FOR THE FAMILY OF
APPLE II COMPUTERS

REQUIRES 64K
AND ONE DISK DRIVE

Universal File Conversion

By Gary Charpentier

A Program To Convert Files From One Operating System To Another

A utility program that allows the user to . . .

- Move data between DOS 3.3, CP/M, Apple Pascal, SOS, and ProDOS
- Move programs between DOS 3.3, CP/M, Apple Pascal, SOS, and ProDOS
- Format disks as DOS 3.3, CP/M, Apple Pascal, SOS, or ProDOS
- Create CP/M files without a Softcard
- Create SOS files without an Apple III
- Copy files from one disk to another
- Learn how each operating system stores files
- Learn the specific format of each file type

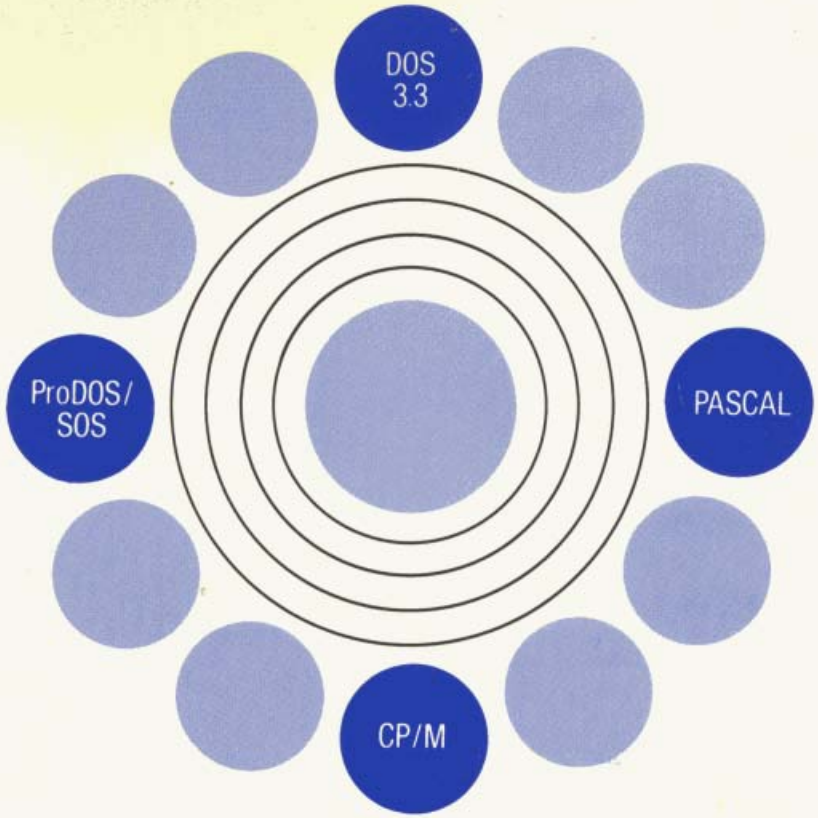
Universal File Conversion

QS QUALITY SOFTWARE
Computer Book Division

ISBN 0-912985-06-2 > $34.95

PRINTED IN USA

Universal File Conversion



A Program To Convert Files From One Operating System To Another

By Gary Charpentier

FOR THE FAMILY OF APPLE II COMPUTERS

QS QUALITY SOFTWARE
Computer Book Division

Universal File Conversion

A Program To Convert Files From One Operating System To Another

By Gary Charpentier

QS QUALITY SOFTWARE
21601 Marilla Street
Chatsworth, California 91311

Apple Books from Quality Software

Beneath Apple DOS by Don Worth and Pieter Lechner	$19.95
Beneath Apple ProDOS by Don Worth and Pieter Lechner	$19.95
Understanding the Apple II by Jim Sather	$22.95
Understanding the Apple IIe (Avail. 11-84) by Jim Sather	$24.95

Apple Utility Software from Quality Software

Bag Of Tricks (includes diskette) by Don Worth and Pieter Lechner	$39.95

"Apple" is a registered trademark of Apple Computer, Inc. This manual was not prepared nor reviewed by Apple Computer, Inc., and use of the term "Apple" should not be construed to represent any endorsement, official or otherwise, by Apple Computer, Inc.

International Standard Book Number: 0-912985-06-2

86 85 84 5 4 3 2 1

Printed in the United States

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Have you ever wanted to take a word processing file that was written using WordStar and read it using a DOS-based word processor like Apple Writer? Or maybe you have a data file generated by a Pascal program that you would like to access with an Applesoft program? In both of these cases you are faced with the difficult and tricky task of transferring a file from one operating system to another.

Universal File Conversion will copy standard Apple II disk files from any one of the four different operating systems available for the Apple II to another diskette that uses a different operating system. Universal File Conversion can also simply copy files between two diskettes in the same operating system. There are four different operating systems currently in use on the Apple II computer: Pascal, DOS 3.3, ProDOS, and CP/M. Figure 1.1 shows all the possible directions for moving a file. Apple III files, which use SOS, may also be transferred with Universal File Conversion, because there is complete interchange between SOS diskettes and ProDOS diskettes, even though the file and volume structures have minor differences. All types of files may be moved, with the exception of "bad block" and "directory" files. In addition, Universal File Conversion can format a disk for any of the four operating systems.

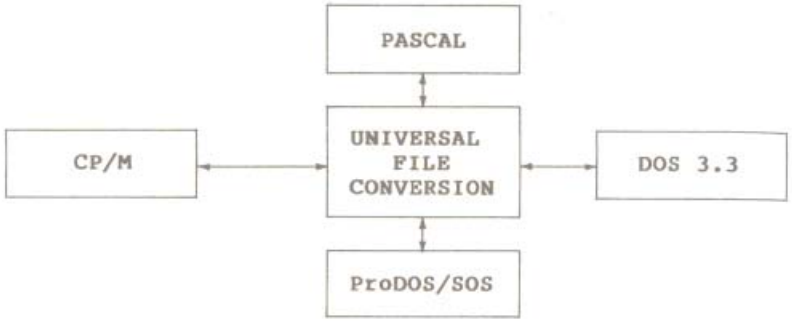


Figure 1.1 Files Can Be Moved To and From Any of Four Operating Systems

HARDWARE REQUIREMENTS

Universal File Conversion requires an Apple II, Apple II Plus, Apple IIc, or Apple IIe computer with 64K of memory and at least one Disk II drive. However, two or more disk drives are recommended to speed copying and avoid an excess of disk swapping.

Standard Apple Disk II formatting is the only type of formatting supported. In other words, other types of disk drives are not supported. These include 8" floppies, hard disks, RAM disks, and even other 5 1/4" floppies. This is not a serious limitation if you have at least one Disk II drive, because most modified operating systems include software that read and write to a standard Disk II drive. Simply transfer the file to a standard 5 1/4" floppy before using Universal File Conversion.

Copy protected diskettes cannot be used with Universal File Conversion.

CONVERTING FILES

When a text file is being converted, Universal File Conversion makes all changes necessary for the target operating system. These changes include setting or clearing the high bit, adding or removing line feeds, adding or removing the two block header for the Pascal editor, expanding or generating space compression codes, etc.

If the file is not a text file, Universal File Conversion makes no changes, with the following exceptions: it reserves space for or removes the address and length of binary files or the length of Applesoft and Integer BASIC files.

Universal File Conversion is a powerful and helpful utility, but merely moving a file from one operating system to another may not be sufficient to begin using that file in a different operating environment. For example, Universal File Conversion can move BASIC programs that are text files from one system to another, but it will not change an Applesoft tokenized program into an MBASIC tokenized program, nor perform other similar changes. It will not properly convert random access files. It will not translate machine language programs from one language to another.

Universal File Conversion is not designed to delete files or write over existing files. These functions can be performed using the normal RENAME and DELETE commands provided by the operating system in question once files have been transferred using Universal File Conversion.

CHAPTER 2

USING THE PROGRAM

BOOTING THE DISK

Universal File Conversion is self-booting. Simply place the diskette in drive 1 (slot 6) and turn on your computer. Or type PR#6 if your computer is already up and running. The Main Menu should appear on the screen in a few seconds.

The Universal File Conversion program comes on a diskette that is in Pascal format. It is not copy protected, and may be copied using COPYA on your DOS Master Disk or the ProDOS Filer utility. Beyond a reasonable number of copies for the buyer's personal use for backing up the original, copying this program is a violation of the copyright laws of the United States and other countries.

SETTING SLOT, DRIVE, AND SYSTEM TYPE

Press "N" from the Main Menu to change slot/drive assignments. The current assignments are printed at the bottom of the screen when the Main Menu is displayed, and during the assignment process.

The "source" slot/drive is where you will put the diskette with the **existing** program. You want to **read from** this diskette.

The "destination" slot/drive is where you will put the diskette you want the file transferred to. You want to **write to** this diskette.

Typing "E" allows you to "exit" or "escape" from what you have been asked to enter.

If you select "SOS/PRODOS" as a system type, you will be asked to enter a PATH. Pressing RETURN will automatically select the pathname of the current volume. The name designated here will be used as the default pathname when copying files to or from a SOS or ProDOS diskette. If copying from one ProDOS diskette to another, different pathnames can be specified for source and destination.

VIEWING THE CATALOG OR DIRECTORY

Type "V" from the Main Menu to look at the catalog or directory of a diskette.

You will be asked to select either the source or destination slot/drive. The program will expect the diskette in the selected drive to match the system type specified at the bottom of the screen. If the system type doesn't match, an error message will be displayed. Otherwise, the catalog or directory will be displayed.

Press any key after you have finished reviewing the directory to return to the previous menu.

COPYING FILES

Press "C" from the Main Menu to copy files.

The Universal File Conversion diskette must be in drive 1 (slot 6) so that the copy routines can be read in. If the program does not find it there when you press "C", you will be asked to put it in. Similarly, when you have finished copying files, you will be asked to put the Universal File Conversion diskette in drive 1 (slot 6) before returning to the Main Menu.

CAUTION: You cannot RENAME or DELETE files using Universal File Conversion. If you want a new filename to be the same as an old filename on the disk, first rename or delete files using standard operating system software, then run Universal File Conversion.

You may press the ESC key to exit the copy procedure and return to the Main Menu.

Enter the filename just as you would under the source operating system. The name may be changed slightly, depending on the destination operating system. See Chapter 4.

When the source and/or destination is ProDOS or SOS, the pathname used is the pathname assigned when the drive parameters were specified (using the N command from the Main Menu).

FORMATTING A DISK

Type "F" from the Main Menu to format a disk. You must have previously assigned the operating system you want formatted to either the source or destination drive (usually destination). This assignment is done using the "N" command from the Main Menu.

You will be asked to select the source or destination drive for formatting. A blank diskette should be placed in the appropriate drive before making this selection.

If the program finds that the diskette in the selected drive is already formatted (standard Apple II format), it will ask if you want to erase the existing data. Otherwise, it will immediately proceed to format the diskette.

If you view a catalog or directory of a diskette formatted by Universal File Conversion, it will be empty. If you try to boot the diskette, you will get a brief message saying that the diskette has no DOS on it. In the case of DOS 3.3, space is reserved for the DOS image even though no DOS is written to the diskette. DOS can be later written to the diskette using the MASTER CREATE program that comes with the disk system.

ERROR MESSAGES

Message	Meaning
Disk is wrong DOS	The disk accessed has the wrong operating system type.
Drive too fast	
Drive too slow	
Error in Filename	The file name is not acceptable. It may have an illegal character.
ERROR = #	System error (should not occur).
I/O Error	
NO BOOT.DATA	The program can't find BOOT.DATA which is on the Universal File Conversion diskette.
No directory space on destination	

Message	Meaning
NO DOS.DATA	The program can't find DOS.DATA, which is on the Universal File Conversion diskette.
NO PASCAL.DATA	The program can't find PASCAL.DATA, which is on the Universal File Conversion diskette.
NO SOS.DATA	The program can't find SOS.DATA, which is on the Universal File Conversion diskette.
No such disk	You don't have a disk drive connected to the disk controller card for the specifed slot and drive.
Not enough room on destination disk	
Only one wild-card character	More than one equal sign was typed in the filename.
OOPS -- MEMORY	System error (should not occur).
Path not found	The directory you indicated (ProDOS or SOS) does not exist.
RUNTIME ERROR ERROR NUMBER =	System error (should not occur).
UNABLE TO LOAD FORMATTER.CODE	An error occurred loading FORMATTER.CODE -- either an I/O error or the program can't find FORMATTER.CODE.
Write protected	

CHAPTER 3

DISKETTE ORGANIZATION

Each operating system divides the diskette into 560 sectors of 256 bytes each. The diskette has 35 tracks with 16 sectors per track. This chapter explains how each operating system organizes these physical sectors to create a directory and a file structure.

All operating systems order tracks the same--namely, track 0 is the outermost track on the diskette and track 34 is the innermost track. The ordering of sectors within tracks is a different story.

In an attempt to maximize the speed at which information can be transferred to and from the diskette, different operating systems order the 16 sectors on each track differently. Sector ordering can be a confusing subject, so we will spend a little time explaining it.

The easiest order to understand is what we call the **physical sector** order. This is simply the order in which the sectors are placed around the disk. The physical position of each sector may be thought of as the **physical order number**, although this number is not written down anywhere. The physical order number may differ from the number in the address field of the sector. We will call the sector number in the address field the **sector ID number**. Some older operating systems and some "fast DOS" utilities actually change the order of the sector ID numbers so that they are different than the physical order number. Fortunately, the standard versions of the operating systems that Universal File Conversion is involved with (DOS 3.3, Pascal, ProDOS, SOS, and CP/M) all format diskettes with the physical order and the sector ID order the same. However, each operating system translates the sector ID number to a **logical sector number**, and it is this logical sector number which is the sector used by the software for disk accesses.

Figure 3.1 shows the relationship between physical (or sector ID) number and the logical sector number for each of the four operating systems. Note that the sectors called out later in Figures 3.2, 3.6, 3.12, and 3.20 are the logical sector numbers for the referenced operating system, and not necessarily the sector ID numbers.

PHYSICAL SECTOR	LOGICAL SECTOR			
	DOS 3.3	PASCAL	SOS/ProDOS	CP/M
0	0	0	0	0
1	7	8	8	B
2	E	1	1	6
3	6	9	9	1
4	D	2	2	C
5	5	A	A	7
6	C	3	3	2
7	4	B	B	D
8	B	4	4	8
9	3	C	C	3
A	A	5	5	E
B	2	D	D	9
C	9	6	6	4
D	1	E	E	F
E	8	7	7	A
F	F	F	F	5

Figure 3.1 Sector Order Table for All Operating Systems

APPLE PASCAL

The Apple Pascal disk is laid out in blocks of two sectors each. Each block is 512 bytes long. The first six blocks on the disk are reserved. See Figure 3.2 for the layout of the Apple Pascal disk. Blocks 0 and 1 are reserved for the bootstrap loader. Blocks 2 through 5 are the reserved for the directory of the disk. The rest of the disk, blocks 6 to 279, is usable for storing files.

The Apple Pascal directory is structured as shown in Figure 3.3. Because Apple Pascal uses a 16-bit pseudo-code, the directory is laid out mostly on word boundaries.

The Pascal directory has two parts. The first part is the **directory header**, which consists of 26 bytes. The second part is a list of up to 77 **file entries**. Each file entry also consists of 26 bytes.

TRACK NUMBER	SECTOR NUMBER															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	BOOT PROG				DIRECTORY								BLK 6			7
1	8		9		A		B		C		D		E			F
2	10		11		12		13		14		15		16			17
3	18		19		1A		1B		1C		1D		1E			1F
4	20		21		22		23		24		25		26			27
5	28		29		2A		2B		2C		2D		2E			2F
6	30		31		32		33		34		35		36			37
7	38		39		3A		3B		3C		3D		3E			3F
8	40		41		42		43		44		45		46			47
9	48		49		4A		4B		4C		4D		4E			4F
A	50		51		52		53		54		55		56			57
B	58		59		5A		5B		5C		5D		5E			5F
C	60		61		62		63		64		65		66			67
D	68		69		6A		6B		6C		6D		6E			6F
E	70		71		72		73		74		75		76			77
F	78		79		7A		7B		7C		7D		7E			7F
10	80		81		82		83		84		85		86			87
11	88		89		8A		8B		8C		8D		8E			8F
12	90		91		92		93		94		95		96			97
13	98		99		9A		9B		9C		9D		9E			9F
14	A0		A1		A2		A3		A4		A5		A6			A7
15	A8		A9		AA		AB		AC		AD		AE			AF
16	B0		B1		B2		B3		B4		B5		B6			B7
17	B8		B9		BA		BB		BC		BD		BE			BF
18	C0		C1		C2		C3		C4		C5		C6			C7
19	C8		C9		CA		CB		CC		CD		CE			CF
1A	D0		D1		D2		D3		D4		D5		D6			D7
1B	D8		D9		DA		DB		DC		DD		DE			DF
1C	E0		E1		E2		E3		E4		E5		E6			E7
1D	E8		E9		EA		EB		EC		ED		EE			EF
1E	F0		F1		F2		F3		F4		F5		F6			F7
1F	F8		F9		FA		FB		FC		FD		FE			FF
20	100		101		102		103		104		105		106			107
21	108		109		10A		10B		10C		10D		10E			10F
22	110		111		112		113		114		115		116			117

Figure 3.2 Pascal Disk Layout by Block Number

The directory header is in the first block of the directory, block 2. In the directory header, bytes $00 and $01 are an integer representing the number of the first block on the disk (always zero). Bytes $02 and $03 are the integer number of the last block in the directory plus one. Bytes $04 and $05 represent the type of entry for this record; in this case it is zero, meaning a directory. Bytes $07 through $0D are a string representing the volume name. Bytes $0E and $0F represent the number of blocks on this disk, 280. Bytes $10 and $11 represent the number of files currently on this disk. Bytes $14 and $15 are the date the disk was created. Figure 3.4 shows how the date is encoded.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	FIRST		LAST		TYPE		VLN	VOLUME NAME							LT BK	
010	#FILE		TIME		DATE		RESERVED				START		END		TYPE	
020	LEN	FILE NAME					FILE 1									
030	#BYTE	DATE					FILE 2									
040																
050							FILE 3									
060																
070							FILE 4									
080																
090							FILE 5									
0A0																
0B0							FILE 6									
0C0																
0D0							FILE 7									
0E0																
0F0							FILE 8									

700	F68	
710		FILE 69
720		FILE 70
730		FILE 71
740		
750		FILE 72
760		
770		FILE 73
780		FILE 74
790		
7A0		FILE 75
7B0		FILE 76
7C0		
7D0		FILE 77
7E0		
7F0		NOT USED

```
FIRST    = FIRST BLOCK OF DISK
LAST     = LAST BLOCK OF DIRECTORY PLUS 1
TYPE     = TYPE OF FILE
VLEN     = LENGTH OF VOLUME NAME
LT BK    = LAST BLOCK OF DISK PLUS 1
#FILE    = NUMBER OF FILES
START    = FIRST BLOCK OF FILE
END      = LAST BLOCK OF FILE PLUS 1
#BYTE    = NUMBER OF BYTES IN LAST BLOCK
LEN      = LENGTH OF FILE NAME
```

Figure 3.3 Pascal Directory

FIRST BYTE								SECOND BYTE							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
MONTH				DAY				YEAR							

Figure 3.4 Pascal Date Encoding

The last four bytes of the directory header, $16 through $19, are not used (they extend the directory header to the same length as a file entry).

Now for a file entry. Relative bytes $00 and $01 of a file entry give the starting block number for the file. Bytes $02 and $03 give the ending block plus one for the file. Bytes $04 and $05 give the type of file. The file types are shown in Figure 3.5. Bytes $06 through $15 are a string giving the name of the file. Bytes $16 and $17 tell how many bytes in the last block of the file are used. Bytes $18 and $19 are the date the file was last modified. The modification date has the same format as in the header (shown in Figure 3.4).

```
0 - UNTYPED
1 - BAD
2 - CODE
3 - TEXT
4 - INFO
5 - DATA
6 - GRAF
7 - FOTO
8 - SECURDIR
```

Figure 3.5 Pascal File Types

DOS 3.3

A DOS 3.3 disk is laid out as shown by Figure 3.6. DOS 3.3 is the only operating system which uses the sector as the smallest unit for allocation of a file. Tracks $00, $01, and $02 are reserved for the DOS image if present. Track $11 is reserved for the volume table of contents and the directory.* The rest of the disk is available for storing files.

The **volume table of contents** or **VTOC** is shown in Figure 3.7. It is located at track $11 sector $00. Bytes $01 and $02 are a link to the first sector of the directory. The only other information of interest in the VTOC are the track bit maps.

*DOS refers to its directory as the "catalog," but we will use the term "directory" even for DOS, because all other operating systems use "directory."

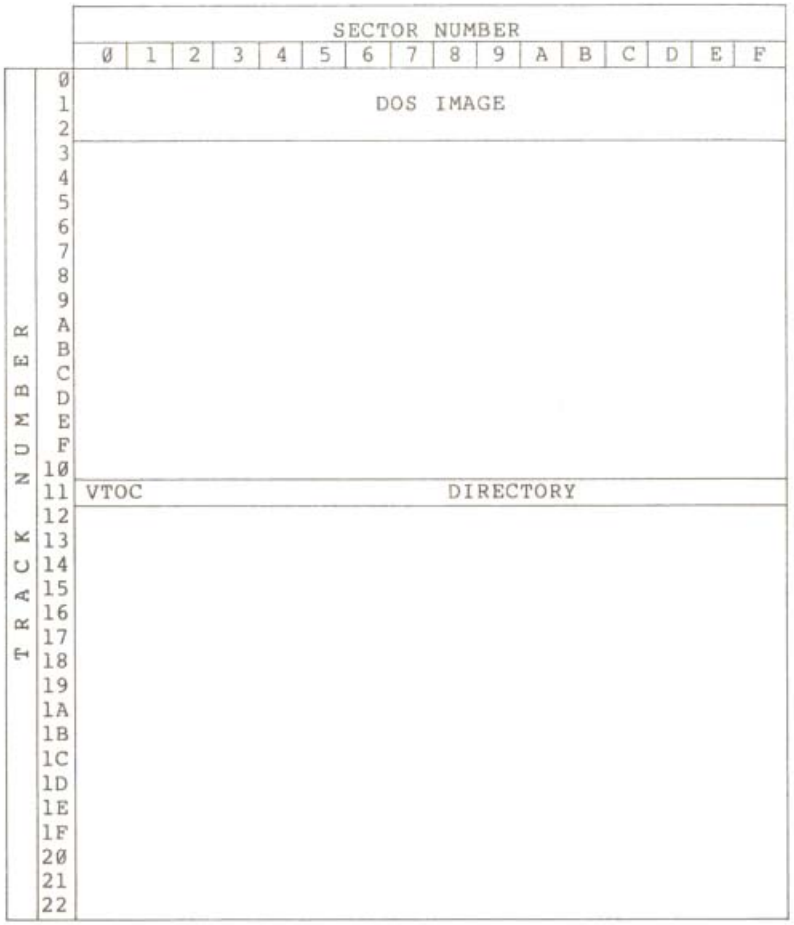
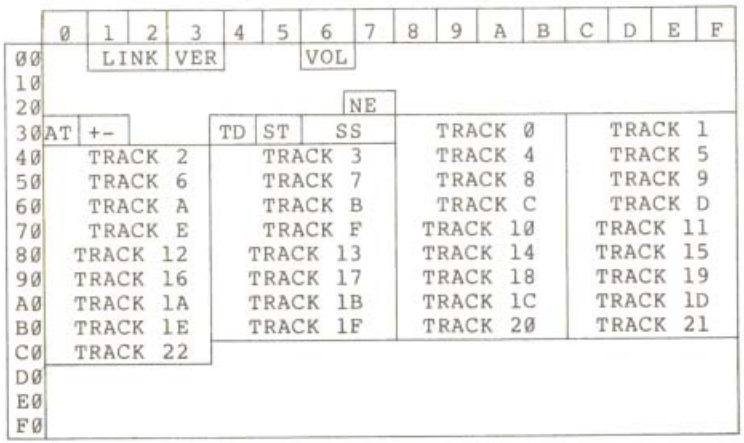


Figure 3.6 DOS 3.3 Disk Layout

For each track on the disk, there is a map showing which sectors are available for use or are already being used. Each track bit map consists of four bytes, but only the first two bytes are used in 16-sector drives. Figure 3.8 shows the correspondence of the bits in each byte to the sectors on a track.

The rest of track $11 is reserved for the **DOS directory** (catalog), which starts at sector $0F and continues onto descending sectors if necessary. It is shown in Figure 3.9. Each sector of the directory has this format. Bytes $01 and $02 point to the next sector if there is one. If there are no more sectors in the directory, then both bytes of the pointer are zero.




```
LINK    = LINK TO FIRST SECTOR OF DIRECTORY
VER     = DOS VERSION
VOL     = VOLUME
NE      = # OF ENTRIES IN TS LIST
AT      = LAST TRACK ALLOCATED
+-      = DIRECTION OF TRACK ALLOCATION
TD      = TRACKS / DISK
ST      = SECTORS / TRACK
SS      = SECTOR SIZE
TRACK N = BIT MAP FOR TRACK N
```

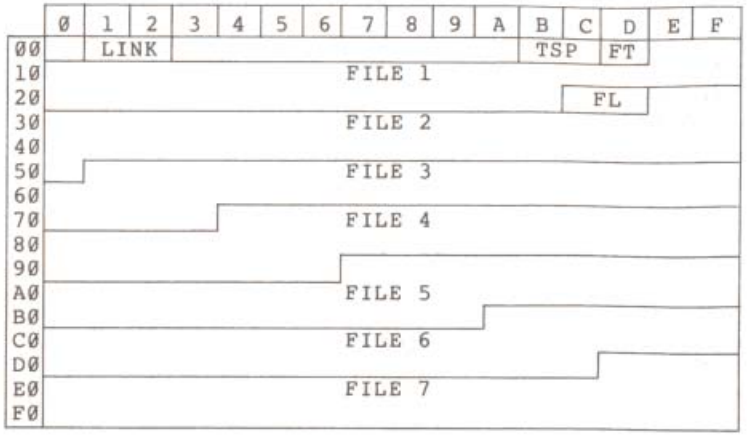
Figure 3.7 DOS 3.3 Volume Table of Contents (VTOC)

BYTE	0	1	2	3
BIT	76543210	76543210	76543210	76543210
SECTOR	FEDCBA98	76543210	UNUSED	UNUSED

Figure 3.8 DOS 3.3 Bit Map for One 16-Sector Track

For each file entry, bytes $00 and $01 point to the beginning of the track sector list for that file. Byte $02 is the file type. The file types are shown in Figure 3.10. The file name occupies bytes $03 through $20. Bytes $21 and $22 are the length of the file in sectors.

The format of each sector of the **track sector list** is shown in Figure 3.11. Bytes $01 and $02 are a link to the next sector, if any, of the track sector list. If there is no next sector, the pointer bytes are zero. Bytes $05 and $06 are the starting sector number of this track sector list. A list of the actual sectors in the file starts at $0C. There can be at most 122 sectors of a file in each track sector list.



LINK = LINK TO NEXT SECTOR OF DIRECTORY
TSP = TRACK SECTOR POINTER
FT = FILE TYPE
FL = FILE LENGTH

Figure 3.9 DOS 3.3 Directory

```
00 - TEXT
01 - INTEGER BASIC
02 - APPLESOFT
04 - BINARY
08 - S
10 - RELOCATABLE BINARY
20 - A
40 - B
```

Figure 3.10 DOS 3.3 File Types

ProDOS AND SOS

ProDOS and SOS are somewhat similar to DOS 3.3 in that they both use a bit map to tell which areas of the disk are in use. The index blocks of ProDOS and SOS could be likened to the track sector lists of DOS 3.3 also, but here the similarity ends. ProDOS and SOS allow directories as files and DOS 3.3 has nothing like that.

Figure 3.12 shows the layout of a ProDOS or SOS disk. ProDOS and SOS use 512-byte blocks of two sectors each, as Apple Pascal does. Blocks 0 and 1 are reserved for the bootstrap loader. Blocks 2 through 5 are the volume directory of the disk. Block 6 is the bit map for the disk. The rest of the disk, blocks 7 through 279, is available to store files.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00		LINK				OFF							S0		S1	
10	S2		S3		S4		S5		S6		S7		S8		S9	
20	SA		SB		SC		SD		SE		SF		S10		S11	
30	S12		S13		S14		S15		S16		S17		S18		S19	
40	S1A		S1B		S1C		S1D		S1E		S1F		S20		S21	
50	S22		S23		S24		S25		S26		S27		S28		S29	
60	S2A		S2B		S2C		S2D		S2E		S2F		S30		S31	
70	S32		S33		S34		S35		S36		S37		S38		S39	
80	S3A		S3B		S3C		S3D		S3E		S3F		S40		S41	
90	S42		S43		S44		S45		S46		S47		S48		S49	
A0	S4A		S4B		S4C		S4D		S4E		S4F		S50		S51	
B0	S52		S53		S54		S55		S56		S57		S58		S59	
C0	S5A		S5B		S5C		S5D		S5E		S5F		S60		S61	
D0	S62		S63		S64		S65		S66		S67		S68		S69	
E0	S6A		S6B		S6C		S6D		S6E		S6F		S70		S71	
F0	S72		S73		S74		S75		S76		S77		S78		S79	

LINK = T/S OF NEXT T/S LIST SECTOR IF DATA SECTORS EXIST BEYONE 79
OFF = SECTOR OFFSET INTO FILE OF FIRST DATA SECTOR LISTED
S0 = T/S OF DATA SECTOR 0 OR 0000 IF THIS SECTOR WAS NOT WRITTEN
S1 = etc.

Figure 3.11 DOS 3.3 Track Sector List

The first block of the **volume directory** is shown in Figure 3.13. The other blocks of the directory are similar except that the volume header information is replaced by a file entry. Relative bytes $00 and $01 of the directory block point to the previous block, if any, in the directory. Bytes $02 and $03 point to the next block, if any.

The first byte of the volume header has two uses. The first four bits (high nibble) indicate the storage type, in this case $F for a volume directory header. The storage types are shown in Figure 3.14.

The last four bits (low nibble) of the the first byte of the volume header are the number of characters in the volume name. Bytes $05 through $13 contain the volume name. Eight reserved bytes, $14 through $1B, are next. Bytes $1C and $1D are the creation date. Bytes $1E and $1F are the creation time. The encoding for the date and time is shown in Figure 3.15.

Bytes $20 and $21 are for checking version compatibility. Byte $22 is the access byte. The meaning of each bit of the access byte is shown in Figure 3.16.

TRACK NUMBER	SECTOR NUMBER 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	BOOT				DIRECTORY								BITMAP		7	
1	8		9		A		B		C		D		E		F	
2	10		11		12		13		14		15		16		17	
3	18		19		1A		1B		1C		1D		1E		1F	
4	20		21		22		23		24		25		26		27	
5	28		29		2A		2B		2C		2D		2E		2F	
6	30		31		32		33		34		35		36		37	
7	38		39		3A		3B		3C		3D		3E		3F	
8	40		41		42		43		44		45		46		47	
9	48		49		4A		4B		4C		4D		4E		4F	
A	50		51		52		53		54		55		56		57	
B	58		59		5A		5B		5C		5D		5E		5F	
C	60		61		62		63		64		65		66		67	
D	68		69		6A		6B		6C		6D		6E		6F	
E	70		71		72		73		74		75		76		77	
F	78		79		7A		7B		7C		7D		7E		7F	
10	80		81		82		83		84		85		86		87	
11	88		89		8A		8B		8C		8D		8E		8F	
12	90		91		92		93		94		95		96		97	
13	98		99		9A		9B		9C		9D		9E		9F	
14	A0		A1		A2		A3		A4		A5		A6		A7	
15	A8		A9		AA		AB		AC		AD		AE		AF	
16	B0		B1		B2		B3		B4		B5		B6		B7	
17	B8		B9		BA		BB		BC		BD		BE		BF	
18	C0		C1		C2		C3		C4		C5		C6		C7	
19	C8		C9		CA		CB		CC		CD		CE		CF	
1A	D0		D1		D2		D3		D4		D5		D6		D7	
1B	D8		D9		DA		DB		DC		DD		DE		DF	
1C	E0		E1		E2		E3		E4		E5		E6		E7	
1D	E8		E9		EA		EB		EC		ED		EE		EF	
1E	F0		F1		F2		F3		F4		F5		F6		F7	
1F	F8		F9		FA		FB		FC		FD		FE		FF	
20	100		101		102		103		104		105		106		107	
21	108		109		10A		10B		10C		10D		10E		10F	
22	110		111		112		113		114		115		116		117	

Figure 3.12 ProDOS/SOS Disk Layout by Block Number

Byte $23 is the entry length of each entry in this directory. Byte $24 is the number of entries in each block of the directory. Bytes $25 and $26 are the number of files in the directory. Bytes $27 and $28 are a pointer to the volume bit map. Bytes $29 and $2A are the total number of blocks in the volume.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	P.PNT			N.PNT		ST NL	VOLUME NAME									
10	VOLUME NAME								RESERVED						C.DATE	C.TIME
20	VER	MVER	ACC	EL	EPB	FILECNT			BITMPNT			TOTBLK			ST NL	FILE NAME
30	FILE NAME													TYPE	KEY PNT	#BLKS
40	EOF				C.DATE		C.TIME		VER	MVER	ACC	AUXTYPE			M.DATE	M.TIME
50	HPNT															
60																
70																
80																
90																
A0																
B0																
C0																
D0																
E0																
F0																
100																
110																
120																
130																
140																
150																
160																
170																
180																
190																
1A0																
1B0																
1C0																
1D0																
1E0																
1F0																NUSD

```
P.PNT    = PREVIOUS BLOCK OF DIRECTORY
N.PNT    = NEXT BLOCK OF DIRECTORY
ST       = STORAGE TYPE
NL       = NAME LENGTH
C.DATE   = CREATION DATE
C.TIME   = CREATION TIME
VER      = VERSION
MVER     = MINIMUM VERSION
ACC      = ACCESS
EL       = ENTRY LENGTH
EPB      = ENTRIES PER BLOCK
FILECNT  = NUMBER OF FILES
BITMPNT  = POINTER TO THE BIT MAP
TOTBLK   = NUMBER OF BLOCKS ON DISK
TYPE     = TYPE OF FILE
KEY PNT  = POINTER TO FILE
#BLKS    = NUMBER OF BLOCKS IN FILE
AUXTYPE  = AUXILIARY TYPE
M.DATE   = MODIFICATION DATE
M.TIME   = MODIFICATION TIME
HPNT     = POINTER TO START OF DIRECTORY
NUSD     = NOT USED
```

Figure 3.13 ProDOS/SOS Directory Header Block

```
0 - DELETED ENTRY
1 - SEEDLING
2 - SAPLING
3 - TREE
D - SUBDIRECTORY FILE
E - SUBDIRECTORY HEADER
F - VOLUME DIRECTORY HEADER
```

Figure 3.14 ProDOS/SOS Storage Types

BYTE $1C								BYTE $1D							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
YEAR								MONTH			DAY				

BYTE $1E								BYTE $1F							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	0	HOUR					0	0	MINUTE					

Figure 3.15 ProDOS/SOS Date and Time Encoding

7	6	5	4	3	2	1	0
DELETE ENABLE	RENAME ENABLE	BACKUP	RESERVED			WRITE ENABLE	READ ENABLE

Figure 3.16 ProDOS/SOS Access Byte Flags

A file entry is almost identical to the volume entry. The first byte of the file entry, $00, is again divided into the storage type and the length of the file name. The file name is again 15 bytes in length. The bytes that were reserved in the volume header take on meaning in a file entry. Byte $10 is the file type. The file types are shown in Figure 3.17.

Bytes $11 and $12 point to the key block of the file. Bytes $13 and $14 give the length of the file in blocks. Bytes $15 through $17 are the logical length of the file in bytes. The creation date and time, version bytes, and access type are next as they were in the volume header. Bytes $1F and $20 are the file's auxiliary type. Bytes $21 and $22 are the modification date. Bytes $23 and $24 are the modification time. Bytes $25 and $26 are the "head pointer," indentifying the starting block of the directory.


```
  0 - Untyped
  1 - Bad
  2 - Pascal Code
  3 - Pascal Text
  4 - Text
  5 - Pascal Data
  6 - Binary
  7 - Font
  8 - Foto
  9 - Business BASIC
 10 - Business BASIC Data
 11 - Word Processor File
 12 - SOS System

 15 - Directory
 16 - RPS Data
 17 - RPS Index

 25 - AppleWorks Data Base File
 26 - AppleWorks Word Processing File
 27 - AppleWorks Spreadsheet File

239 - ProDOS Pascal File
240 - Command

250 - Integer BASIC
251 - Integer Variable
252 - Applesoft BASIC
253 - Applesoft Variable
254 - Relocatable Binary
255 - ProDOS System
```

Figure 3.17 ProDOS/SOS File Types

An example of the **ProDOS/SOS bit map** is shown in Figure 3.18. As discussed earlier, the bit map is usually block 6 of the volume in question. One bit of the bit map is on (equal to 1) for every block that is "free" for use. For a standard diskette, only 280 blocks are available. Thus, only the first 35 bytes of the bit map are used. The remaining (irrelevant) bytes are set to zero. For hard disks or other volumes that have more than 4096 blocks, the bit map carries over into the immediately following blocks.

ProDOS and SOS have four different methods of storing files. These four methods are subdirectory files, seedling files, sapling files, and tree files. The storage method is dictated by the storage type (see Figure 3.14). Figure 3.19 describes the four storage methods.

Subdirectory files can be any number of blocks in length. Each block in the file has pointers which point forward and backward to the next and previous blocks in the file.

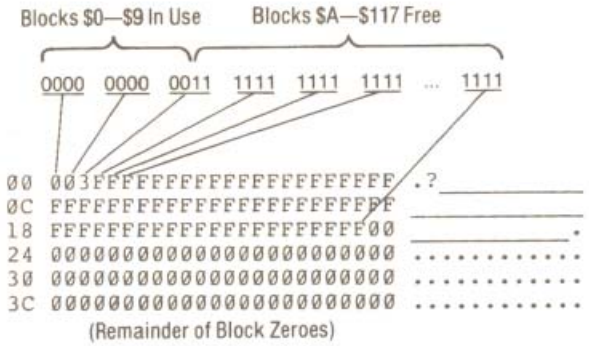


Figure 3.18 ProDOS/SOS Volume Bit Map*

*Reprinted with permission from Beneath Apple ProDOS, by Don Worth and Pieter Lechner (Quality Software, 1984).

Seedling files are the easiest to understand. The pointer from the directory points directly to the only block of data in the file.

Sapling files have a single index block. The pointer from the directory points to the index block. The index block has 256 pointers to the actual data blocks of the file. The pointers are stored in an odd fashion. The high byte of the pointer is 256 bytes after the low byte. All pointers in the index block that are not needed are set to zero.

Tree files have two levels of index blocks. The index block the directory points to is called the master index block. It points to 128 more index blocks. The other index blocks are just like the sapling file's index block. Again, pointers that are not needed are set to zero.

CP/M

CP/M uses a very different file structure than the other operating systems. It helps to remember that it was designed for eight inch floppy disks that had twenty six 128-byte sectors per track. This is very different than the Disk II's sixteen 256-byte sectors per track.

The layout of a CP/M disk is shown in Figure 3.20. CP/M uses blocks of 1024 bytes or four sectors. However, block zero does not start at track zero, sector zero, but at track three, sector zero. Tracks 0, 1, and 2 are reserved for the CP/M operating system. Blocks 0 and 1 are reserved for the directory. The rest of the disk, blocks 2 through 127, is available to store files.

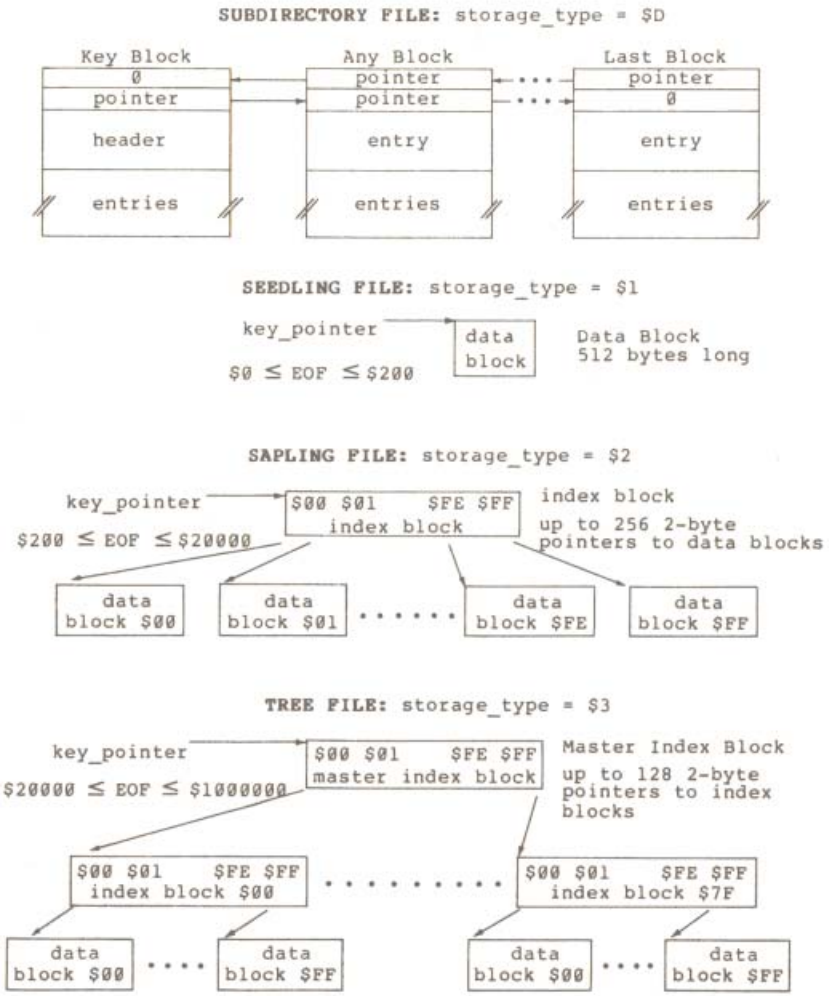


Figure 3.19 ProDOS/SOS File Storage Formats

TRACK NUMBER	SECTOR NUMBER 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	RESERVED															
1																
2																
3	DIRECTORY												BLOCK 2			BLOCK 3
4	BLOCK 4												BLOCK 6			BLOCK 7
5	BLOCK 8												BLOCK A			BLOCK B
6	BLOCK C												BLOCK E			BLOCK F
7	BLOCK 10												BLOCK 12			BLOCK 13
8	BLOCK 14												BLOCK 16			BLOCK 17
9	BLOCK 18												BLOCK 1A			BLOCK 1B
A	BLOCK 1C												BLOCK 1E			BLOCK 1F
B	BLOCK 20												BLOCK 22			BLOCK 23
C	BLOCK 24												BLOCK 26			BLOCK 27
D	BLOCK 28												BLOCK 2A			BLOCK 2B
E	BLOCK 2C												BLOCK 2E			BLOCK 2F
F	BLOCK 30												BLOCK 32			BLOCK 33
10	BLOCK 34												BLOCK 36			BLOCK 37
11	BLOCK 38												BLOCK 3A			BLOCK 3B
12	BLOCK 3C												BLOCK 3E			BLOCK 3F
13	BLOCK 40												BLOCK 42			BLOCK 43
14	BLOCK 44												BLOCK 46			BLOCK 47
15	BLOCK 48												BLOCK 4A			BLOCK 4B
16	BLOCK 4C												BLOCK 4E			BLOCK 4F
17	BLOCK 50												BLOCK 52			BLOCK 53
18	BLOCK 54												BLOCK 56			BLOCK 57
19	BLOCK 58												BLOCK 5A			BLOCK 5B
1A	BLOCK 5C												BLOCK 5E			BLOCK 5F
1B	BLOCK 60												BLOCK 62			BLOCK 63
1C	BLOCK 64												BLOCK 66			BLOCK 67
1D	BLOCK 68												BLOCK 6A			BLOCK 6B
1E	BLOCK 6C												BLOCK 6E			BLOCK 6F
1F	BLOCK 70												BLOCK 72			BLOCK 73
20	BLOCK 74												BLOCK 76			BLOCK 77
21	BLOCK 78												BLOCK 7A			BLOCK 7B
22	BLOCK 7C												BLOCK 7E			BLOCK 7F

Figure 3.20 CP/M Disk Layout

The CP/M directory is shown in Figure 3.21. There are 48 directory entries in the Microsoft implementation of CP/M for the Apple. At least one other implementation of CP/M for the Apple uses 64 directory entries. Byte $0 of each entry is a delete flag. The entry is empty if it equals $E5. Bytes $01 through $08 are the primary part of the file name. Bytes $09 through $0B are the file name's extension. Byte $0C determines which extent of the file is represented by this entry. A file may use as many extents (new file entries in the directory) as necessary for its length. Byte $F is the number of 128-byte records in this entry for the file. Bytes $10 through $1F represent up to 16 blocks that the file uses. If any are unused, they are zero.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	DEL	NAME								EXTENSION			XT			#R
10	..block numbers of blocks used by this extent..															

DEL = DELETE FLAG
XT = EXTENSION NUMBER
#R = NUMBER OF 128 BYTE RECORDS

Figure 3.21 CP/M Directory

CHAPTER 4

FILE TYPES

The purpose of this chapter is to describe the different types of files that are available in each of the four operating systems. The topics we will cover are:

1. How Universal File Conversion derives a name for new files based on the old filename.
2. A general explanation of how BASIC programs are stored.
3. Some words about converting binary files.
4. A description of those file types that are common to more than one operating system.
5. A description of those file types that appear in only one operating system.

The term **file types** as used in this chapter refers to the contents of the file, not to any special format the operating system uses to store the data (a Seedling file or a Sapling file, for example). We are concerned with the data as if it were a continuous stream of bytes.

NEW FILES

When assigning the name for a new file, Universal File Conversion uses the name of the original file as a starting point. The different operating systems have different maximum lengths for a filename. If the old filename is too long for the new operating system, it is truncated. Also, because only DOS 3.3 allows spaces as part of the filename, Universal File Conversion will remove spaces when converting to other operating systems.

Periods are not allowed in CP/M filenames, so Universal File conversion will remove them when creating a new CP/M file. Here are some examples:

```
Converting from CP/M to DOS 3.3--
              INVEN.DAT becomes INVEN.DAT

Converting from DOS 3.3 to CP/M--
               INVEN.DAT becomes INVENDAT
             INVEN DATA becomes INVENDAT.A
```

Universal File Conversion will not allow you to write over a file with the same name. If you wish to destroy an existing file and replace it with a new file of the same name, you must first use the operating system software to delete the old file, then use Universal File Conversion to write the new one. If you wish to preserve the existing file with a different name (e.g. FILE.BAK), you must first use the operating system to rename the old file, then use Universal File Conversion to write the new one.

CONVERTING BASIC PROGRAM FILES

Most BASIC interpreters store a program in a tokenized form. This means that each keyword, such as "PRINT" or "INPUT", is replaced by a byte or "token" which represents the keyword. Life would be simple if each BASIC replaced each keyword by the same "token" byte. Unfortunately, this is not the case.

This means that if you were to move the program in its tokenized form from one BASIC to another, it most likely would not make sense to the second BASIC. For example, the token for the PRINT statement in the first BASIC might be the token for the SIN function in the second.

Fortunately, there is a way to move BASIC programs. Most BASIC interpreters allow you to, optionally, store a program as a text file. It is in this form that Universal File Conversion will correctly move BASIC programs around. Note that DOS 3.3 and ProDOS use the same BASIC, namely Applesoft, so you do not have to convert to a text file before using Universal File Conversion when going back and forth from DOS 3.3 and ProDOS. This is the only exception.

After Universal File Conversion has moved the BASIC program in text form, you will probably still have to make changes. For example, Microsoft BASIC for CP/M has a "PRINT USING" statement, but Applesoft does not. You would have to change all the "PRINT USING" statements in the program so they make sense to Applesoft. A more drastic example is an Applesoft program that uses graphics. Such a program would require extensive changes to run in a BASIC that did not support graphics.

CONVERTING BINARY FILES

The intended use of the binary file is to store a machine code program. In practice, this is not always the case. For example, it may be used to store the high resolution graphics screen, or it could be text used by certain word processors. Universal File Conversion treats binary files as a machine code program or data--it never treats a binary file as text.

Note that moving a machine code program from CP/M to DOS 3.3 does not make much sense. The processor used by CP/M is a Z-80 and DOS 3.3 uses a 6502. They are not compatible. Generally, you cannot move a machine code program from one operating system to another and have it make any sense. Universal File Conversion permits you to move the file over, but you are responsible for any translation from one language to another.

COMMON FILE TYPES

There are 10 file types that are defined in more than one operating system on the Apple: text, binary, relocatable binary, Applesoft BASIC, Integer BASIC, Pascal code, Pascal text, Pascal data, foto, and bad files. The other file types are defined in only one operating system.

Note that CP/M does not have file types as such. The only file type in any way special to CP/M is the ".COM" file type. To CP/M, a ".COM" file is an executable 8080 (or Z-80) machine code program. Any other file to CP/M is simply some program's data file. Universal File Conversion treats CP/M files with the extensions ".TXT", ".ASC", ".FOR", ".MAC", ".ASM", and ".BAK" as containing text. All other files are treated as binary data.

Also note that all file types used by DOS 3.3 are defined in at least one other operating system. And, all Pascal file types but one are defined in another operating system.

Figure 4.1 indicates the type of destination file that will be created when a given file type is converted from one operating system to another. The destination file type depends on what operating system the file is being moved to. If the source file type is not shown in Figure 4.1, it is treated the same as a data file.

	DESTINATION			
	CP/M	DOS 3.3	PASCAL	ProDOS/SOS
TEXT	TEXT	TEXT	PAS TEXT	TEXT
PASCAL TEXT	TEXT	TEXT	PAS TEXT	TEXT
S APPLESOFT	DATA	APPLSFT	PAS DATA	APPLSFT
O INTEGER BASIC	DATA	INT BAS	PAS DATA	INT BAS
U BINARY	DATA	BINARY	PAS DATA	BINARY
R REL BINARY	DATA	REL BIN	PAS DATA	REL BIN
C FOTO	DATA	BINARY	FOTO	FOTO
E PASCAL CODE	DATA	BINARY	PAS CODE	PAS CODE
PASCAL DATA	DATA	BINARY	PAS DATA	PAS DATA
DATA / OTHER	DATA	BINARY	PAS DATA	TYPELESS
BAD	N/A	N/A	N/A	N/A

Figure 4.1 File Type Translation Table

Now we will look at each common file type in detail.

BAD FILES

The bad file is the easiest to understand. It is just a file created during formatting to prevent the use of damaged areas on the disk. As such, it is unreadable and contains no data.

TEXT FILES

The text file, which is used in some form by all of the operating systems, is also easy to understand. It consists of one or more records. Each record contains ASCII-type data and ends with a carriage return. A generalized example of a text file is shown in Figure 4.2.

```
54 48 49 53 20 49 53 20   THIS IS
41 4E 0D 45 58 41 4D 50   AN.EXAMP
4C 45 20 4F 46 0D 41 20   LE OF.A
54 45 58 54 20 46 49 4C   TEXT FIL
45 00 .....               E.
```

Figure 4.2 Text File Sample

Each operating system defines a text file slightly differently. For CP/M, the carriage return is always followed by a line feed. This is the only difference in CP/M. For DOS 3.3, there are two differences. The first is that the file is stored with the high bit of each byte "on." Second, the file may contain null bytes, also called zero bytes. A ProDOS or SOS text file may also contain null bytes. There are several differences for a Pascal text file. First, the first 1024 bytes of the file are reserved as data for the Pascal text editor. Second, the rest of the file is broken into 1024-byte pieces. No line of text may cross the 1K boundary. The unused space at the end of a 1K block is filled with null bytes. Third, the file must be a multiple of 1K in length. Fourth, each line may start with a space compression code. This is a DLE character ($10) followed by a character representing the number of spaces at the beginning of the line plus 32.

Universal File Conversion knows about the differences in format between different types of text files, and will do all the steps necessary to convert a text file from one format to another.

BINARY FILES

The binary file is a memory image with two other pieces of information--the address in memory where the image belongs, and the length of the image. A memory image is simply a copy of a specified portion of the computer's memory at the time the file was created. In ProDOS and SOS, the address and length information is kept in the directory entry for the file. The address is stored in the "auxiliary type" and the length is stored in the "EOF". In DOS 3.3, this information is stored in the file itself--the first two bytes of the file being the address and the second two being the length.

Universal File Conversion will add or remove, as necessary, the 4-byte header for a DOS 3.3 binary file. If it is adding the 4-byte header, the memory address it supplies is the previous address if there was one, otherwise it uses $2000. The length it supplies is the number of bytes in the file. If the destination operating system has a place for the 4-byte header, it is stored there. If there is no place for the four bytes, they are simply discarded.

APPLESOFT AND INTEGER BASIC FILES

The Applesoft and Integer BASIC file types are very similar to each other and to the binary file. These files consist of a memory image of the BASIC program, which is tokenized. The length of the file is also saved, in the same way as for the binary file. ProDOS and SOS store the length of the file in the directory. DOS 3.3 stores it in the first two bytes of the file.

Universal File Conversion will add or remove, as necessary, the two-byte length of an Applesoft or Integer BASIC file. When moving between ProDOS or SOS and DOS 3.3, the length information is transferred correctly.

RELOCATABLE BINARY FILES

The relocatable binary file is the output of the DOS Toolkit assembler or the ProDOS Toolkit assembler. Its format is described in the manuals for those programs.

Universal File Conversion makes no changes to this type of file.

FOTO FILES

The foto file is a memory image of the screen, usually the high resolution graphics. Because there are several different screen modes in both the Apple II and Apple III computers, we will not attempt to describe foto files in more detail.

Universal File Conversion makes no changes to this type of file unless it is being moved into DOS 3.3. Then the rules for a binary file apply.

PASCAL DATA FILES

The Pascal data file is made up of records. Each record is a memory image of whatever the user defined the file to represent. The Pascal operating system does not store any information telling how long a record is or what the record represents.

Universal File Conversion makes no change to a Pascal data file when moving between Apple Pascal and ProDOS (or SOS). When a Pascal data file is moved into DOS 3.3, the rules for a binary file apply.

PASCAL CODE FILES

The Pascal code file takes several forms. Regardless of which form the file is in, the first block (Ø) of the file is a segment dictionary. It contains information on up to 16 segments. Each segment begins on a block boundary. For a complete discussion of the internals of a Pascal code file, see Apple's "Pascal Technical Note #16".

Universal File Conversion makes no change to a Pascal code file when moving between Apple Pascal and ProDOS (or SOS). When a Pascal code file is moved into DOS 3.3, the rules for a binary file apply.

UNIQUE FILE TYPES

The remaining file types are defined in only one operating system. These files generally would not make sense if moved to a different operating system. Universal File Conversion makes no changes to any of these file types unless the rules for a binary file apply (only when the destination is DOS 3.3).

PASCAL-ONLY FILE TYPES

The Pascal **Graf** file is not used in the Apple. In U.C.S.D. Pascal, it was used for a special graphics terminal.

SOS-ONLY FILE TYPES

The **typeless** file ($ØØ) is just that. It has no type and its format is undefined.

The **font** file ($Ø7) is a special file containing a set of characters for the Apple III.

The **BASIC program** file ($Ø9) is an Apple III BASIC program. It is a tokenized memory image of the program.

The **BASIC data** file ($ØA) is a data file made by Apple III BASIC. Its format is up to the user.

The **word processor** file ($ØB) is not presently used.

The **SOS** file ($ØC) is a special Apple III system file, the formats of which are described in the "SOS Reference Manual".

The **RPS data** file ($1Ø) and the **RPS index** file ($11) are files from Apple III Record Processing Services. Their formats are described in the manual for RPS.

PRODOS-ONLY FILE TYPES

The ProDOS **added command** file ($FØ) is not used at present by ProDOS.

The ProDOS **System** file ($FF) is a special Apple II system file. This format is described in the "ProDOS Technical Reference Manual".

The ProDOS **AppleWorks** files ($19, $1A, and $1B) are designed to work with the AppleWorks package and their structures are specific to that package. Type $19 is an AppleWorks data base file, type $1A is an AppleWorks word processing file, and type $1B is an AppleWorks spreadsheet file.

The ProDOS **Pascal** file ($EF) is not used at present by ProDOS.

CHAPTER 5

GENERAL PROGRAM FLOW

This chapter explains in a somewhat general way how the Universal File Conversion program is laid out. You do not have to understand this chapter (or the rest of this manual, for that matter) to run Universal File Conversion. This chapter is included to give some insight to those who are curious to know how the program works and to help those who may want to modify the program for their own special use. The rest of this chapter assumes the reader has some knowledge about programming and a familiarity with Apple Pascal.

Universal File Conversion is organized into several modules. It makes extensive use of Apple Pascal's unit and segment facilities to accomplish this. Figure 5.1 shows the relationship of the different modules of the program. There is a unit for each of the four operating systems. Each of these units contains all the routines necessary for reading and writing files to disks that use that operating system. There is also a global unit. The global unit contains procedures for things that are needed for all operating systems, such as screen control and error messages.

There are segments for each Main Menu function except Copy and Quit. There is also a segment that initializes all the global variables of the program. This segment is executed only once. There are also some assembly language procedures that are linked into the segments and units where needed.

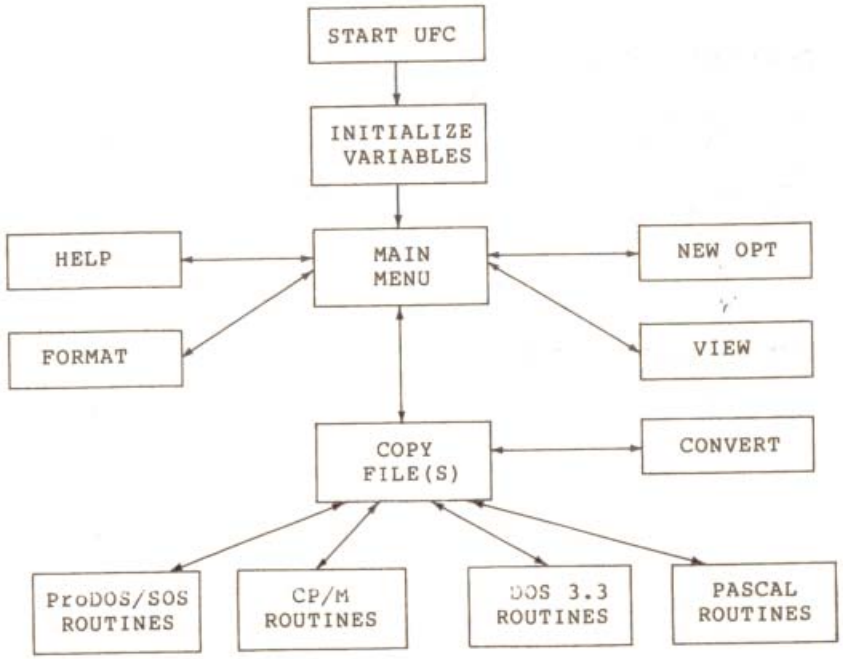


Figure 5.1 Overview of UFC Program Structure

The program also makes use of the Heap (Pascal's dynamic variables), to store data awaiting conversion from one operating system to another. The Heap is used because the different operating system units use differing amounts of memory. This allows as much data as possible to be loaded into memory at one time. The Heap is also used for storing directories and pointers into those directories for each operating system.

THE INITIALIZE SEGMENT

When the Universal File Conversion program begins to execute, the first thing it does is call the Initialize segment. This procedure does the following four things: it sets all of the pointer variables to NIL, loads the screen control characters from the *SYSTEM.MISCINFO file, sets the source and destination indexes, and sets the default system types and drives.

THE MAIN MENU

The program next displays the Main Menu. This procedure directly coordinates all functions of Universal File Conversion, except for the copying of files. When the Menu procedure begins, it loads into memory all the segment procedures for all the commands except copying files. When the user wants to copy files, the Menu procedure is exited and all of the segments it loaded into memory are released.

THE FORMAT PROCEDURE

The Format procedure begins by loading several data files and the actual disk formatter into memory. The data files contain empty directories and a bootup program. The bootup program will be written to the formatted disk. It will be executed when the formatted disk is booted and will simply turn the drive off and display a message saying that there is no DOS on the disk.

The Format procedure asks the user which disk, source or destination, he wants to format. The program then checks to see if it can read the disk in the specified drive. If it can, it asks the user if he wants to erase the disk. If yes, the program then begins to format the disk. The formatter does a quick speed check on the specified drive before it formats the disk.

After the formatter finishes, there is nothing on the diskette. The bootup program is then written to track 0, sector 0. Then a case statement is used to determine which directory must be written on the disk. After this is finished, the program asks the user if he wants to format another disk by asking him which disk, if any, he wants to format.

THE VIEW PROCEDURE

The View procedure is used when the user wants to look at a directory. The user specifies the drive for which he wants to view the directory. A case statement is used to call the appropriate procedure for the operating system associated with the specified disk drive. Each operating system procedure displays the file names in the directory (or catalog), and then waits for the user to press a key. When the user presses the key, he is asked if he wants to view another directory or return to the Main Menu.

THE HELP PROCEDURE

The Help procedure displays a help screen menu, from which the user can select the particular help screen he wants. The program then displays the selected help screen. Pressing a key from a help screen returns to the help screen menu.

THE NEWOPT PROCEDURE

The NEWOPT procedure is used when the user wishes to change operating system types, or disk drives. It uses nested case statements to get the new disk drive assignments and operating system types from the user.

COPYING FILES

When it is time to copy files, a nested case statement is used to determine which operating system units and variables need to be loaded into memory. It calls different procedures which load the correct unit(s) and variables. All of these procedures in turn call the procedure that does the actual copying. In this way, only those units and variables that are actually needed are assigned memory space.

When the copying procedure begins, it asks the user for the file name(s) he wants to copy. Here the copy procedure begins a loop, looking for any file matching the given name on the source disk. If it doesn't find one, it exits, ending up back at the Main Menu. If a file is found, it next looks at the destination disk to see if a file of the same name exists. If so, it asks the user for a new destination name (existing files cannot be copied over). When it has a name for the file that is not already on the destination disk, it copies the file. When it has copied the file, it goes back to see if another file on the source disk is to be copied.

OPERATING SYSTEM MODULES

The above description of copying files is a greatly simplified look at the program. It does not include the many places the program checks to see that the right disk is in the right drive before accessing it, or how each of the operating system modules work. We will take a look at some of the support modules for the Main Program below.

Each operating system module has five procedures that control the reading or writing of disks. They are: Search Directory, Open for input, Open for output, Read a sector, and Write a sector. The four operating system modules are: PASCALSTUFF, SOSSTUFF, CPMSTUFF, and DOSSTUFF.

Any procedure, such as Read a sector, in any operating system is called with the same parameters. This means that the procedure must take care of all differences of its respective operating system. By doing this, the Main Program does not have to know anything about what operating system it is using. This also makes it easy to add another operating system to the program.

To read a file using a module, you must first call the Search Directory routine. This sets up some pointers for the Open for input routine, which is called next. Finally, you can call the Read a sector routine to read the file. The process is similar for writing to a file.

THE CONVERT PROCEDURE

The operating system modules take care of the physical translation of a file from one operating system to another operating system. The Convert assembly language procedure takes care of the logical translation. It handles shifting the file left or right by two or four bytes if necessary, such as to put an address and length on a binary file. If the file is a text file, it will add or remove line feeds, add or remove Pascal space compression codes, and add or remove the Pascal 1K blocking of text files. Of course, it will not make unnecessary changes.

Another of Convert's duties is to handle any leftover part of a file, if the file does not fit into memory all at once. Convert does this using two 1-sector buffers that are hidden from the Main Program.

VERIFYING DISKETTE TYPE

There are many times that Universal File Conversion must check to see that the correct disk is in the drive. Whenever the Universal File Conversion disk itself is needed, it first checks to see that it is in the drive. The only times the program needs its own disk, after initial boot, are at the beginning and end of the copy process or when a diskette is to be formatted.

As for data disks, two different sets of rules apply depending on whether a single drive copy or a double drive copy is requested. If a 2-drive copy is requested, the only time the program checks for format compatibility is the first time it accesses the disk. It assumes the disks stay put until the copying is complete. Note that the program never asks the user to place a disk in a drive when doing so would remove a data disk that is needed later. In single drive mode, however, the program checks to see if the correct disk is installed after every disk swap.

How does the program determine if the correct diskette is in the drive? To find the Universal File Conversion disk, it looks at the directory to see that the disk is called "UFC:". For a data disk, the program only checks to see that the disk has the proper type of directory for the operating system it expects. This does not necessarily prevent the wrong disk from being used, but it greatly reduces the chance of it happening.

BIBLIOGRAPHY

"Apple 6502 Assembler/Editor." Cupertino: Apple Computer, Inc., 1980.

"Apple Business BASIC Reference Manual," Vol. 1. Cupertino: Apple Computer, Inc., 1981.

"Apple Pascal Object Module Format." Apple Computer, Inc., Pascal Technical Note #16. Cupertino: 1981.

"Apple Pascal Operating System Reference Manual." Cupertino: Apple Computer, Inc., 1980.

"Apple Pascal System Disk Directory Structure." Apple Computer, Inc., Pascal Technical Note #4. Cupertino: 1981.

De Groat, Ron, ed., "Call A.P.P.L.E. In Depth: All About Pascal." Seattle: Apple Pugetsound Program Library Exchange, 1982.

Reed, Don. "SOS Reference Manual," Vol. 1. Cupertino: Apple Computer, Inc., 1982.

"The DOS Manual." Cupertino: Apple Computer, Inc., 1980.

"The ProDOS Technical Reference Manual." Cupertino: Apple Computer, Inc., 1983.

Worth, Don, and Pieter Lechner. "Beneath Apple DOS." Chatsworth: Quality Software, 1981.

Worth, Don, and Pieter Lechner. "Beneath Apple ProDOS." Chatsworth: Quality Software, 1984.

ORDERING THE SOURCE CODE

For those users who desire to examine the source code to Universal File Conversion, or attempt to modify the program for special applications, an annotated listing of the source code is available directly from Quality Software. ~~Two separate packages are available:~~

~~1. The printed annotated listings, and~~
2. A diskette containing the source files.

Most of Universal File Conversion was written in Pascal, and therefore the source is primarily in Pascal. Substantial portions of the program are written in assembly language, however. To properly use the source files on diskette requires the Apple Pascal system version 1.2 (including the Apple Pascal assembler).

To order ~~the listings and/or~~ the source files on diskette, you must mail the coupon on the following page ~~(one for each product ordered)~~ directly to Quality Software along with a payment of $10.00 ~~for each product ordered~~ **plus** shipping and handling charges*. Your payment can be a check or bank draft in US dollars, or your VISA or MASTERCARD number and expiration date. California residents must add the appropriate sales tax (6 or 6.5%). No phone orders or CODs will be accepted.

Send your order to:

Quality Software
21601 Marilla Street
Chatsworth, CA 91311

* Shipping and handling charges are (per order):
United States, Canada, Mexico.....$ 2.50
All other countries (insured air mail)...$10.00

COUPON FOR ORDERING SOURCE CODE LISTING TO UNIVERSAL FILE CONVERSION

Please cut this coupon out of the manual and mail it (not a copy) to Quality Software.

Please send me:

The printed annotated source listing $10.00
(CA Residents) Sales Tax _____
Shipping and handling* _____

TOTAL _____

Check #_____
OR VISA/MASTERCARD #_____Exp_____

Name _____

Street Address _____

City, State, Postal Code _____

Country _____

==========
==========

COUPON FOR ORDERING SOURCE CODE DISKETTE TO UNIVERSAL FILE CONVERSION

Please cut this page out of the manual and mail it (not a copy) to Quality Software.

Please send me:

A diskette containing the source code $10.00
(CA Residents) Sales Tax _____
Shipping and handling* _____

TOTAL _____

Check #_____
OR VISA/MASTERCARD #_____Exp_____

Name _____

Street Address _____

City, State, Postal Code _____

Country _____