year (0->1900, 84->1984)
Bits 7..11 (5 bits):   day
Bits 12..15 (4 bits):  month (1->Jan, 12->Dec)
```

Directory entry for file



Driver calls

CCOS defines the following driver calls:

```
UNITREAD (unit number, buffer, byte count, block number,
          mode )
UNITWRITE(unit number, buffer, byte count, block number,
          mode )
UNITSTATUS (unit number, buffer, function code)
UNITBUSY (unit number)
UNITCLEAR (unit number)
UNITMOUNT
UNITUNMOUNT
UNITINSTALL
```

Refer to the *Pascal Reference Manual* or the *Operating System Reference Manual* for explanation of the parameters.

The disk driver implements the following driver calls:

```
UNITREAD (unit number, buffer, byte count, block number,
          mode )
UNITWRITE(unit number, buffer, byte count, block number,
          mode )
```

The SLOTIO driver implements the following driver calls:

```
UNITREAD (unit number, buffer, byte count, block number,
          mode )
UNITWRITE(unit number, buffer, byte count, block number,
          mode )
```

UNITWRITE is used to send a drive command, and UNITREAD is used to receive the results.

The **unit number** of the SLOTIO driver can be obtained by calling the parameterless function OSSltDv.

The parameter **buffer** contains the drive command on UNITWRITE, and the command result bytes on UNITREAD.

The parameter **byte count** contains the length of the command on UNITWRITE; for UNITREAD, **byte count** is set to the maximum number of bytes expected on the call, and on return it will be set to the number of bytes actually received.

The parameter **block number** is not used.

The **mode** parameter is used to specify the slot and server number that the command is directed to; **mode**.lsb = slot number;

mode.msb = server number (transporter address).

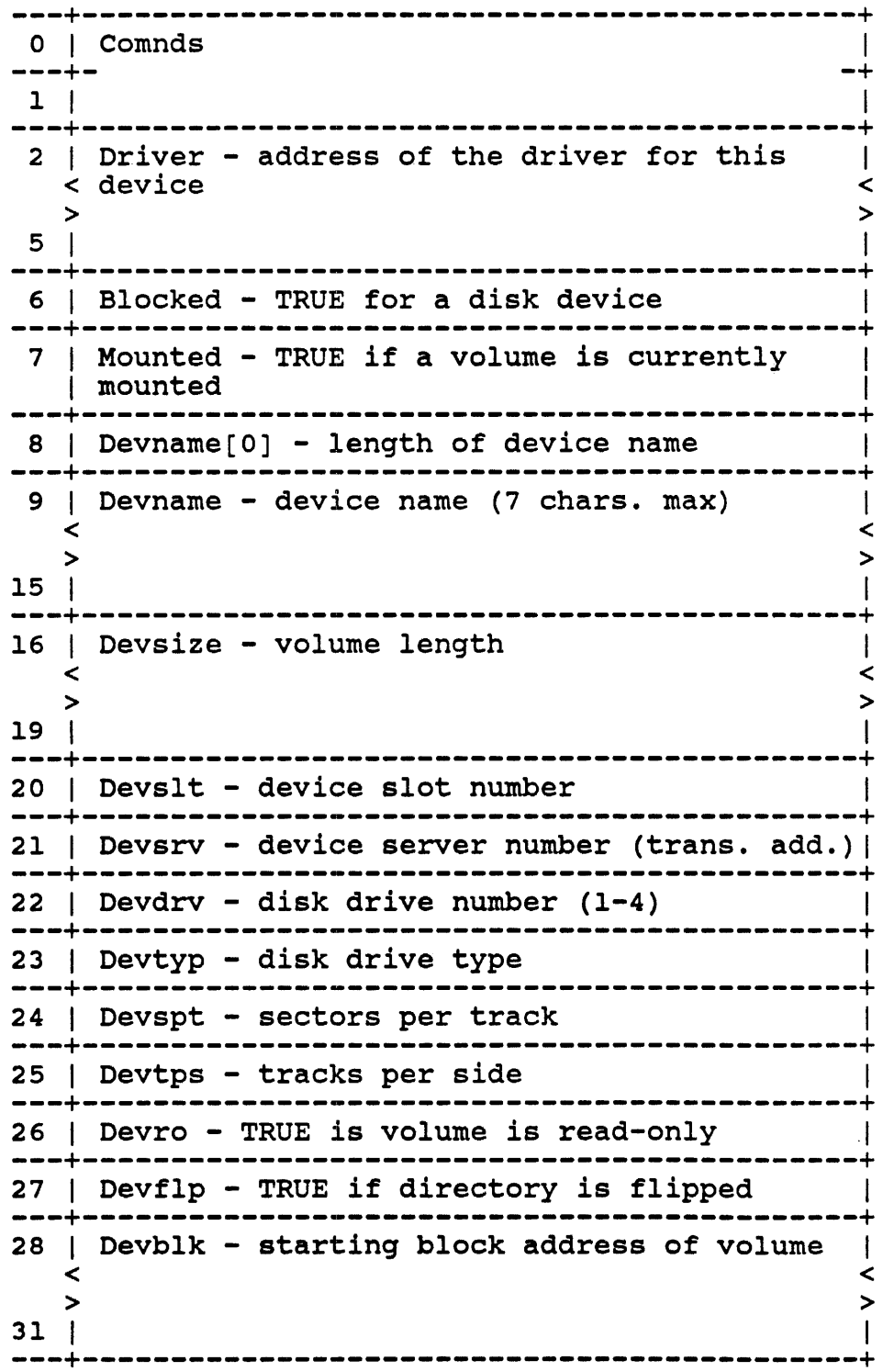
The OMNINET driver implements the following driver calls:

UNITSTATUS (unit number, buffer, function code)

For a description of the OMNINET driver, refer to the documentation on the CCOTCIO unit.

Description of Concept mount table

The Concept mount table is called the system device table (sysdevtab), and is pointed to by bytes 20-23 of the system global area. The first word of the system device table indicates how many entries there are; each entry is described by the table on the next page:


0 Comnds
1
2 Driver - address of the driver for this device
5
6 Blocked - TRUE for a disk device
7 Mounted - TRUE if a volume is currently mounted
8 Devname[0] - length of device name
9 Devname - device name (7 chars. max)
15
16 Devsize - volume length
19
20 Devslt - device slot number
21 Devsrv - device server number (trans. add.)
22 Devdrv - disk drive number (1-4)
23 Devtyp - disk drive type
24 Devspt - sectors per track
25 Devtps - tracks per side
26 Devro - TRUE is volume is read-only
27 Devflp - TRUE if directory is flipped
28 Devblk - starting block address of volume
31

The following fields must be set when mounting a hard disk volume:

```
Comnds  - 15
Driver  - set to driver address, obtained from system slot
          table (SysSltTab.drv[Devslt])
Blocked - TRUE
Mounted - TRUE
Devname - first 7 characters of volume name
Devsize - volume size
Devslt -  slot number of volume
Devsrv -  server number (transporter address)
Devdrv -  Corvus drive number
Devro -   TRUE for read-only; FALSE for read-write
Devblk -  starting block address of volume
```

Memory locations used

The pointer to the System Common area is located in bytes 180h through 183h.

The system device table pointer is located in bytes 20h through 23h of the System Common area.

The encrypted user name is located in bytes 726h through 72Fh. The unencrypted user name is located in bytes 71Ch through 725h.

Boot process

After finding the user's boot volume, the boot code looks for the file CC.KERNEL in the boot volume. This file is loaded and control is passed to it, along with the following information:

- User name (encrypted and unencrypted)
- Boot volume information: unit number, slot number, server number, drive number and starting block address.

When the screen goes blank after entering the user password, the boot code has completed executing. The lines of dots seen next on the screen are output during execution of CC.KERNEL.

Once the kernel has completed, it transfers control to a program called CC.SETUP, which must also reside in the user's boot volume. This program blanks the screen, puts up the user windows, and builds the user's mount table.

The mount table is built as follows: the access table is read from the current server, and each volume is either mounted on the

specified unit, or, if a volume is already mounted on the specified unit, put into an "overflow" table. Additional access tables are read and volumes mounted from all other disk servers, starting with the lowest transporter address. After all access tables have been read, volumes from the "overflow" table are mounted on any available units.

Utilities

The following library units are described in the System Library User Guide:

CCLIB
- CCDIRIO - directory description and I/O
- CCOMNIO - sending Omninet messages; obsolete
- CCOTCIO - new unit for Omninet interface; use it instead of CCOMNIIO

C2LIB
- CCDRVIO - sending disk commands
- CCPIPES - sending pipes commands
- CCSEMA4 - sending semaphore commands

The following library units are described in Chapter 6 of this manual:

CIILIB
- C2UTIL1 - reading and writing volume directories, name and password encryption and decryption
- MOUNTS - mounting, unmounting disk volumes

Utilities include a mount manager (CC.MNTMGR), a spooler (SPOOL), and a despooler (DESPOOL).

B.2 MSDOS 1.x

Drivers currently available: IBM PC/XT, TI Pro

Overview

DOS 1.x uses two files to boot from floppy: IBMBIO.COM and IBMDOS.COM. The file IBMBIO.COM contains the BIOS, while the file IBMDOS.COM contains the operating system.

The BIOS provided by IBM for MSDOS 1.x is replaced by the BIOS supplied by Corvus. The Corvus-supplied BIOS duplicates that provided by IBM as well as providing a Corvus driver. The user must boot from the Corvus drive.

A maximum of 10 Corvus volumes and 2 floppies may be mounted. Volumes are referred to externally by letters A through L; internally, volumes are referred to as unit numbers 0 through 11. The volume mounted on unit A must contain the file COMMAND.COM.

The ability to dynamically mount and unmount volumes is severely limited in MSDOS 1.x. MSDOS uses a File Allocation Table (FAT) to control disk allocation; a copy of the FAT of each on-line volume is kept in memory. The memory space for the FAT is allocated at boot time, and cannot be changed. Therefore, whenever a volume is mounted, its FAT table must fit within the previous volume's FAT table. Corvus has extended this restriction to say that a volume mounted over another must match that volume's configuration exactly.

Volume format

See section B.3.

Driver calls

The driver calls for MSDOS 1.x are not documented.

Description of mount table

Several tables within the BIOS must be modified when mounting a volume. These are listed here merely as a summary; for complete information, consult the driver listings.

MTTAB: Unit Assignment Table. One byte entry for each possible unit (0 to 11), indicating whether unit is

unassigned, assigned to a floppy, or assigned to a Corvus volume. Also indicates read/write access for a Corvus volume.

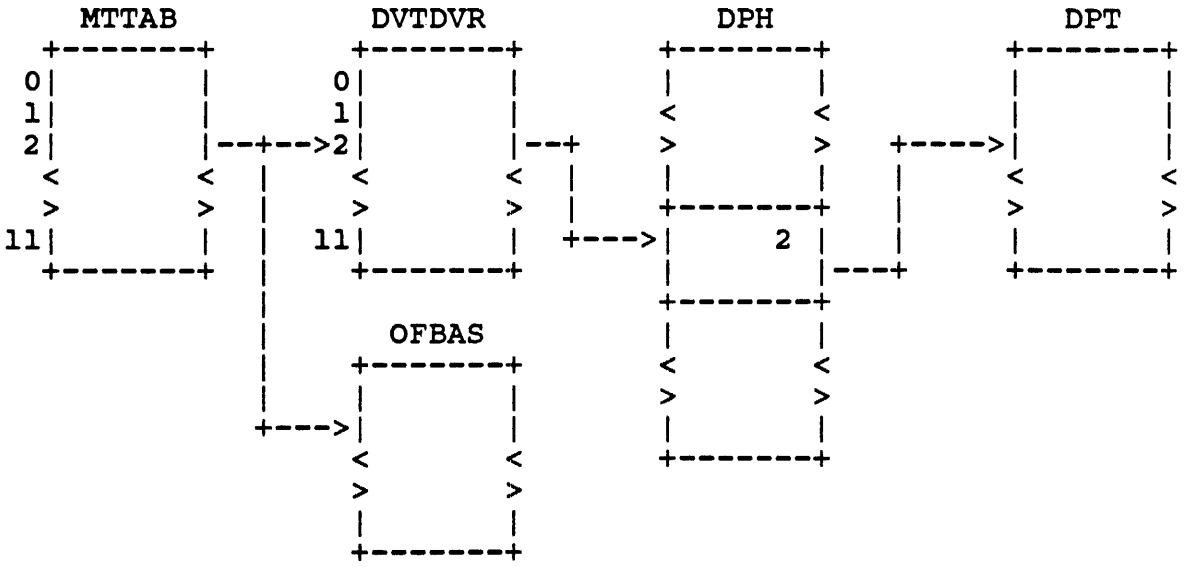
DVTDVR: Volume Configuration Pointer Table. One byte entry for each possible unit (0 to 11). Each entry is a DPH entry number (see below), so this table is really just a mapper of unit number to DPH entry number.

OFBAS: Corvus drive(s) map. 5 byte entry for each possible Corvus volume. Index is masked from MTTAB entry. Each entry contains the starting disk block address of the volume and the Corvus server and drive number.

DPH: Disk Configuration Tables. Each entry consists of a unit number followed by a pointer to a volume configuration. For any unit, at most one entry in this table is "activated" by an entry in the volume configuration pointer table DVTDVR.

DPTx: Disk Parameter Table(s). 10 bytes each, one for each possible disk configuration. Pointed to by DPH. Includes information such as number of reserved sectors, FAT copies on disk number of directory entries, and sectors per disk.

The figure below shows how the tables fit together:



Memory locations used

```
Flat cable interface card:
    Disk i/o port #:     2EEh
    Disk status port #:  2EFh
    Drive ready:         001h
    Bus direction:       002h

Omninet and flab cable interface card ROMs:
  Segment # of I/O ROM:  DF00h
  ROM entry points:      DF00h   ; cold start
                         DF03h   ; warm start
                         DF06h   ; ROM I/O support routines
```

Boot process

MSDOS 1.x must be booted from the Corvus hard disk; that is, it is not possible to boot MSDOS 1.x from a floppy and then "log-on" to the Corvus. A boot from the Corvus hard disk is accomplished by jumping to the entry point of the Corvus supplied interface ROM at location DF00h. The following BASIC program shows a jump to this ROM:

```
10 DEF SEG=&HDF00
20 X = 0
30 CALL X
40 END
```

The ROM code finds a disk server and loads the IBM boot code from the Corvus volume, using the Corvus Read Boot Block command. The boot code then prompts for the user name and password. It finds the user's home server and boot volume.

The boot volume (the one marked with x in the Access Manager) must be a UCSD-format volume which contains the files IBM.MSDOS.BIO and IBM.MSDOS.DOS. The IBM.MSDOS.BIO file is supplied by Corvus on the distribution diskette CORIBM3:. The IBM.MSDOS.DOS file is read from your IBM DOS boot diskette during installation.

The boot code loads the files IBM.MSDOS.BIO and IBM.MSDOS.DOS and builds the user's mount table. Any units not assigned to a Corvus volume are assigned to the built-in floppy. Control is then transferred to DOS.

All volumes mounted for the user must be MSDOS format volumes. The volume mounted on unit A must contain the IBM supplied file COMMAND.COM.

Utilities

A mount manager program (MNTMGR.EXE) provides the ability for a user to mount or unmount specific volumes. As mentioned above, this capability is limited in that volumes must be mounted over existing volumes, and the volume configurations of the two volumes must match exactly.

A spooler (SPOOL.EXE) and despooler (DESPOOL.EXE) are also supplied.

Each of these utilities is written in MS Pascal.

An assembly language module provides the capability to send arbitrary commands to the Corvus drive. This module calls the driver to actually send the commands. The module DRIVEC2.OBJ is for use with the MS Pascal compiler; the module BDRIVEC2.OBJ is for use with the MS Basic compiler.

The procedures supplied by the module are listed below:

```
Pascal:  Function INITIO:  INTEGER
Basic:   CALL INITIO(A%)
```

Must be called before any calls to other procedures in this module. This function ensures that the Corvus driver is installed, and returns a non-zero value if it is successful. If this function fails, it returns 0.

```
Pascal:  Procedure CDSEND( VAR st: LONGSTRING)
         Procedure CDRECV( VAR st: LONGSTRING)
BASIC:   CALL CDSEND(B$)
```

These procedures are used to send (CDSEND) arbitrary commands to the Corvus drive, and receive (CDRECV) the results. The first two bytes of the parameter are treated as an integer specifying the number of bytes to send (CDSEND), or the number of bytes received (CDRECV). The remainder of the string contains the command bytes or result bytes.

The ROM entry points are described next:

To use the ROM, call the entry points with standard 8086 intersegment CALL's (long CALL's). All registers will be preserved (including the flag register). When using the I/O services entry point, codes will be returned in the AX and CX registers.

DF000h - Coldstart. The coldstart entry initializes the interface card, locates a disk server(if the interface card is Omninet), loads the Constellation II boot program and transfers control to it.

DF003h - Warmstart. The warmstart entry point initializes the interface card. This should be done once by all drivers and routines that use the I/O services.

DF006h - ROM i/o services. The i/o services entry point contains routines to communicate with the Corvus disk drives and network servers.

The request code and parameters for I/O services are passed in 8086 registers as shown below:

```
; (AH) = 0      Identify interface
;
;              On exit
;              (AL) = 0 if Omninet interface card
;                     1 if flatCable interface
;
;              with Omninet interfaces
;
;              (AH) = Omninet transporter address
;                     If (AH) = 255 then another transporter
;                     with the same address is on the network.
;
; (AH) = 1      Xmit/Recv data to drive
;
;              DS:(SI) address of data to send to drive
;              ES:(DI) address of buffer for data from drive
;              (CX)    number of bytes of data to send to drive
;                      (max = 530)
;              (DX)    number of bytes wanted back from drive
;                      (do not include return code; max=530)
;
;              with Omninet interfaces
;
;              (AL)    network address of disk server
;              (BL)    number of timer units to wait for reply
;                      from disk server; 0 = do not abort, wait
;                      forever; timer unit is approx .86 seconds
;              (BH)    number of xmit trys; 255 = retry until
;                      success; should be greater than 0;
;                      0 = 255 trys.
;
;
;              with flatCable interfaces
;
```

```
;                   (BL)    number of timer units to wait for disk
;                           drive to come ready before first byte is
;                           sent to drive; this is also the number
;                           of timer units to wait for the interface
;                           to turn around after the command sequence
;                           is sent to the drive; 0 = do not abort,
;                           wait forever; timer unit is approx .86
;                           seconds
;
;                   On exit
;
;                   (AL)    return code from drive. 255 = aborted
;                   (CX)    number of bytes received from drive;
;                           count includes the return code.
;
; (AH) = 2          Xmit data to network server
;
;                   ES:(SI) address of data to transmit to server
;                   (CX)    number of bytes to transmit
;                   (AL)    network address of server. Use 255 to
;                           broadcast to all servers.
;                   (BH)    number of times to try to xmit. If this is
;                           a broadcast, number of times to xmit data.
;
;                   On exit
;
;                   (AL)      0 = all ok
;                           255 = transmission aborted
;
; (AH) = 3          Xmit/Recv data to network server
;
;                   DS:(SI) address of data to transmit to server
;                   ES:(DI) address of buffer for data from server
;                   (CX)    number of bytes to transmit (530 max)
;                   (DX)    number of bytes to receive (530 max)
;                   (AL)    network address of server. Use 255 to
;                           broadcast to all servers.
;                   (BL)    number of timer units to wait for reply
;                           from server; 0 = do not abort, wait
;                           forever; timer unit = approx .86 seconds
;                   (BH)    number of times to try. Should be greater
;                           than 0; 0 = 255 times.
;
;                   On exit
;
;                   (AL)      0 = all ok
;                           255 = transmission aborted
;
; (AH) = 4          Find any disk server on network. Uses simple
;                   broadcast command, since any version disk server
```

```
;                   recognizes this command. Requests 2 and 3
;                   require a disk server with multiple server support.
;
;                   (BL)    number of timer units to wait for reply
;                           from server; 0 = do not abort, wait
;                           forever; timer unit = approx .86 seconds
;                   (BH)    number of times to try. Should be greater
;                           than 0; 0 = 255 times.
;
;                   On exit
;
;                   (AL)    0 = all ok
;                         255 = transmission aborted
;
;                   (AH)    network address of disk server that
;                           responded
;
; (AH) = 5          Send write command to drive
;
;                   DS:(SI) address of command to send to drive
;                   ES:(DI) address of data to send to drive
;                   (CX)    number of bytes of command to send to
;                           drive (normally 4)
;                   (DX)    number of bytes of data to send to drive
;                           (normally 512.)
;
;                   with Omninet interfaces
;
;                   (AL)    network address of disk server
;                   (BL)    number of timer units to wait for reply
;                           from disk server; 0 = do not abort, wait
;                           forever; timer unit is approx .86 seconds
;                   (BH)    number of xmit trys. 255 = retry until
;                           success; should be greater than 0;
;                           0 = 255 trys.
;
;
;                   with flatCable interfaces
;
;                   (BL)    number of timer units to wait for disk
;                           drive to come ready before first byte is
;                           sent to drive.  This is also the number
;                           of timer units to wait for the interface
;                           to turn around after the command sequence
;                           is sent to the drive; 0 = do not abort,
;                           wait forever; timer unit is approx .86
;                           seconds
;
;
;                   On exit
```

```
;
;               (AL)    return code from drive. 255 = aborted
;               (CX)    number of bytes received from drive;
;                       count includes the return code
```

B.3 MSDOS 2.x, 3.x

Drivers currently available: IBM PC/XT, TI Pro, DEC Rainbow

You should also consult the following manuals supplied by your computer manufacturer:

For IBM-PC, DOS 2.0
DOS Manual

For IBM-PC, DOS 3.0
DOS 3.0 Technical Reference Manual, IBM P/N 6322677

For TI Pro,

For DEC Rainbow,

General
MSDOS Technical Reference Manual, Microsoft P/N

Overview

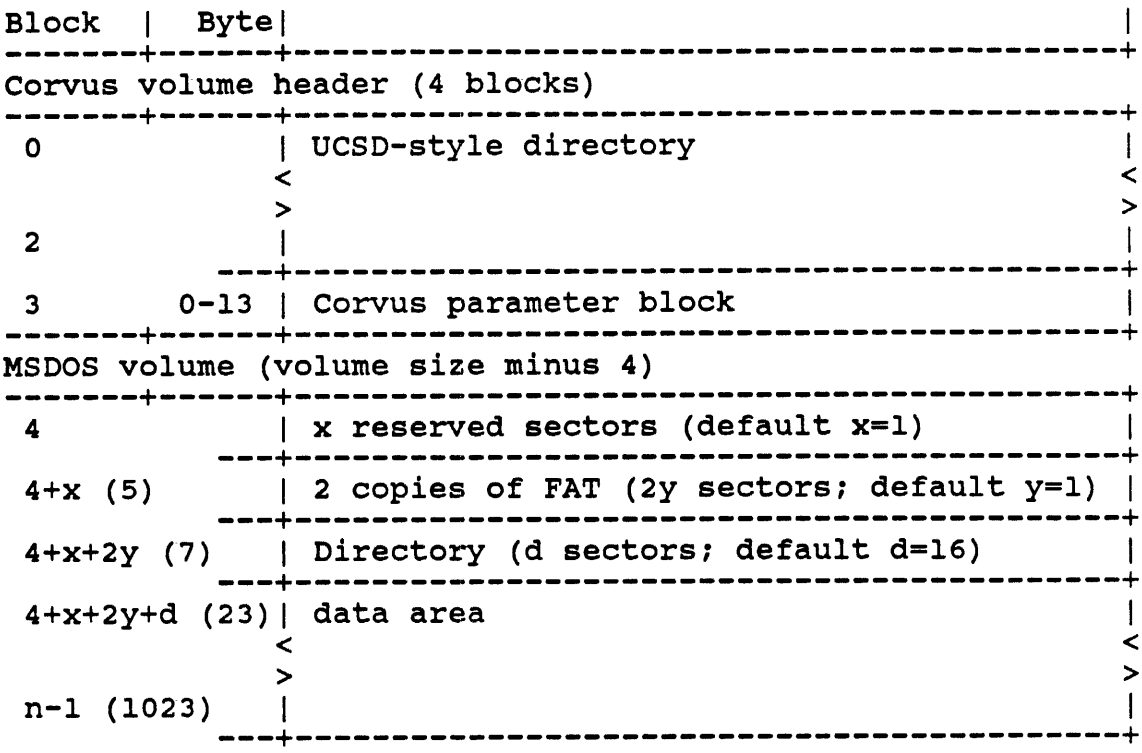
Versions 2 and 3 of MSDOS support attachable device drivers. The user boots from either the floppy disk or the fixed disk, and the Corvus-supplied driver file is loaded into the operating system. Note that booting from the Corvus drive is NOT supported for MSDOS 2.x and 3.x.

Volume format

MSDOS volumes on the Corvus consist of a 4 block Corvus volume header, followed by a normal MSDOS volume as described in your MSDOS documentation. Volume attributes are specified by the system manager when the volume is created; these include volume size, number of reserved sectors, cluster size, and number of directory entries. The other attributes of a volume are either fixed, or computed as shown below.

Important note: If the number of reserved sectors is 1 or more, then a BIOS parameter block is written into the first reserved sector, as described in the DOS documentation. Many applications use this BIOS parameter block, so you should always specify at least one reserved sector when creating a volume.

MSDOS Volume layout



The block offsets for a volume created with the default parameters are given in parentheses.

n is the volume size, specified by the user.

There are 3 options a user can choose from when formatting a volume with the Corvus VOLUME MANAGER utility:

1) Initialize volume: Write volume header (blocks 2 and 3), BIOS parameter block, FATs, and directory.

2) Write header: Writes only blocks 2 and 3, and BIOS parameter block.

3) Zero directory (volume must have a valid header): Writes FATs and directory.

Corvus Volume Header
UCSD style directory blocks (blocks 0, 1, and 2)

Blocks 0 and 1 are unused. Block 2 is written with the following information:

0,1	First block number of volume	0000h
2,3	First block number of data	0300h
4,5	File kind	0000h
6	Length of volume name - user specified	xxh
7 < > 13	Volume name (7 bytes max) (first 7 bytes of user-specified volume name)	
14,15	Volume size - user specified	xxxxh
16,17	Number of files in volume	0100h
18,19	Time of last access	0000h
20,21	Date of last boot	xxxxh
22,23	Unused	0000h
24,25	Unused	0000h
26,27	First block number of file	0300h
28,29	Next block - volume size	xxxxh
30,31	File kind - data file	0500h
32	Length of file name - 10	0Ah
33 < > 47	File name (15 bytes max)	"MSDOS.DATA"
48,49	Bytes in last block - 512	0002h
50,51	Last modification date	xxxxh

The Corvus volume header is written on volume initialization, but is not used by any of the MSDOS utilities or drivers.

Corvus parameter block

The Corvus parameter block is block 3 of the volume.

0	Bytes per sector - always 512
1	
2	Sectors per cluster - user specified
3	
4	Number of reserved sectors - user
5	specified
6	Number of FAT copies - always 2
7	
8	Number of directory entries - user
9	specified
10	Sectors per volume - user specified
11	
12	Sectors per FAT - computed
13	

This data is written on volume initialization, and is used by the Corvus logon procedure and mount manager program when mounting a volume.

Sectors per cluster may be one of the following values: 1, 2, 4, 8, 16, 32, 64, 128. A volume may have a maximum of 4086 clusters, which means the maximum volume size is 523008 blocks (4086*128). The default cluster size depends upon the volume size. The defaults are given in the table below:

Volume size:	0-1999	2000-15999	16000-31999	>=32000
Default cluster size:	4	8	16	32

Number of reserved sectors may be between 0 and 32; default is 1. If 1 or more reserved sectors are specified, a DOS BIOS parameter block is written into the first reserved sector (see description in following pages).

Number of directory entries may be between 16 and 4080; default is 256. Each directory entry occupies 32 bytes. Since the directory occupies an integral number of sectors, you should always specify the number of directory entries as a multiple of 16 (16 entries * 32 bytes per entry = 512 bytes). The first entry of the directory is initialized with a volume label, and the other entries are set to E5h (deleted file). E5h is used rather than 00h to maintain compatibility with DOS 1.x.

Sectors per volume is equal to the user specified volume length, minus 4.

Sectors per FAT is computed from the number of clusters in the data area. The FAT contains 1.5 bytes for each cluster. Each FAT is initialized with the first 3 bytes set to F8h, FFh, FFh, and the remaining bytes set to 00h.

DOS 3.x supports 2 kinds of FATs: 1.5 bytes per cluster, and 2 bytes per cluster. Corvus only supports 1.5 bytes per cluster.

DOS BIOS parameter block

The BIOS parameter block (BPB) is located in the first reserved sector. The format of the BPB is specified by DOS; this table shows, in parentheses, the values used by Corvus. CPB refers to the Corvus parameter block, described above.

0	3 byte near JUMP to boot code
1	(E9h, FDh, FFh -- JMP 0)
2	
3	8 bytes OEM name and version
< > 10	("CORVUS20")
11	Bytes per sector - always 512
12	(same as CPB, bytes 0,1)
13	Sectors per allocation unit (same as CPB, bytes 2,3)
14	Number of reserved sectors
15	(same as CPB, bytes 4,5)
16	Number of FATs - always 2 (same as CPB, bytes 6,7)
17	Number of root directory entries
18	(same as CPB, bytes 8,9)
19	Number of sectors in logical image
20	(same as CPB, bytes 10,11)
21	Media descriptor - always FEh
22	Number of FAT sectors
23	(same as CPB, bytes 12,13)

(continued on next page)

DOS BIOS parameter block
(continued from previous page)

24	Sectors per track - always 0
25	(0000h)
26	Number of heads - always 0
27	(0000h)
28	Number of hidden sectors - always 0
29	(0000h)

Driver calls

Refer to the description of the MSDOS driver interface contained in your MSDOS documentation.

The Corvus supplied driver file actually contains two drivers: a character device driver used for mounting volumes, and a blocked device driver, used for disk i/o.

The blocked device driver supports up to 10 devices; the actual number of devices is determined at boot time, and is 2 greater than the number of mounted volumes.

The character device driver is called "UTILHOOK". If you output a character to it, it returns a 4-byte address of the Corvus tables located within the driver. These tables can then be examined or modified. The Mount Manger program for MSDOS uses this feature to display and modify the user's mount information.

Description of mount table

Look at the Pascal source file CVUTILS.IMP for a description of the MSDOS mount tables.

Memory locations used

Not applicable.

Boot/logon process

Corvus does not support booting from the Corvus drive; the user must boot from some other device (e.g., a built-in floppy or fixed disk), and have the Corvus driver configured in. Log-on is performed by the INIT function of the Corvus driver.

Log-on prompts for the user name and password, locates the home disk server, and builds the user's mount table.

Utilities

See Section B.2 for information on utilities. In addition to the utilities described there, Corvus also supplies a spooler driver for MSDOS 2.x.

B.4 Softech p-system Version IV

Drivers available for: IBM, DEC Rainbow, TI Pro, Z-100

Internal Archictecture Manual, Chapter on Low-level I/O
Program Development Reference Manual
Adaptable System Installation Manual

Overview

There are many versions of the p-system available. The ones supported by Corvus are listed below.

Version IV.3 from Softech supports external drivers. Corvus does not supply a driver for this version, although it is fairly straightforward for you to develop your own driver.

IBM

Corvus uses the 8086 p-system interpreter from Softech, and has modified the BIOS to support the Corvus drive. Corvus is currently supporting the version IV.1 interpreter.

Corvus also supplies a driver which links into the NCI C.x version of Softech's version IV.

Note that Corvus does NOT support the p-system supplied by IBM.

TI

Corvus uses a version IV interpreter supplied by NCI.

Note that Corvus does NOT support the p-system supplied by TI.

DEC Rainbow, Z-100

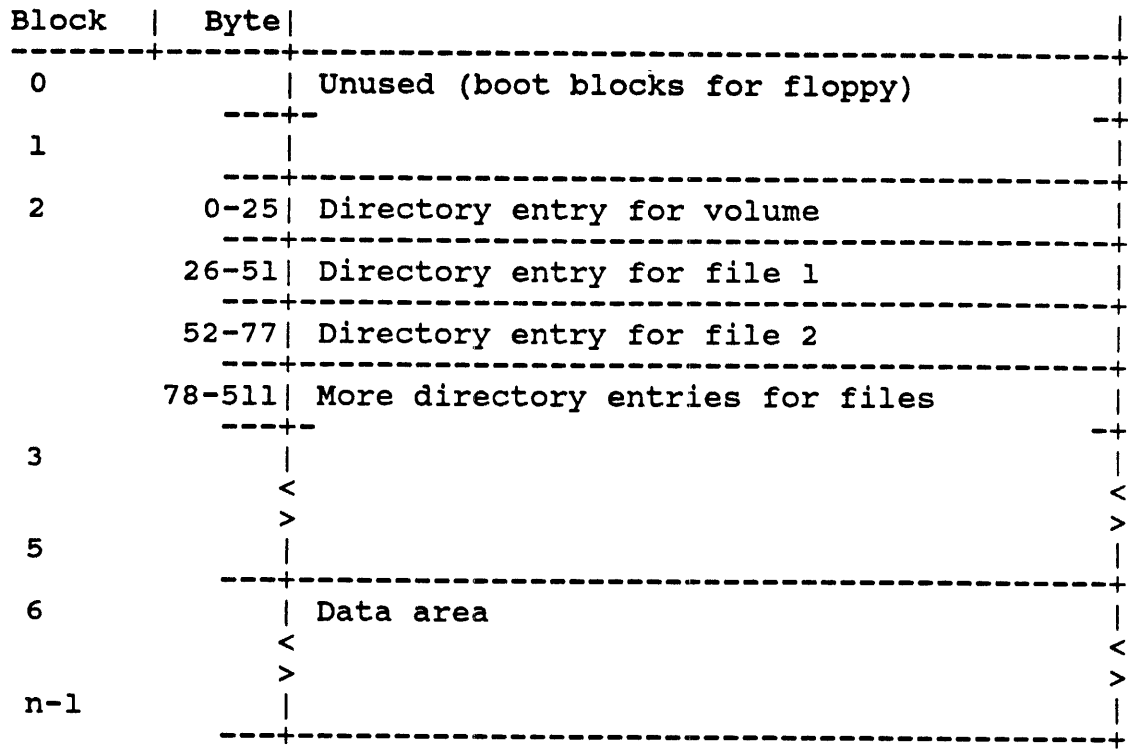
Corvus uses the CP/M-86 adaptable p-system interpreter from Softech. All communication with the Corvus drive is handled with calls to the CP/M-86 BIOS, which contains a driver for the Corvus drive.

Volume format

The first two blocks of each volume are currently unused; these blocks are used for boot code on a floppy. The next 4 blocks are the directory. Each directory entry is 26 bytes;

there is space for 78 entries in the 4 blocks (78*26=2028 bytes; the remaining 20 bytes are unused). The first entry in the directory always describes the volume itself, giving volume size and number of files. The rest of the entries describe the individual files. Only the first n+1 entries are valid, where n is the number of files indicated in the volume entry. Unused entries may appear valid, since previously used entries are not zeroed out.

p-system volume layout



n is the volume length, as specified by the user when the volume was created.

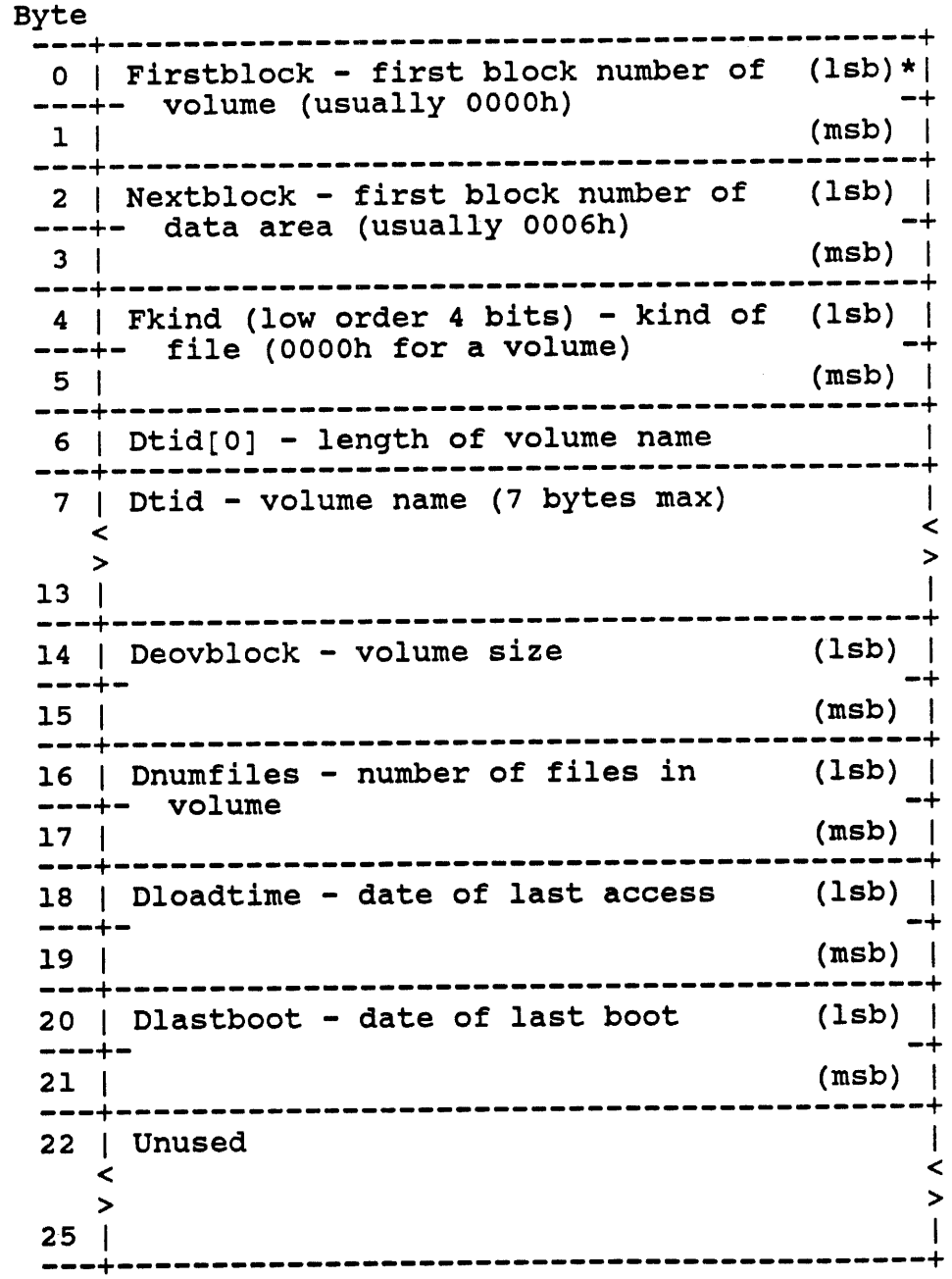
When formatting a volume with the Constellation II VOLUME MANAGER program, the user can choose 1 of 3 options. These have the following effect:

1) Initialize volume: Read blocks 2 through 5. Create directory entry for volume, truncating volume name to 7 characters, and indicating 0 files. Fill remainder of block 2 with 00h. Write blocks 2 through 5. Note that only block 2 is actually changed by this option.

2) Write volume header: Read blocks 2 through 5. Put new volume name and volume length into volume entry. Write blocks 2 through 5. Note that only the volume name and volume length are actually changed by this option.

3) Zero directory: Same as option (1).

Directory entry for volume



* All entries marked with (lsb) and (msb) may be flipped.

The following test determines whether a directory is flipped or not:

```
   IF Nextblock = 3 OR Nextblock = 6 OR Nextblock = 10 THEN
      directory is not flipped
    ELSE
      IF Flip(Nextblock) = 3 OR Flip(Nextblock) = 6 OR
         Flip(Nextblock) = 10 THEN
         directory is flipped
      ELSE
         invalid directory.
```

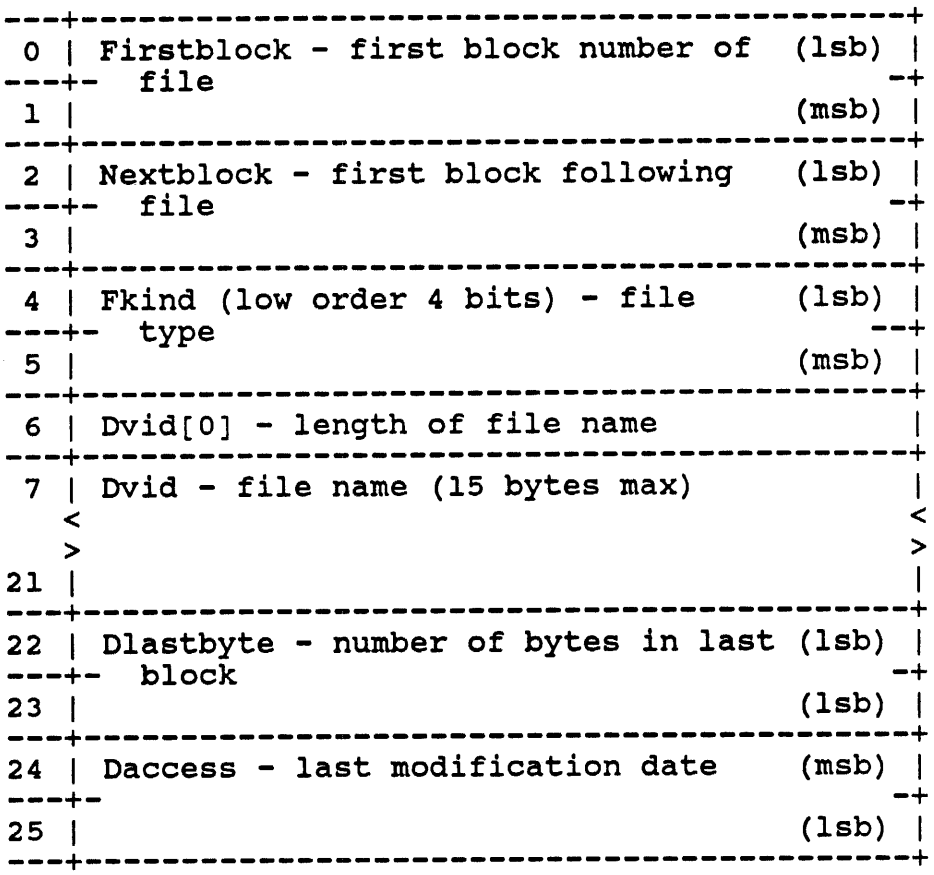
Values for FKind are given below:

Fkind	Meaning
0	Volume
2	Code file
3	Text file
5	Data file
8	Volume

The date field has the following format:

```
    Bits 0..6 (7 bits):    year (0->1900, 84->1984)
    Bits 7..11 (5 bits):   day
    Bits 12..15 (4 bits):  month (1->Jan, 12->Dec)
```

Directory entry for file



Driver calls

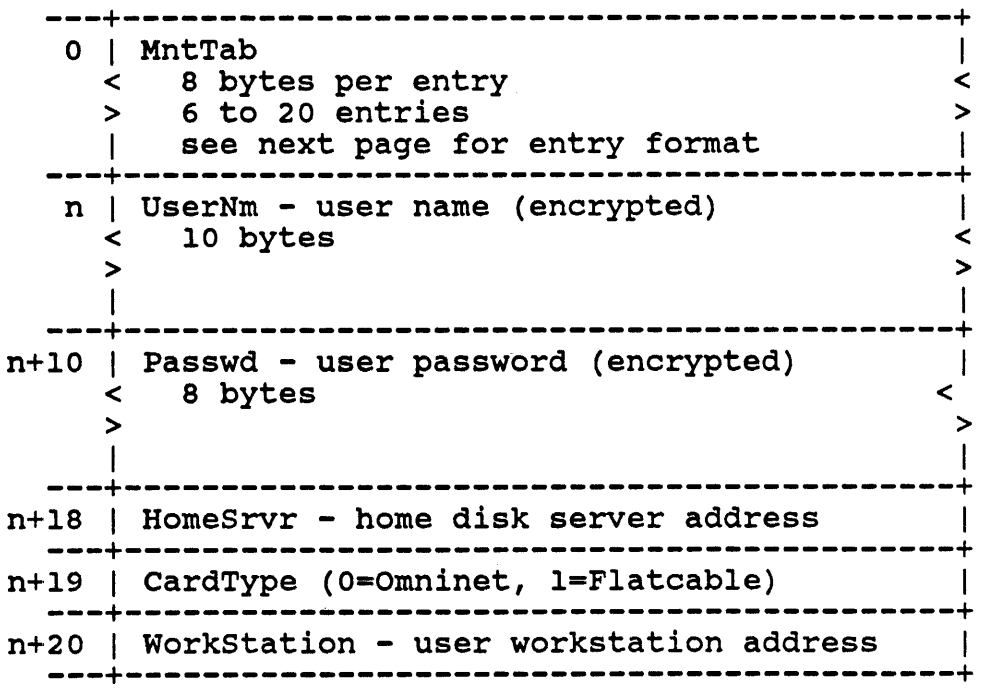
The p-system defines the following driver calls:

```
UNITREAD (unit number, buffer, byte count, block number,
          mode )
UNITWRITE(unit number, buffer, byte count, block number,
          mode )
UNITSTATUS (unit number, buffer, function code)
UNITBUSY (unit number)
UNITCLEAR (unit number)
```

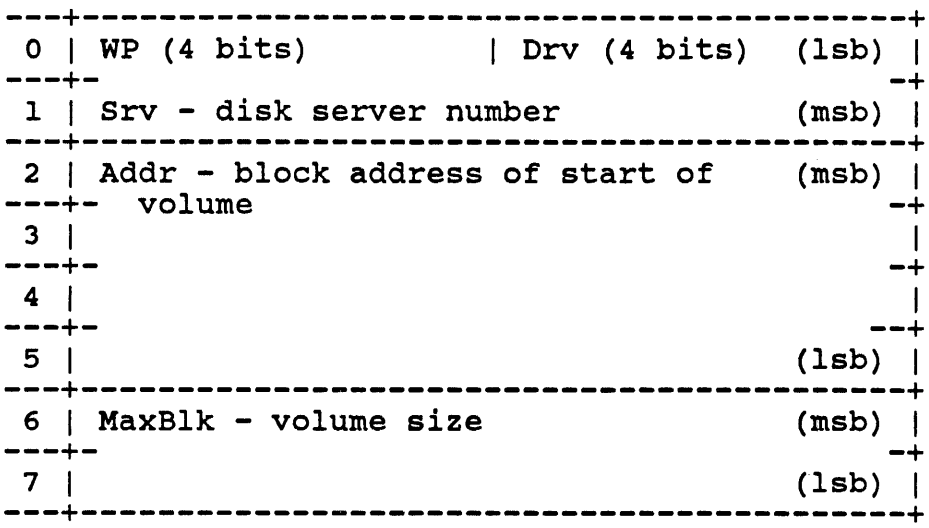
The parameters to these calls are documented in the p-system *Program Development Reference Manual*.

Description of mount table

p-system mount table



Mount table entry



Memory locations used

Boot process

For Softech p-system - 8086 interpreter

The boot code is loaded from the Corvus volume. This code prompts for user name and password, locates the home disk server, loads the interpreter from the user's boot volume, builds the user's mount table, and transfers control to the interpreter. The interpreter expects to find the file SYSTEM.PASCAL in the volume mounted on unit 4.

For NCI p-system

The user boots from a floppy which has the Corvus driver configured into the file SYSTEM.IBM. Then the user executes the Corvus program LOGON.CODE, which prompts for the user name and password and builds the user's mount table.

For Softech Adaptable p-system - CP/M 86

The user boots into CP/M 86 from a floppy. Then the user executes the Corvus program ...

Utilities

Corvus supplies a mount manager (MNTMGR.CODE), a spooler (SPOOL.CODE), and a despooler (DESPOOL.CODE).

A logon program is also supplied.

The following library units are supplied in the library CORVUS.LIB:

Unit name	Description
UCDefn	Miscellaneous declarations
UCCrtIO	Console I/O
UCOmnIO	Omninet message sending
UCDrvIO	Corvus drive I/O, CDSEND, CDRECV
UCSema4	Semaphore operations
UCPipes	Pipes operations
Mounts	Mount procedures
CVCrypt	Encryption/decryption
C2Util1	Miscellaneous procedures

B.5 CP/M 86

Drivers currently available: DEC Rainbow, Z-100

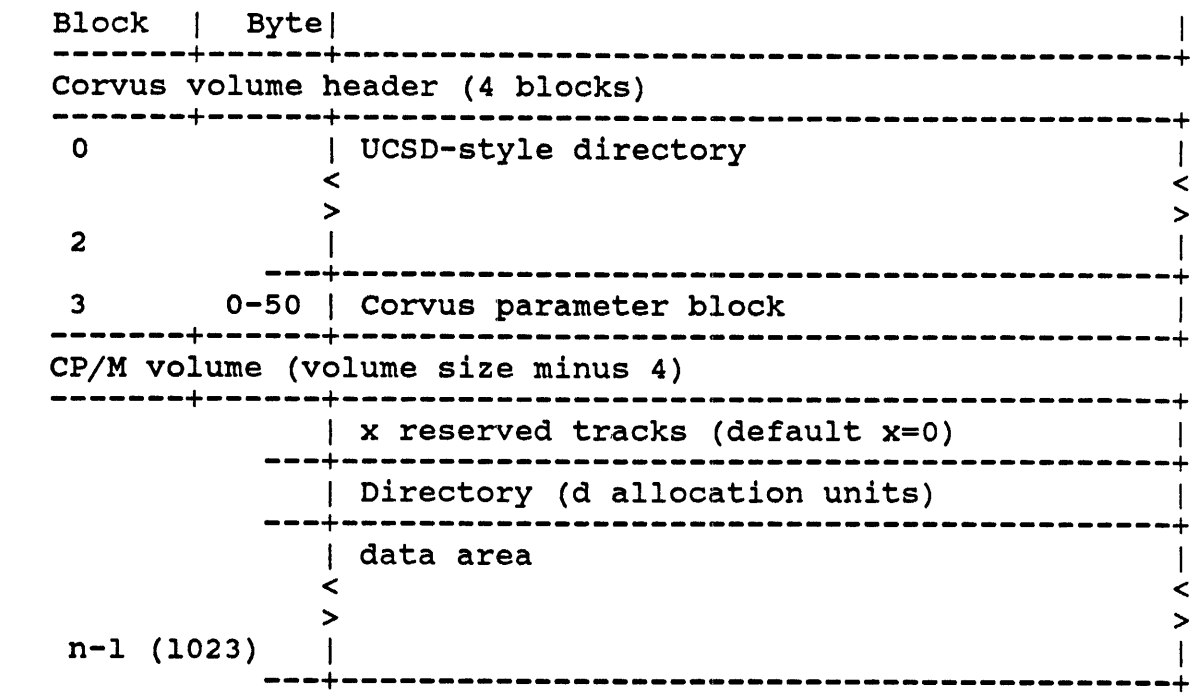
Overview

The user boots into CP/M 86 normally (i.e., from a floppy or fixed disk), and executes a Corvus supplied program, CLINK86x, which links itself into CP/M. This program intercepts all disk calls and handles the ones directed to volumes on the Corvus drive. A second program, LOGON.CRV, is used to build the user's default mount table.

Volume format

CP/M volumes on the Corvus consist of a 4 block Corvus volume header, followed by a normal CP/M volume as described in your CP/M documentation. Volume attributes are specified by the system manager when the volume is created; these include volume size, block allocation size, number of directory entries, number of sectors per track, and number of reserved tracks. The other attributes of a volume are either fixed, or computed as shown below.

CP/M Volume layout



Maximum volume size is 16388 blocks.

Corvus parameter block

The Corvus parameter block is block 3 of the volume.

0	CP/M volume header ID (4 bytes of "CP/M")	[1]
3		[4]
4	CP/M volume information (identifies system which created volume)	[1]
29		[26]
30	Volume header revision level	(lsb)
31		(msb)

CP/M disk parameter block (DPB)

32	SPT - sectors per track	(lsb)
33		(msb)
34	BSH - block shift factor	
35	BLM - block mask	
36	EXM - extent mask	
37	DSM - total storage capacity (in BLS units, minus 1)	(lsb)
38		(msb)
39	DRM - total directory entries minus 1	(lsb)
40		(msb)
41	AL0 - directory allocation flags	
42	AL1 - directory allocation flags	

(continued on next page)

Corvus parameter block
(continued from previous page)

43	CKS - directory check vector size	(lsb)
44		(msb)
45	OFF - total reserved tracks	(lsb)
46		(msb)

end of CP/M DPB

47	BLS - block allocation size	(lsb)
48		(msb)
49	TBK - total number of 512 byte blocks	(lsb)
50		(msb)

User specifes volume size, block allocation size (BLS), number of directory entries (DRM), sectors per track (SPT), and number of reserved tracks.

Maximum volume size is 16388 blocks.

Block allocation size (BLS) specifies the size of each allocation unit, and is in units of 1024 bytes. Valid values are 1k, 2k, 4k, 8k, or 16k. The default value depends on volume size:

Volume size (in blocks):	<2000	2000-7999	>=8000
Default BLS:	2k	4k	8k

In specifying number of directory entries, you should remember that each directory entry occupies 32 bytes. Also, the directory is allocated in BLS units; 1k will hold 32 directory entries. E.g., if your BLS is 8k, the number of directory entries should be a multiple of 256 (8*32).

The value specified for sectors per track (SPT) is only used to determine the amount of space to be reserved for number of reserved tracks (OFF). Normally, SPT is 64 and OFF is 0. SPT must be a multiple of 4.

The values stored in the Corvus parameter block are determined as shown below:

SPT (sectors per track) - user specified.

BSH, BLM - computed from BLS, as shown:

BLS	BSH	BLM
2	4	15
4	5	31
8	6	63
16	7	127

EXM - computed from BLS and DSM, as shown:

	EXM	
BLS	DSM<256	DSM>=256
2k	1	0
4k	3	1
8k	7	3
16k	15	7

DSM - computed from volume size and BLS, as shown:

DSM+1 = (volume size - 4 - OFF*SPT/4) * 512 / BLS
(rounded down to the nearest whole number)

DRM - user specified

AL0, AL1 - computed from DRM and BLS.

CKS - always 0, which disables checksumming of directory.

OFF - user specified

BLS - user specified

TBK - computed from volume size, as shown:

TBK = (DSM+1)*BLS/2 + OFF*SPT/4

Driver calls

<< to be supplied >>

Description of mount table

<< to be supplied >>

Memory locations used

<< to be supplied >>

Boot process

The user boots from the floppy, and installs the Corvus driver by running the program CLINK86T (CLINK86F for flat cable systems). The user then logs on by running the program LOGON.

Utilities

Utilities include a mount manager (MNTMGR.CMD), a spooler (SPOOL.CMD), and a despooler (DESPOOL.CMD).

An example program showing how to use semaphores is also provided (SEMA4.CMD, SEMA4.PAS).

Since the driver does not support CDSEND and CDRECV, the assembly language module CPMIO86.R86 is provided for this purpose. This module supports three functions:

INITIO

This function is provided for compatibility with MSDOS programs. It is a no-op for CP/M systems.

SEND (var st: LONGSTRING)
RECV (var st: LONGSTRING)

The first two bytes of the parameter are the number of bytes to be sent or received. The remainder of the bytes of the parameter are the data to be sent or the data that was received.

The file SEMA4.PAS gives examples of how these routines are called from Pascal MT86+.

B.6 Apple II DOS 3.3

Documentation:

The Apple II DOS manual
Beneath Apple DOS, by Don Worth and Pieter Lechner

Overview

Reference to the Corvus drive is accomplished by a logical mapping of the slot, drive, and volume specifiers of DOS to a volume on the Corvus. One Corvus super volume may be associated with each slot-drive combination; the volume specifier of DOS is then mapped to a DOS volume located within the Corvus super volume.

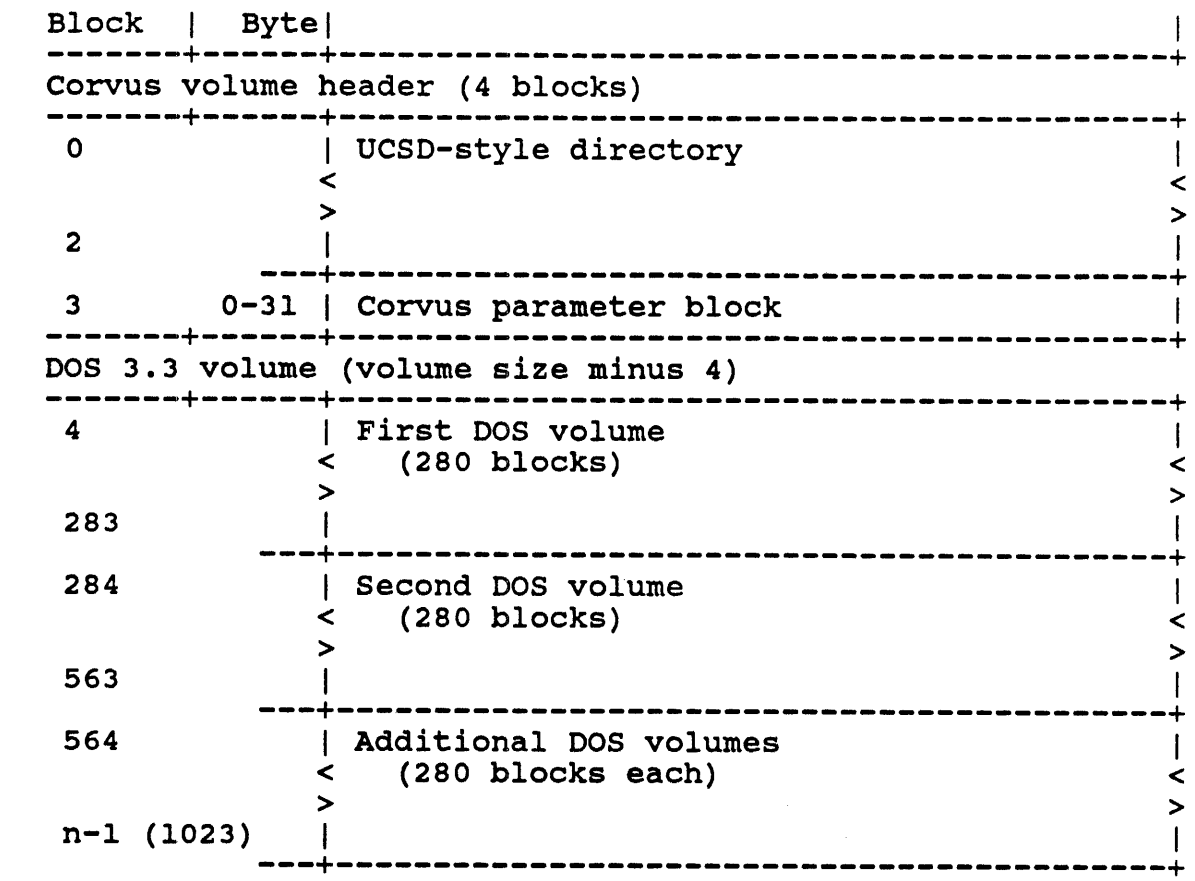
The user must boot from the Corvus drive. A DOS image is loaded from a reserved area of the Corvus drive, and is modified to use the Corvus driver which resides in RAM on the Corvus interface card.

Volume format

Constellation II uses a "super volume" to hold one or more DOS volumes. A user who has access to a super volume has access to each DOS volume within the super volume. DOS volumes within the super volume are numbered starting from 1.

The 4 block Corvus volume header is initialized, but not used by any applications.

Apple DOS 3.3 Volume layout



Each DOS volume has the format described in the DOS manual.

Directory starts at relative block 136 (track 17, sector 0) of each volume. Blocks 136 through 143 are written for directory initialization.

Corvus Volume Header
UCSD style directory blocks (blocks 0, 1, and 2)

Blocks 0 and 1 are unused. Block 2 is written with the following information:

0,1	First block number of volume	0000h
2,3	First block number of data	0300h
4,5	File kind	0000h
6	Length of volume name - user specified	xxh
7 < > 13	Volume name (7 bytes max) (first 7 bytes of user-specified volume name)	
14,15	Volume size - user specified	xxxxh
16,17	Number of files in volume	0100h
18,19	Time of last access	0000h
20,21	Date of last boot	xxxxh
22,23	Unused	0000h
24,25	Unused	0000h
26,27	First block number of file	0300h
28,29	Next block - volume size	xxxxh
30,31	File kind - data file	0500h
32	Length of file name - 10	0Ah
33 < > 47	File name (15 bytes max)	"DOS.VOLUME"
48,49	Bytes in last block - 512	0002h
50,51	Last modification date	xxxxh

Corvus Volume Header
Corvus parameter block (block 3)

3	0-3	"DOS "
	4-29	"Apple II DOS 3.3"
	30-31	Revision level
	32-511	Unused

The Corvus volume header is written on volume initialization, but is not used by any of the MSDOS utilities or drivers.

Driver calls

The Corvus driver resides in RAM on the Corvus interface card.

The Corvus driver uses the RWTS interface defined by DOS. This interface is described in the Apple DOS manual.

The first action taken by the Corvus driver is to examine the IOB. If the driver determines that the call is not for a Corvus device, then it returns control to DOS, which completes a normal RWTS call. If the call is for a Corvus device, then the driver executes the call, indicates the results by setting the field IBSTAT in the IOB, and returns control to the end of RWTS.

The following describes how the fields of the IOB are used by the Corvus driver:

IBSLOT — For Constellaion II, this value is checked against the entries in the **slot/drive array**. If it matches one of those, the call is assumed to be for Corvus.

IBDRVN — Drive number (1 or 2). For Constellation II, this value is used to find the proper entry in the **slot/drive array**.

IBVOL — If IBVOL is 0, then it is replaced by the value conained in the last volume used location. If IBVOL is not 0, then the last volume used location is updated with IBVOL.

IBTRK
IBSECT — Corvus relative sector address =
16*IBTRK + IBSECT

IBBUFP — Buffer address of where the data is read into or written from.

IBCMD — 0 - command ignored; 2 - write; any other - read.

IBSTAT — Set to 10h if volume locked; 0 if access denied; 40h for any other error. Carry is always set for any error, including access denied. Volume locked and access denied errors only occur with Constellation I and Single-user.

IBSMOD

The following fields of the IOB are not used:

IBTYPE
IBDCTP
IOBPSN
IOBPDN

Description of mount table

See next section, Memory locations used.

Memory locations used

The following memory locations are initialized on boot, and must be left unmodified. They hold information used by the Corvus

drivers. These locations are documented being used to hold DOS secondary boot code in "Beneath Apple DOS". This area is written to the boot tracks when a floppy is initialized. Hence, a floppy initialized when running Constellation II DOS is **not** bootable.

B600h through B65C

These bytes are used as follows:

Location	How used
B600h	RTS
B601h-B603h	JSR CxF4 ;where x=slot number--Corvus RWTS
B604h-B606h	JMP BD04 ;DOS RWTS
B607h-B636h	**Mount table.** See below for description.
B637h-B640h	User name (encrypted).
B641h	Network transporter address.
B642h	Address of home disk server.
B643h	Home slot number.
B644h	**Interface card bit map.** See below for description.
B645h-B64Ch	**Slot/drive array.** See below for description.
B64Bh-B650h	**Max volume array.** See below for description.
B651h-B656h	**Last volume used array.** See below for description.
B657h-B65Ch	Reserved.

The **mount table** (bytes B607h-B636h) is 48 bytes long, and contains 6 entries of 8 bytes each. Each entry specified the network location of each super volume currently mounted, in the following format:

byte 0 — Bit 7 (msb) is a write protect flag. If bit 7 is on, the volume is read-only; if bit 7 is off, the volume is read-write.

Bits 6, 5, and 4 contain a slot number. This is the number of the slot containing the interface card connected to the server containing the volume.

Bits 3, 2, 1, and 0 contain a drive number. This is the number of the drive containing the volume.

byte 1 — Server number of the server containing the volume.

bytes 2-5 — A 4-byte absolute value (msb first) containing the starting block address of the volume.

bytes 6-7 — A 2-byte absolute value (lsb first) containing the length of the volume.

The **interface card bit map** (byte B644h) is used to indicate which slots contain a Corvus interface card. Bits 6 through 0 (msb first) map to slots 7 through 1. For example, bit 6 is on if there is a card in slot 7, and bit 3 is on if there is a card in slot 4.

The **slot/drive array** (bytes B645h-B64Ah) contains 6 bytes. Each byte indicates the slot/drive reference number for the correnspondingentry in the mount table. The upper 4 bits contain the slot number of the volume, and the lower 5 bits contain the drive number. For instance, if byte 0 contains 41h, then slot 4, drive 1 refers to the first entry in the mount table. If byte 3 is 72h, then slot 7, drive 1 refers to entry number 3 in the mount table.

The **max volume array** (bytes B64Bh-B650h) contains 6 bytes. It indicates the number of DOS volumes in each super volume that is currently mounted. For instance, if byte 3 of the slot/drive array contains 42h, and byte 3 of the max volume array contains 10, then the following references are valid: S4,D2,V5; S4,D2,V10. The following reference is invalid: S4,D2,V20.

The **last volume used array** (bytes B651h-B656h) also contains 6 bytes. It indicates the last volume accessed on each slot/drive entry. For instance, if you reference S4,D1,V6, then the entry

that corresponds to 41h will be set to 6. The next time you reference S4,D1, the driver will automatically access volume 6.

It is possible to make a wholesale replacement of the DOS image in memory after booting Constellation II. Care must be taken to patch the appropriate memory locations described below, and the values of the B600 variables must be preserved during I/O operations. For example, these values may be moved by the application and the locations used for other data during non I/O operations. Replacement of the DOS image may, however, disable operation of the Corvus Constellation II utilities MOUNT MANAGER, SPOOL, and DESPOOL.

The Corvus boot code patches DOS so that all RWTS calls are sent to the Corvus driver. This patch is described below.

Location 3DBh contains the high order byte of the start of the RWTS subroutine. The Corvus patch is actually installed at a subroutine located 600h past the start of the RWTS subroutine. For instance, in standard DOS 3.3, location 3DBh contains the value B7h, and the Corvus patch is installed at location B700h+600h or BD00h. For Constellation I, the patch consists of overwriting location BD00h through BD02h with a JMP instruction to the Corvus interface card. For Constellation II, the destination of the JMP instruction is memory location B601.

Boot process

After booting, Constellation II looks for a HELLO program first on drive 1 of the highest logical slot mounted, and then on drive 2 of this slot. If a HELLO program is found, it is loaded and run; otherwise the Applesoft prompt is displayed.

Utilities

<< to be supplied >>