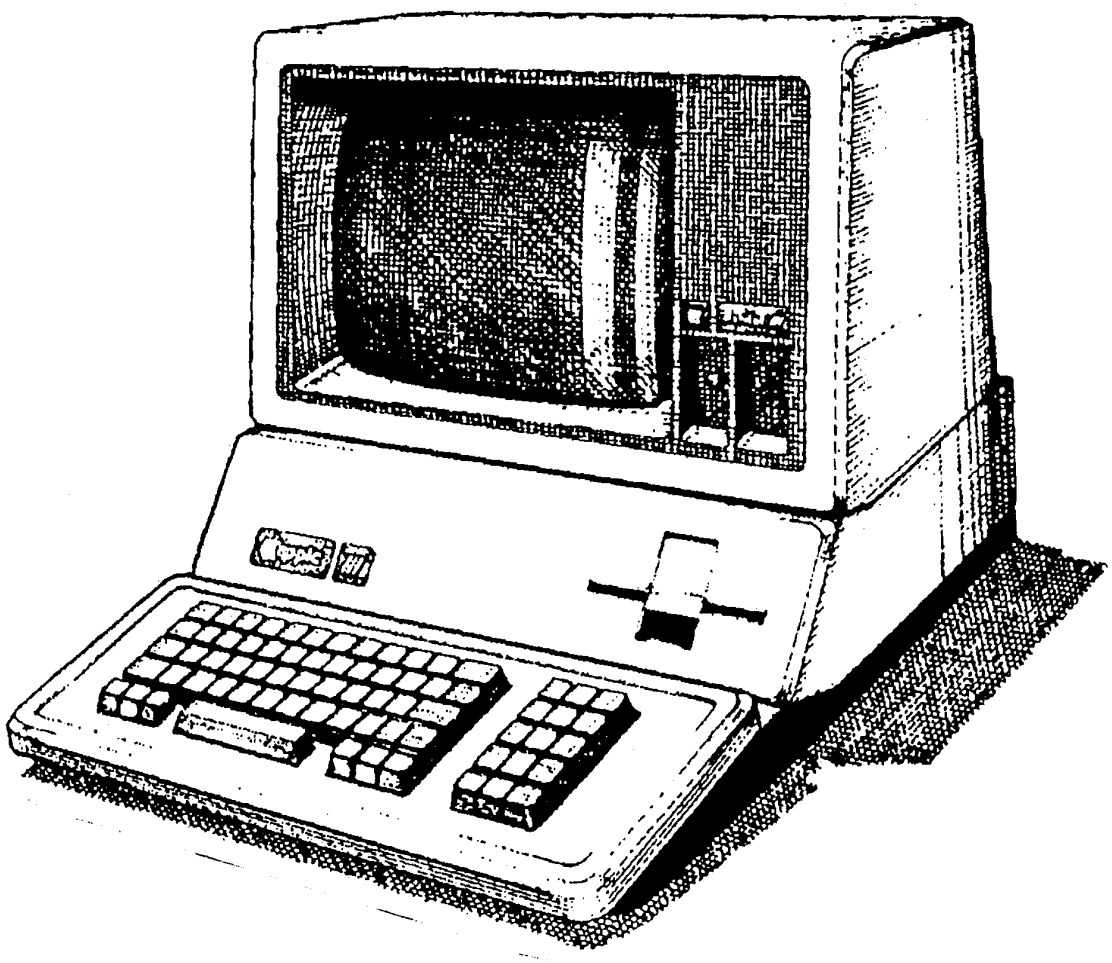
Apple /// Computer Information



DOCUMENT NAME

APPLE /// 5.25" FLOPPY DISK IMAGE MAKER PROGRAM SPECIFICATION (FEB. 1999)

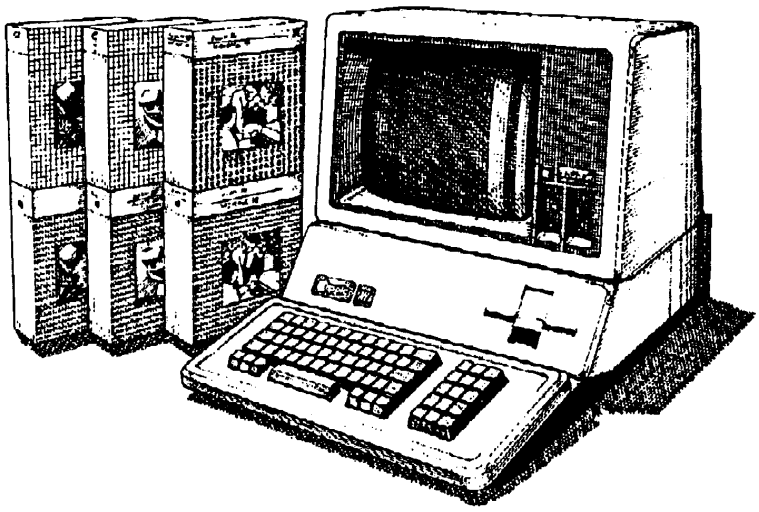
#

221

Ex Libris David T. Craig

"_001.PICT" 255 KB 2001-08-25 dpi: 300h x 300v pix: 2232h x 2954v

Apple /// Computer Information



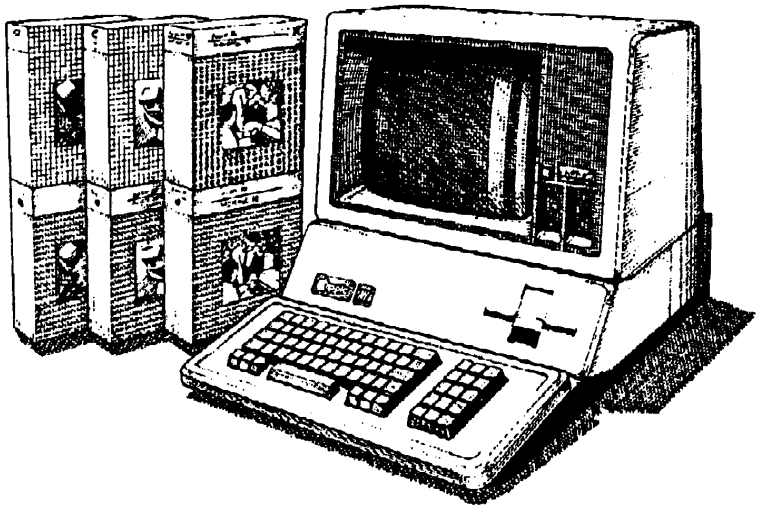
David Craig's
Apple /// 5.25" Floppy Disk Image Maker
Program Specification

Document Table of Contents

Revision 0	24 February 1999
Revision 1	?
Revision 2	08 March 1999
Revision 3	10 March 1999
Revision 4	11 March 1999
Revision 5	18 March 1999 (matrix)
Revision 5	18 March 1999 (laser)

"_002.PICT" 188 KB 2001-08-25 dpi: 300h x 300v pix: 2544h x 3338v

Apple /// Computer Information



David Craig's
Apple /// 5.25" Floppy Disk Image Maker
Program Specification

Revision 0
24 February 1999

"_003.PICT" 164 KB 2001-08-25 dpi: 300h x 300v pix: 2544h x 3338v

APPLE /// INFO

APPLE ///
5.25" FLOPPY DISK IMAGE MAKER PROGRAM SPECIFICATION

Revision 0 -- David T. Craig -- 24 February 1999

ADD: TABLE OF CONTENTS, NUMBERED SECTIONS

REVISIONS

Date	Rev	Comments
1999-02-24	0	Preliminary spec.

USE SINGLE DASH

AUTHOR

David T. Craig
941 Calle Mejia # 1006
Santa Fe, NM 87501 USA

home phone: (505) 820-0358
email: 71533.606@compuserve.com

o AUDIENCE

Apple /// computer users who want to preserve their collections of Apple /// 5.25" floppy disks.

ADD SAVE TO CP/M

o INTRODUCTION

This specification describes a proposed Apple /// program that would let users maker disk image files from Apple /// 5.25" 140K floppy disks and also re-create these disks from a disk image file.

This specification should at this time be considered preliminary.

Comments about this specification are welcome.

o SCHEDULE

This program should hopefully be finished in about 2 weeks. The reason for this deadline is I have borrowed some Apple /// disks that I need to

return in around 2 weeks and I want to make disk image files from these disks before returning them.

o GOALS

The goals of this program are:

+ Provide a simple to use Apple /// program that allows users to save 5.25" Apple /// floppy disks to external media.

+ Send disk data out the Apple /// computer's serial port to another computer which will save the disk image files as regular files on its disk media. This other computer would only need a terminal program that can receive files via its serial ports.

+ Receive disk image files to an Apple /// computer via its serial port and recreate the original disk onto an Apple /// formatted or unformatted floppy disk.

+ Image file sending and receiving will be on a floppy disk block basis. A disk block contains 512 bytes.

+ Support Apple // 5.25" formatted disks also. See section APPLE // DISK SUPPORT at the end for more details.

+ Copy protected disks will not be supported. If a disk's blocks cannot all be read then no disk image will be created by this program.

o IMAGE FILE FORMAT

Disk image files will have the following characteristics:

Contain only human-readable characters (i.e. no control or Extended ASCII characters, except for the carriage return (ASCII 13) and line feed (ASCII 10) characters). Reason: So these files can easily be transferred between machines and over email networks with worrying about any special characters causing problems.

Stored as text files with whatever machine operating system extension information is needed. For example, on PCs they would normally end in ".TXT", on Macs the internal document type should be "TEXT".

Formatted on a line basis with each line terminated with carriage return and line feed characters. Reason: So Microsoft Windows-based machines and UNIX-based machines can view the files with their word processors. Macintosh machines should also be able to view these files though there will appear a space before all but the first line of the image file.

Blank lines may exist and are ignored.

Divided into disk block groups with each group headed by a comment specifying the block number starting with block 0.

Disk blocks are divided into 32 byte chunks for a total of 16 chunks. Each 32 byte chunk appears as 64 characters with a leading address value and with a following ASCII representation of the bytes. Each 32 byte chunk is also terminated with a checksum value that is computed from the 32 bytes. WHOLE LINE (NOT JUST 32 DATA BYTES)

File starts with a header and ends with a footer comment. All comments are stored on a line basis with each comment line starting with the ";" character marker (comment marker must start at position 1 of the line). Most header comments are not required but the footer comment is required. The first line of the file must be "APPLE 5.25 INCH DISK IMAGE FILE". The header comment specifies the following information:

Disk contents	e.g. Apple Writer /// 1.1
Disk format	e.g. Apple /// SOS
Disk block count	e.g. 280
Image creation date	e.g. 24 February 1999
Image program creator name	e.g. Apple3DiskImageMaker 1.0.0
Person who created image	e.g. David T. Craig
Address of person	e.g. 123 Main, City, State Zip USA
Person Contact info	e.g. 71533.606@compuserve.com

A sample header is:

```
; APPLE 5.25 INCH DISK IMAGE FILE
;
; DISK_NAME: Apple Writer /// 1.1
; DISK_FORMAT: Apple /// SOS
; DISK_BLOCKS: 280
; DATE: 24 February 1999
; CREATED_BY: Apple3DiskImageMaker 1.0.0
; CONTACT_NAME: David T. Craig
; CONTACT_ADDRESS: 941 Calle Mejia #1006, Santa Fe, NM 87501 USA
; CONTACT_EMAIL: 71533.606@compuserve.com
```

The footer comment contains the following - these two lines must be the last lines in the file:

```
; DISK_CHECKSUM: nnnnnnnn
; FINIS
```

Note: The final checksum may not be used in the first version of this program if its calculation causes problems.

Missing header or footer information will cause the image file to be seen as corrupted and will not be processed. Incorrect checksums indicate disk image is corrupt.

Image files will be verified on a line basis. Verification will test that only human readable characters exist (line feed at start of a line is OK), line address is correct, data section is correct (has correct length, no spaces, correct hex characters), ASCII characters are correct (length, characters), and the line checksum is correct for the data section. BLOCK HAS ALSO VERIFIED

Each disk block may MUST begin with the following comment (the $ number is the hexadecimal number for the block followed by a "/" then the block number in decimal):

```
; BLOCK: $0081/129
```

Each block line contains the following sections:

```
block(hex) address(hex): data(hex) [ascii] CHECKSUM: checksum(dec)

block(hex)      - block number in hex
addreses(hex)   - 4 hex digits teminated with ":"
data(hex)       - 64 hex digits
[ascii]         - 32 characters bound by "[" and "]"
                - control codes, spaces, DEL and extended are "."
checksum(dec)   - decimal checksum of whole line
```

A sample block with a few lines is:

GIVE DETAILS ABOUT EXACTLY WHAT IS CHECKSUMMED + CS ALGOR.

```
; BLOCK: $0000/0
;
0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHECKSUM: 12345
0000 0020: ...
0000 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHECKSUM: 12345
```

Sample skeleton image file is:

```
; APPLE 5.25 INCH DISK IMAGE FILE
;
; DISK_NAME: Apple Writer /// 1.1
; DISK_FORMAT: Apple /// SOS
; DISK_BLOCKS: 280
; DATE: 24 February 1999
; CREATED_BY: Apple3DiskImageMaker 1.0.0
; CONTACT_NAME: David T. Craig
; CONTACT_ADDRESS: 941 Calle Mejia #1006, Santa Fe, NM 87501 USA
; CONTACT_EMAIL: 71533.606@compuserve.com

; BLOCK: $0000/0
;
0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHECKSUM: 12345
0000 0020: ...
0000 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHECKSUM: 12345

...
```

```
      ; BLOCK: $0117/279
      ;
      0117 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHECKSUM: 12345
      0117 0020: ...
      0117 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHECKSUM: 12345

      ; DISK_CHECKSUM: 458923461
      ; FINIS
```

o **USER INTERFACE**

The disk image program will be named Apple3DiskImageMaker. When the user runs this program it should ask the user what to do, either send a disk image our a serial port or receive a disk image from an external machine. The user interface should be a simple command-line with the following contents:

```
Command:  S)end-disk  R)eceive-image  H)elp  Q)uit ?
```

The program should provide as many defaults as possible so users need to type too much. For example, when the program as for a disk drive the program should display a default drive specifier and if the user presses the Return key this default is used. Also, once an item is specified that item should retain that value when the value is prompted for again. For example, if the user enters .D2 as the drive specifier, then this item should appear as the default for the next prompt that needs a drive specifier. Processing should show some type of indicator, for example a line of "*" characters when a disk's blocks are being sent.

Character case of inputs should be ignored (e.g. "x" = "X") and extra leading or trailing spaces should also be ignored (e.g. " david " = "david").

If S is pressed:

```
Disk Drive to Send [.D1] ? .D2
Disk Name [Apple Writer /// 1.1] ?
Disk Format [Apple /// SOS] ?
Disk Blocks [280] ?
Date [24 February 1999] ?
Contact Name [David T. Craig] ?
Contact Address [941 Calle Mejia #1006, Santa Fe, NM 87501 USA] ?
Contact Email [71533.606@compuserve.com] ?
Sending 280 disk blocks to the serial port ...
Blocks: ********************
```

If R is pressed:

```
Disk Drive to Receive [.D2] .D3
Disk in drive .D3 is not formatted, formatting disk ...
Receiving disk image from serial port ...
```

```
     Blocks: ********************

     If the disk in the drive to receive is already formatted then the
following prompt should appear:

     WARNING -- Disk in drive .D3 is already formatted.
     Overwrite this disk [N] (Y/N) ?

If H is pressed:

     This program's purpose is to yadda yadda yadda ...

If Q is pressed:

     Really Quit [Y] (Y/N) ?

The program should when it starts display information about itself such
as the program name, version number, creation date, and author info.  For
example,

     Apple /// Disk Image File Sender and Receiver Utility
     Version 1.0.0 / 24 Feburary 1999
     Written by David T. Craig

     Command:  S)end-disk  R)eceive-image  H)elp  Q)uit ?

     ======================================================================
o    APPLE // DISK SUPPORT
     ======================================================================

This program should also be implemented so that it runs on the Apple //
family of machines (e.g. Apple ][, Apple ][+, Apple //e, Apple //c, Apple
//gs).  The reason for this is teh low-level disk format of the Apple ///
5.25" disks and the Apple // fmaily 5.25" disks are the same (the
directory structures may differ but that is immaterial at the ~~low-level~~
block level).                                                  DISK

This Apple // family version program should also fully support the
sending and receiving of Apple /// disk image files.

Disk drive specifiers may be different on the // version of this program
since the // family does not use .D1, .D2, etc for drives. I will need to
check how disks are specified in the // world.  Or, the /// drive
specifiers could be used and the // program would translate them to what
it really needs.

This program should use the Apple // family serial I/O devices such as
the Super Serial Card in the Apple //e or the built-in serial port of the
Apple //c (//gs?) machine.  PGM DOES NOT CHANGE SERIAL I/O PARMS - USERS DO
                                                          EXTERNALLY
This program will most likely not support automatic disk formatting for
when a disk image is received from an external machine since I believe
the Apple // family OS does not support a formatting driver such as the
```

/// supports. I know you can get the formatter for Apple // Pascal use but I don't have it and don't want to bother at this time with this since the user can just pre-format a floppy for disk creation use.

o PROGRAMMING DETAILS

The program should be programmed in a high-level language and not assembly language (why? I'm too old to deal with assembly any more). I plan on using Apple /// Pascal for the /// implementation. For the // implmentation I plan to use Apple // Pascal 1.3 or 1.2.

I will most likely create the /// version first and then use the ///'s sources for the // implementation. The source should have conditional compilation statements governing the destination platform so that only a single instance of the source code need to exist. For example, the first line of the source could be:

```
{$SETC APPLE3 := TRUE}  { /// version use TRUE, // use FALSE }
```

Any areas of the program that are // or /// specific (e.g. disk formatting) would be bounded by the APPLE3 condition and do what is needed for that area.

o UTILITY PROGRAMS ON OTHER MACHINES

It may be useful to have some utility programs running on other machines (e.g. Macintosh) that deal with disk image files.

For example, there should exist a program that verifies text disk images files. Also, there may need to be a program that can convert a text disk image file to a binary disk image file for use by other programs that can deal with binary disk images (e.g. the various Apple // emulator programs seem to use binary images only). Likewise, there should be a utility that converts binary disk images to text files in case you want to send such a text image to my program to re-create a real 5.25" floppy disk.

These programs could all be the same program. I most likely will create such a utility for my Macintosh.

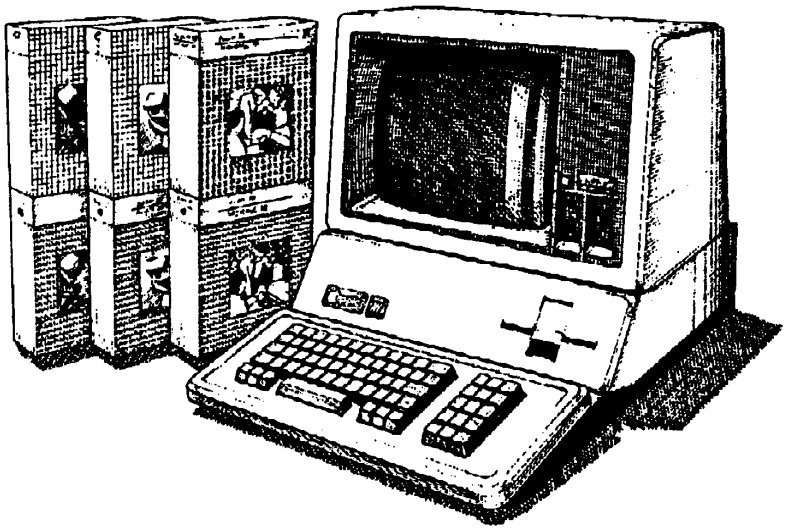
o REFERENCES

(TBD)

###

"_010.PICT" 202 KB 2001-08-25 dpi: 300h x 300v pix: 2472h x 3213v

Apple /// Computer Information

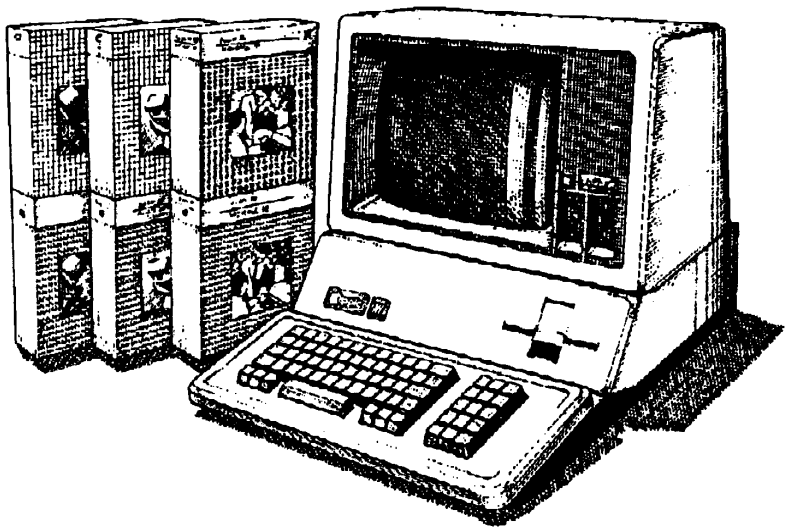


David Craig's
Apple /// 5.25" Floppy Disk Image Maker
Program Specification

Revision 1
? not present ?

"_011.PICT" 165 KB 2001-08-25 dpi: 300h x 300v pix: 2472h x 3213v

Apple /// Computer Information



David Craig's
Apple /// 5.25" Floppy Disk Image Maker
Program Specification

Revision 2
08 March 1999

"_012.PICT" 161 KB 2001-08-25 dpi: 300h x 300v pix: 2472h x 3213v

APPLE ///
5.25" FLOPPY DISK IMAGE MAKER PROGRAM SPECIFICATION

Revision 2 -- David T. Craig -- 08 March 1999

TABLE OF CONTENTS

0.0 REVISIONS

Date	Rev	Comments
1999-02-24	0	Preliminary spec
1999-02-25	1	Added table of contents, spelling corrections
1999-03-08	2	Added line checksum detail

APPLE /// INFO

1.0 AUTHOR

David T. Craig
941 Calle Mejia # 1006
Santa Fe, NM 87501 USA

home phone: (505) 820-0358
email: 71533.606@compuserve.com

2.0 AUDIENCE

Apple /// computer users who want to preserve their collections of Apple /// 5.25" floppy disks.

APPLE /// 5.25" FLOPPY DISK IMAGE MAKER PROGRAM SPECIFICATION
Revision 2 -- David T. Craig -- 08 March 1999 -- 2 / 16

"_014.PICT" 67 KB 2001-08-25 dpi: 300h x 300v pix: 2472h x 3213v

3.0 INTRODUCTION

This specification describes a proposed Apple /// program that would let users make disk image files from Apple /// 5.25" 140K floppy disks and also re-create these disks from a disk image file. Apple // computer 140K disks can also be made into disk images (see section APPLE // DISK SUPPORT).

This specification should at this time be considered preliminary.

Comments about this specification are welcome.

4.0 SCHEDULE

This program should hopefully be finished in about 2 weeks. The reason for this deadline is I have borrowed some Apple /// disks that I need to return in around 2 weeks and I want to make disk image files from these disks before returning them.

I plan to implement this program first on the Apple // since my // is setup and my /// is not and I'm too lazy to get it out of its box and set everything up.

5.0 GOALS

The goals of this program are:

- Provide a simple to use Apple /// program that allows users to save 5.25" Apple /// floppy disks to external media in a reliable manner.

- Send disk data out the Apple /// computer's serial port to another computer which will save the disk image files as regular files on its disk media. This other computer would only need a terminal program that can receive files via its serial ports.

- Receive disk image files to an Apple /// computer via its serial port and recreate the original disk onto an Apple /// formatted or unformatted floppy disk.

- Image file sending and receiving will be on a floppy disk block basis. A disk block contains 512 bytes.

- Support Apple // 5.25" formatted disks also. See section APPLE // DISK SUPPORT at the end for more details.

- Copy protected disks will not be supported. If a disk's blocks cannot all be read then no disk image will be created by this program.

6.0 IMAGE FILE FORMAT

Disk image files will have the following characteristics:

- Contain only human-readable characters (i.e. no control or Extended ASCII characters, except for the carriage return (ASCII 13) and line feed (ASCII 10) characters). Reason: So these files can easily be transferred between machines and over email networks without worrying about any special characters causing problems.

- Stored as text files with whatever machine operating system extension information is needed. For example, on PCs they would normally end in ".TXT", on Macs the internal document type should be "TEXT".

- Formatted on a line basis with each line terminated with carriage return and line feed characters. Reason: So Microsoft Windows-based machines and UNIX-based machines can view the files with their word processors. Macintosh machines should also be able to view these files though there will appear a space before all but the first line of the image file.

- Blank lines may exist and are ignored.

- Divided into disk block groups with each group headed by a comment specifying the block number starting with block 0.

- Disk blocks are divided into 32 byte chunks for a total of 16 chunks. Each 32 byte chunk appears as 64 characters with a leading address value and with a following ASCII representation of the bytes. Each 32 byte chunk is also terminated with a checksum value that is computed from the 32 bytes.

- File starts with a header and ends with a footer comment. All comments are stored on a line basis with each comment line starting with the ";" character marker (comment marker must start at position 1 of the line). Most header comments are not required but the footer comment is required. The first line of the file must be "APPLE 5.25 INCH DISK IMAGE FILE". The header comment specifies the following information:

Disk contents	e.g. Apple Writer /// 1.1
Disk format	e.g. Apple /// SOS
Disk block count	e.g. 280
Image creation date	e.g. 24 February 1999
Image program creator name	e.g. Apple3DiskImageMaker 1.0.0
Person who created image	e.g. David T. Craig
Address of person	e.g. 123 Main, City, State Zip USA

APPLE /// INFO

Person Contact info e.g. 71533.606@compuserve.com
Misc. comment e.g. Really neat program

A sample header is:

```
; APPLE 5.25 INCH DISK IMAGE FILE
;
; DISK_NAME: Apple Writer /// 1.1
; DISK_FORMAT: Apple /// SOS
; DISK_BLOCKS: 280
; DATE: 24 February 1999
; CREATED_BY: Apple3DiskImageMaker 1.0.0
; CONTACT_NAME: David T. Craig
; CONTACT_ADDRESS: 941 Calle Mejia #1006, Santa Fe, NM 87501 USA
; CONTACT_EMAIL: 71533.606@compuserve.com
; COMMENT: Really neat program
```

The footer comment contains the following:

```
; DISK_CHECKSUM: nnnnnnnn
; FINIS
```

The "FINIS" line MUST be the last line in the image file otherwise the disk restoration will see the file as corrupt.

- Missing header or footer information will cause the image file to be seen as corrupted and will not be processed. Incorrect checksums indicate disk images are corrupt.

- Image files will be verified on a line basis. Verification will test that only human readable characters exist (line feed at start of a line is OK), line address is correct, data section is correct (has correct length, no spaces, correct hex characters), ASCII characters are correct (length, characters), and the line checksum is correct for the whole line.

- Each disk block begins with the following comment (the $ number is the hexadecimal number for the block followed by a "/" then the block number in decimal):

```
; BLOCK: $0081/129
```

Each block line contains the following sections:

```
block(hex) address(hex): data(hex) [ascii] CHKSUM: checksum(dec)
```

block(hex) - block number in hex

APPLE /// 5.25" FLOPPY DISK IMAGE MAKER PROGRAM SPECIFICATION
Revision 2 -- David T. Craig -- 08 March 1999 -- 6 / 16

"_018.PICT" 159 KB 2001-08-25 dpi: 300h x 300v pix: 2472h x 3213v

```
addreses(hex)     - 4 hex digits teminated with ":"
data(hex)         - 64 hex digits
[ascii]           - 32 characters bound by "[" and "]"
                  - control codes, spaces, DEL and extended are "."
checksum(dec)     - decimal checksum of whole line

A sample block with a few lines is:

; BLOCK: $0000/0
;
0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 12345
0000 0020: ...
0000 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 12345
```

o The checksum is a decimal value that is calculated based on ALL of the characters in the line. The calculation is a repeated sum of all of the line character ASCII values multipled against a specific digit "Key". The "key" here are the digits of the number Pi. For example, if the line contains the following:

"DAVID"

then the checksum will be calculated as follows:

PI to 99 digits =
3141592653 5897932384 6264338327 9502884197 1693993751
0582097494 4592307816 4062862089 9862803482 5342117067

ASCII "D" = 68	ASCII "3" = 51
ASCII "A" = 65	ASCII "1" = 49
ASCII "V" = 86	ASCII "4" = 52
ASCII "I" = 73	ASCII "1" = 49
ASCII "D" = 68	ASCII "5" = 53

Checksum = (68 x 51)+(65 x 49)+(86 x 52)+(73 x 49)+(68 x 53)
= 3,468 + 3,185 + 4,472 + 3,577 + 3,604
= 18,306 (appears as 18306 in the file, no commas)

Illustrative programming code in the 4th Dimension language by ACI of France for this checksum follows:

```
$pi100 :=   "31415926535897932384626433832795028884"   +
            "19716939937510582097494459923"  +
            "07816406286208998628034825342117067 9"

Repeat
      $data:=Request("Enter  data to checksum:";"")

      If ($data # "")
            $checksum:=0

            For ($i;1;Length($data))
                  $data_ascii  := Ascii( Substring($data;$i;1)  )
                  $pi_ascii  := Ascii( Substring($pi100;$i;1)  )

                  $checksum  := $checksum + ($data_ascii*$pi_ascii)
            End for

            ALERT("Checksum  = " + String($checksum))
      End if
Until ($data="")
```

The checksum is calculated from all of the data in a single line. For example, the above sample disk block line would have a checksum calculated on the following data which includes the spaces and other non-disk block data:

"0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67]"

(Side Note) Why is this algorithm used? I have used the more traditional checksum algorithms such as exclusive-ORs and CRCs, but wanted something different for this program. This algorithm should prove sufficient and besides that, I have an interest in the number Pi and wanted to use it in some real world situation.

o The footer checksum is calculated based on the checksums of the individual lines. This checksum is just the sum of the ASCII values of the digits in the line checksums. For example, if you have a file with 2 lines with the line checksums "12" and "3" then the footer checksum would be:

```
Footer Checksum = ASCII("1") + ASCII("2") + ASCII("3")
                = 49 + 50 + 51
                  = 150
```

APPLE /// 5.25" FLOPPY DISK IMAGE MAKER PROGRAM SPECIFICATION
Revision 2 -- David T. Craig -- 08 March 1999 -- 8 / 16

o Sample skeleton image file is:

```
; APPLE 5.25 INCH DISK IMAGE FILE
;
; DISK_NAME: Apple Writer /// 1.1
; DISK_FORMAT: Apple /// SOS
; DISK_BLOCKS: 280
; DATE: 24 February 1999
; CREATED_BY: Apple3DiskImageMaker 1.0.0
; CONTACT_NAME: David T. Craig
; CONTACT_ADDRESS: 941 Calle Mejia #1006, Santa Fe, NM 87501 USA
; CONTACT_EMAIL: 71533.606@compuserve.com
; COMMENT: Really neat program

; BLOCK: $0000/0
;
0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 344785
0000 0020: ...
0000 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 56873

...

; BLOCK: $0117/279
;
0117 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 437823
0117 0020: ...
0117 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 5623445

; DISK_CHECKSUM: 57856
; FINIS
```

7.0 USER INTERFACE

The disk image program will be named Apple3DiskImageMaker. When the user runs this program it asks the user what to do, mainly either send a disk image our a serial port or receive a disk image from an external machine. The user interface should be a simple command-line with the following contents:

```
S)end-disk R)eceive-image F)ormat V)erify-disk H)elp Q)uit ?
```

Disk drives are specified by numbers. For example, the number 1 means the first disk drive, number 2 the second disk drive, etc. For Apple /// users, drive 1 means .D1, drive 2 means .D2, etc. For Apple // users, drive 1 means unit 4, drive 2 unit 5, drive 3 unit 9, drive 4 unit 10. This is done so that this program has consistent disk drive references for both the Apple /// and the Apple // platforms. Apple /// users may be more used to drive references such as ".D1" but I think this standardized way is better.

Note: This program may also accept standard drive designations for each platform. But this feature is only an idea at this time and may not become real until this program has been used for real. This means that on the /// the user could type ".D1" and the program would accept this and use drive 1. On the // the user could type "#4" to mean drive 1 (I will need to check if this is the standard way of designating a drive in Apple // Pascal -- its been a L O N G time since I've delt with Apple // Pascal).

The program should provide as many defaults as possible to minimize user typing. For example, when the program asks for a disk drive the program should display a default drive specifier and if the user presses the Return key this default is used. Also, once an item is specified that item should retain that value when the value is prompted for again. For example, if the user enters 2 as the drive specifier, then this item should appear as the default for the next prompt that needs a drive specifier. Processing should show some type of indicator, for example a line of "*" characters when a disk's blocks are being sent.

Character case of inputs should be ignored (e.g. "x" = "X") and extra leading or trailing spaces should also be ignored (e.g. " david " = "david").

If S is pressed:

```
Disk Drive to Send [1] ? 2
Disk Name [Apple Writer /// 1.1] ?
Disk Format [Apple /// SOS] ?
Disk Blocks [280] ?
Date [24 February 1999] ?
```

APPLE /// 5.25" FLOPPY DISK IMAGE MAKER PROGRAM SPECIFICATION
Revision 2 -- David T. Craig -- 08 March 1999 -- 10 / 16

```
      Contact Name [David T. Craig] ?
      Contact Address [941 Calle Mejia #1006, Santa Fe, NM 87501 USA] ?
      Contact Email [71533.606@compuserve.com]  ?
      Sending 280 disk blocks to the serial port ...
      Blocks: ********************
```

If R is pressed:

```
      Disk Drive to Receive [2] 3
      Disk in drive 3 is not formatted, formatting disk ...
      Receiving disk image from serial port ...
      Blocks: ********************
```

If the disk in the drive to receive is already formatted then the following prompt should appear:

```
      WARNING -- Disk in drive 3 is already formatted.
      Overwrite this disk [N] (Y/N) ?
```

If H is pressed:

A brief description of this program should appear followed by a description of each command. For example,

"This program's purpose is to yadda yadda yadda ..."

If Q is pressed:

```
      Really Quit [Y] (Y/N) ?
```

The program should when it starts display information about itself such as the program name, version number, creation date, and author info. This information will be followed by the command line. For example,

```
      Apple /// Disk Image File Sender and Receiver Utility
      Version 1.0.0 / 24 Feburary 1999
      Written by David T. Craig
```

Future Features:

o Saving disk images to other attached drives such as a hard drive or a 3.5" 800K floppy. This would allow users to make disk images on their machine and later transmit them to another machine either directly via disk or via their own telecommunications program such as Access ///. I don't envision this feature being part of the first release but a definite possibility for version 2.

APPLE /// 5.25" FLOPPY DISK IMAGE MAKER PROGRAM SPECIFICATION
Revision 2 -- David T. Craig -- 08 March 1999 -- 11 / 16

This feature could exist for both the /// and the //.

o Saving non-140K disk drives such as the ProFile 5MB hard disk. This would allow users to dump the contents of these drives as a disk image. Recreating these images could be a problem that I need to check into further. The user could specify these devices via their device name such as .PROFILE. The user would need to know the number of blocks on these other devices.

This feature would only exist on the /// and not on the //.

8.0 APPLE // DISK SUPPORT

This program should also be implemented so that it runs on the Apple // family of machines (e.g. Apple][, Apple][+, Apple //e, Apple //c, Apple //gs). The reason for this is the low-level disk format of the Apple /// 5.25" disks and the Apple // fmaily 5.25" disks are the same (the directory structures may differ but that is immaterial at the low-level block level).

This Apple // family version program should also fully support the sending and receiving of Apple /// disk image files.

Disk drive specifiers may be different on the // version of this program since the // family does not use .D1, .D2, etc for drives. I will need to check how disks are specified in the // world. Or, the /// drive specifiers could be used and the // program would translate them to what it really needs. See the above discussion for more about this.

This program should use the Apple // family serial I/O devices such as the Super Serial Card in the Apple //e or the built-in serial port of the Apple //c machine (a //GS machine could also be used if this has a serial port, but I'm not familiar with this machine so I can't say more at this time).

This program will most likely not support automatic disk formatting on an Apple //-class machine when a disk image is received from an external machine since I believe the Apple // family OS does not support a formatting driver such as the /// supports. I know you can get the formatter for Apple // Pascal use but I don't have it and don't want to bother at this time with this since the user can just pre-format a floppy for disk creation use using a program such as Apple's ProDOS utility.

9.0 PROGRAMMING DETAILS

The program should be programmed in a high-level language and not assembly language. Why? I'm too old to deal with assembly any more and assembly is just not needed. I plan on using Apple /// Pascal for the /// implementation. For the // implmentation I plan to use Apple // Pascal 1.3 or 1.2.

I will most likely create the /// version first and then use the ///'s sources for the // implementation. The source should have conditional compilation statements governing the destination platform so that only a single instance of the source code need to exist. For example, the first line of the source could be:

```
{$SETC APPLE3 := TRUE}  { /// version use TRUE, // use FALSE }
```

Any areas of the program that are // or /// specific (e.g. disk formatting) would be bounded by the APPLE3 condition and do what is needed for that area.

Disk I/O will be achieved using the Pascal commands UNITREAD and UNITWRITE to read and write 512 byte disk blocks respectively.

Serial I/O will be achieved using the Pascal commands WRITELN and READLN output and input serial port data. The serial ports will be accessed using Pascal's device names REMIN: and REMOUT: (from an Apple // perspective) or the .RS232 device (from an Apple /// perspective).

Serial I/O configuring will not be done by this program. The user will need to make certain that their /// (or //) serial I/O are configured correctly for the remote machine they are using. The /// distribution disk for this program will come with the program and the necessary OS files so that the user can just boot this disk and this program will work with pre-configured serial I/O (e.g. 9600 baud, XON-XOFF flow control, ...). For the Apple // world users will most likely need to setup their serial I/O card's DIP switches (ugh!). The distribution disk should also come with this specification and the source code for both the /// and // programs (I believe in making all of this program public so others with an interest in how it works or want to modify it can have everything they need).

10.0 UTILITY PROGRAMS ON OTHER MACHINES

It may be useful to have some utility programs running on other machines (e.g. Macintosh) that deal with disk image files.

For example, there should exist a program that verifies text disk images files. Also, there may need to be a program that can convert a text disk image file to a binary disk image file for use by other programs that can deal with binary disk images (e.g. the various Apple // emulator programs seem to use binary images only). Likewise, there should be a utility that converts binary disk images to text files in case you want to send such a text image to my program to re-create a real 5.25" floppy disk.

These programs could all be the same program. I most likely will create such a utility for my Macintosh.

11.0 REFERENCES

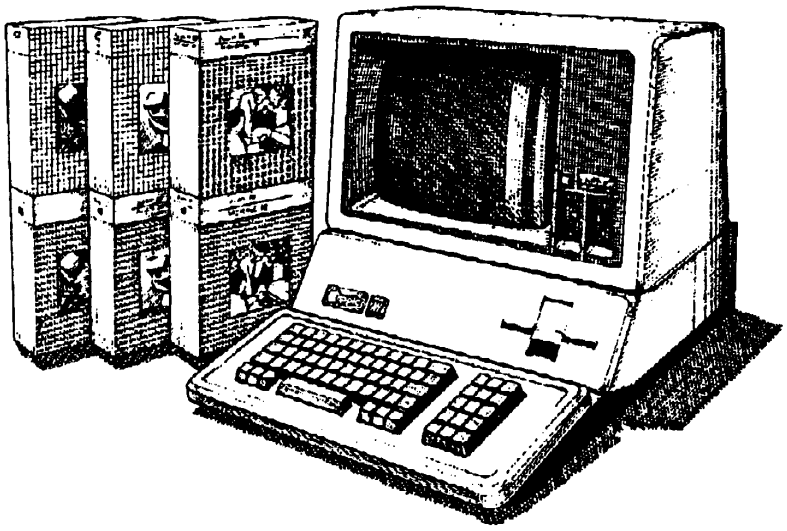
Apple /// Owner's Guide, Apple Computer, 1981

Apple /// Pascal Programmer's Manual (volumes 1 and 2), Apple Computer, 1981

"Extraordinary Pi" web site (www.users.globalnet.com.uk/~nickjh/pi.htm) for the Pi digits.

###

Apple /// Computer Information



David Craig's
Apple /// 5.25" Floppy Disk Image Maker
Program Specification

Revision 3
10 March 1999

"_029.PICT" 166 KB 2001-08-25 dpi: 300h x 300v pix: 2472h x 3213v

APPLE /// INFO

APPLE /// 5.25" FLOPPY DISK IMAGE MAKER PROGRAM SPECIFICATION

Revision 3 -- David T. Craig -- 10 March 1999

TABLE OF CONTENTS

0.0 REVISIONS

Date	Rev	Comments
1999-02-24	0	Created the preliminary spec.
1999-02-25	1	Added table of contents, spelling corrections.
1999-03-08	2	Added line checksum detail.
1999-03-10	3	Added comments about storing disk images on a CD-ROM. Changed checksum algorithm to not produce large numbers. Added D)evices command to command line. Improved reference section.

"_030.PICT" 141 KB 2001-08-25 dpi: 300h x 300v pix: 2472h x 3213v

APPLE /// INFO

1.0 AUTHOR

David T. Craig
941 Calle Mejia # 1006
Santa Fe, NM 87501 USA

home phone: (505) 820-0358
email: 71533.606@compuserve.com

2.0 AUDIENCE

Apple /// computer users who want to preserve their collections of Apple /// 5.25" 140K floppy disks.

Apple // computer users will also be interested in this program since it will support Apple // 140K floppy disks.

3.0 INTRODUCTION

This specification describes a proposed Apple /// program that will let users make disk image files from Apple /// 5.25" 140K floppy disks and also re-create these disks from a disk image file.

Apple // computer 140K disks can also be made into disk images (see section APPLE // DISK SUPPORT).

These disk image files can be stored on other machines for safe-keeping. For example, one could use a CD-ROM to store around 2,000 disk images files (at around 300K each).

Comments about this specification are welcome.

4.0 SCHEDULE

This program should hopefully be finished in about 2 weeks. The reason for this deadline is I have borrowed some Apple /// disks that I need to return in around 2 weeks and I want to make disk image files from these disks before returning them.

I plan to implement this program first on the Apple /// since the ///'s development system, Apple /// Pascal, can produce Apple // Pascal compatible programs.

5.0 GOALS

The goals of this program are:

- Provide a simple to use Apple /// program that allows users to save 5.25" Apple /// floppy disks to external media in a reliable manner.

- Send disk data out the Apple /// computer's serial port to another computer which will save the disk image files as regular files on its disk media. This other computer would only need a terminal program that can receive files via its serial ports.

- Receive disk image files to an Apple /// computer via its serial port and recreate the original disk onto an Apple /// formatted or unformatted floppy disk.

- Image file sending and receiving will be on a floppy disk block basis. A disk block contains 512 bytes.

- Support Apple // 5.25" formatted disks also. See section APPLE // DISK SUPPORT at the end for more details.

- Copy protected disks will not be supported. If a disk's blocks cannot all be read then no disk image will be created by this program.

6.0 IMAGE FILE FORMAT

Disk image files will have the following characteristics:

- Contain only human-readable characters (i.e. no control or Extended ASCII characters, except for the carriage return (ASCII 13) and line feed (ASCII 10) characters). Reason: So these files can easily be transferred between machines and over email networks without worrying about any special characters causing problems.

- Stored as text files with whatever machine operating system extension information is needed. For example, on PCs they would normally end in ".TXT", on Macs the internal document type should be "TEXT".

- Formatted on a line basis with each line terminated with carriage return and line feed characters. Reason: So Microsoft Windows-based machines and UNIX-based machines can view the files with their word processors. Macintosh machines should also be able to view these files though there will appear a space before all but the first line of the image file.

- Blank lines may exist and are ignored.

- Divided into disk block groups with each group headed by a comment specifying the block number starting with block 0.

- Disk blocks are divided into 32 byte chunks for a total of 16 chunks. Each 32 byte chunk appears as 64 characters with a leading address value and with a following ASCII representation of the bytes. Each 32 byte chunk is also terminated with a checksum value that is computed from the 32 bytes.

- File starts with a header and ends with a footer comment. All comments are stored on a line basis with each comment line starting with the ";" character marker (comment marker must start at position 1 of the line). Most header comments are not required but the footer comment is required. The first line of the file must be "APPLE 5.25 INCH DISK IMAGE FILE". The header comment specifies the following information:

Disk contents	e.g. Apple Writer /// 1.1
Disk format	e.g. Apple /// SOS
Disk block count	e.g. 280
Image creation date	e.g. 24 February 1999
Image program creator name	e.g. Apple3DiskImageMaker 1.0.0
Person who created image	e.g. David T. Craig

```
Address of person              e.g. 123 Main, City, State Zip USA
Person Contact info            e.g. 71533.606@compuserve.com
Misc. comment                  e.g. Really neat program
```

A sample header is:

```
; APPLE 5.25 INCH DISK IMAGE FILE
;
; DISK_NAME: Apple Writer /// 1.1
; DISK_FORMAT: Apple /// SOS
; DISK_BLOCKS: 280
; DATE: 24 February 1999
; CREATED_BY: Apple3DiskImageMaker 1.0.0
; CONTACT_NAME: David T. Craig
; CONTACT_ADDRESS: 941 Calle Mejia #1006, Santa Fe, NM 87501 USA
; CONTACT_EMAIL: 71533.606@compuserve.com
; COMMENT: Really neat program
```

The footer comment contains the following:

```
; DISK_CHECKSUM: nnnnnnnn
; FINIS
```

The "FINIS" line MUST be the last line in the image file otherwise the disk restoration will see the file as corrupt.

- Missing header or footer information will cause the image file to be seen as corrupted and will not be processed. Incorrect checksums indicate disk images are corrupt.

- Image files will be verified on a line basis. Verification will test that only human readable characters exist (line feed at start of a line is OK), line address is correct, data section is correct (has correct length, no spaces, correct hex characters), ASCII characters are correct (length, characters), and the line checksum is correct for the whole line.

- Each disk block begins with the following comment (the $ number is the hexadecimal number for the block followed by a "/" then the block number in decimal):

```
; BLOCK: $0081/129
```

Each block line contains the following sections:

```
block(hex) address(hex): data(hex) [ascii] CHKSUM: checksum(dec)
```

```
block(hex)  - block number in hex
addreses(hex)     - 4 hex digits teminated with ":"
data(hex)         - 64 hex digits
[ascii]           - 32 characters bound by "[" and "]"
                  - control codes, spaces, DEL and extended are "."
checksum(dec)     - decimal checksum of whole line

A sample block with a few lines is:

; BLOCK: $0000/0
;
0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 12345
0000 0020: ...
0000 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 12345
```

o The checksum is a decimal value that is calculated based on ALL of the characters in the line. The calculation is a repeated sum of all of the line character ASCII values multipled against a specific digit "key". The "key" here are the digits of the mathematical number Pi (3.14159265...). For example, if the line contains the following:

```
     "DAVID"

then the checksum will be calculated as follows:

     PI to 255 digits:

          3141592653 5897932384 6264338327 9502884197 1693993751
          0582097494 4592307816 4062862089 9862803482 5342117067
          9821480865 1328230664 7093844609 5505822317 2535940812
          8481117450 2841027019 3852110555 9644622948 9549303819
          6442881097 5665933446 1284756482 3378678316 5271201909
          14564

     ASCII "D" = 68    "3" = 3
     ASCII "A" = 65    "1" = 1
     ASCII "V" = 86    "4" = 4
     ASCII "I" = 73    "1" = 1
     ASCII "D" = 68    "5" = 5

     Checksum = (68 x 3)+(65 x 1)+(86 x 4)+(73 x 1)+(68 x 5)
              = 204 + 65 + 344 + 73 + 340
              = 1026
```

Illustrative programming code in the 4th Dimension language by ACI of France for this checksum follows:

```
` 255 digitis of the number Pi.  This means this routine can compute
` the checksum for strings with up to 255 characters.
`
` This string should be initialized at the beginning of a program
` so that it doesn't have to initialized each time this routine
` is used.

$pi :=

"3141592653589793238462643383279502884197169399375105820974944" +
"5923078164062862089986280348253421170679821480865132823066 4" +
"7093844609550582231725359408128481117450284102701938521105 55" +
"9644622948954930381964428810975665933446128475648233786783 16" +
"527120190914564"

Repeat
      $data := Request("Enter data to checksum:";"")

      If ($data # "")
            $checksum := 0

            For ($i;1;Length($data))
                  $data_ascii := Ascii( Substring($data;$i;1) )
                  $pi_digit := Ascii( Substring($pi;$i;1) ) - 48

                  $checksum := $checksum + ($data_ascii * $pi_digit)
            End for

            ALERT("Checksum = " + String($checksum))
      End if
Until ($data="")
```

The checksum is calculated from all of the data in a single line. For example, the above sample disk block line would have a checksum calculated on the following data which includes the spaces and other non-disk block data:

"0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67]"

(Side Note) Why is this algorithm used? I have used the more traditional checksum algorithms such as exclusive-ORs and CRCs, but wanted something different for this program. This algorithm should prove sufficient and

besides that, I have an interest in the number Pi and wanted to use it in some real world situation.

Note: The header portion of the disk image file contains a CHECKSUM_VERSION line which tells programs that process these disk image files what checksum algorithm is used. This version information exists in case this algorithm changes in the future, e.g. to a more robust algorithm. The current checksum version is 1, the next will be 2, etc.

o The footer checksum is calculated based on the checksums of the individual lines. This checksum is just the sum of the weighted ASCII values of the digits in the line checksums. The weights here are the digits of the number pi. For example, if you have a file with 2 lines with the line checksums "12" and "3" then the footer checksum would be:

```
Footer Checksum  = ASCII("1")*3 + ASCII("2")*1 + ASCII("3")*4
                 = (48 * 3) + (49 * 1) + (50 * 4)
                 = 144 + 49 + 200
                 = 393
```

o Sample skeleton image file is:

```
; APPLE 5.25 INCH DISK IMAGE FILE
;
; DISK_NAME: Apple Writer /// 1.1
; DISK_FORMAT: Apple /// SOS
; DISK_BLOCKS: 280
; DATE: 24 February 1999
; CREATED_BY: Apple3DiskImageMaker 1.0.0
; CONTACT_NAME: David T. Craig
; CONTACT_ADDRESS: 941 Calle Mejia #1006, Santa Fe, NM 87501 USA
; CONTACT_EMAIL: 71533.606@compuserve.com
; COMMENT: Really neat program

; CHECKSUM_VERSION: 1

; BLOCK: $0000/0
;
0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 344785
0000 0020: ...
0000 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 56873

...

; BLOCK: $0117/279
```

```
;
0117 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 437823
0117 0020: ...
0117 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 5623445

; DISK_CHECKSUM: 57856
; FINIS
```

7.0 USER INTERFACE

The disk image program will be named Apple3DiskImageMaker. When the user runs this program it asks the user what to do, mainly either send a disk image our a serial port or receive a disk image from an external machine. The user interface should be a simple command-line inteface (CLI) with the following contents (Apple /// and // Pascal [and Apple Lisa Workshop] users or P-System users in general will be familiar with this simple but very functional command interface):

S)end-disk R)eceive-image F)ormat V)erify-disk D)evices H)elp Q)uit ?

Disk drives are specified by numbers. For example, the number 1 means the first disk drive, number 2 the second disk drive, etc. For Apple /// users, drive 1 means .D1, drive 2 means .D2, etc. For Apple // users, drive 1 means unit 4, drive 2 unit 5, drive 3 unit 9, drive 4 unit 10. This is done so that this program has consistent disk drive references for both the Apple /// and the Apple // platforms. Apple /// users may be more used to drive references such as ".D1" but I think this standardized way is better.

Note: This program may also accept standard drive designations for each platform. But this feature is only an idea at this time and may not become real until this program has been used for real. This means that on the /// the user could type ".D1" and the program would accept this and use drive 1. On the // the user could type "#4" to mean drive 1 (I will need to check if this is the standard way of designating a drive in Apple // Pascal -- its been a L O N G time since I've delt with Apple // Pascal).

The program should provide as many defaults as possible to minimize user typing. For example, when the program asks for a disk drive the program should display a default drive specifier and if the user presses the Return key this default is used. Also, once an item is specified that item should retain that value when the value is prompted for again. For example, if the user enters 2 as the drive specifier, then this item should appear as the default for the next prompt that needs a drive specifier. Processing should show some type of indicator, for example a line of "*" characters when a disk's blocks are being sent.

Character case of inputs should be ignored (e.g. "x" = "X") and extra leading or trailing spaces should also be ignored (e.g. " david " = "david").

Here's a description of each command with sample command output:

S)end-disk

Purpose: Send a disk's image out a serial port. This command should ideally determine the number of blocks on a disk and use that instead of always using 280 blocks.

```
Disk Drive to Send Disk Image [1] ? 2
Disk Name [Apple Writer /// 1.1] ?
Disk Format [Apple /// SOS] ?
Disk Blocks [280] ?
Date [24 February 1999] ?
Contact Name [David T. Craig] ?
Contact Address [941 Calle Mejia #1006, Santa Fe, NM 87501 USA] ?
Contact Email [71533.606@compuserve.com] ?
Sending 280 disk blocks to the serial port ...
Blocks Sent: ********************
```

R)eceive-image

Purpose: Receive a disk's image from a serial port and save to a floppy. If the disk image specifies a disk size that differs from the receiving disk then the reception should not take place.

```
Disk Drive to Receive Disk Image [2] ? 3
Disk in drive 3 is not formatted, formatting disk ...
Receiving disk image from serial port to 280 block disk ...
Blocks Received: ********************
```

Note: If the disk in the drive to receive is already formatted then the following prompt should appear (the default for such destructive actions should always be No):

```
WARNING -- Disk in drive 3 is already formatted.
Overwrite this disk (Y/N) [N] ?
```

F)ormat

Purpose: Formats a disk, no directory information is put on the disk.

```
Disk Drive to Format [2] ? 3
Formatting disk in drive 3 ...
```

Note: If the disk in the drive to receive the image is already formatted then the following prompt should appear (the default for such destructive actions should always be No):

```
WARNING -- Disk in drive 3 is already formatted.
```

```
     Overwrite this disk (Y/N) [N] ?
```

V)erify-disk

Purpose: Verify a disk's blocks by reading all the blocks. The blocks may also be written after the read to make certain that the blocks can be both read and written. The read and written data will be compared.

```
Disk Drive to Verify [2] ? 3
Verifying disk in drive 3 ...
Blocks Verified: ********************
```

D)evices

Purpose: Lists the devices connected to your computer. For the Apple /// the SOS device numbers, device unit numbers, and device names will appear (for block devices the number of blocks will also be listed). For the Apple // most likely only the disk device unit numbers will be listed. May also list if a disk is write-protected.

H)elp

Purpose: Display a brief description of this program.

"This program's purpose is to yadda yadda yadda ..."

Q)uit

Purpose: Quits the program, but asks the user first.

```
Really Quit (Y/N) [Y] ?
```

When this program starts it will display information about itself such as the program name, version number, creation date, and author info. This information will be followed by the command line. For example,

```
Apple /// Disk Image File Sender and Receiver Utility
Version 1.0.0 -- 24 Feburary 1999
Written by David T. Craig
```

Future Features:

- Saving disk images to other attached drives such as a hard drive or a 3.5" 800K floppy. This would allow users to make disk images on their machine and later transmit them to another machine either directly via disk or via their own telecommunications program such as Access ///. I don't envision this feature being part of the first release but a definite possibility for version 2.

  This feature could exist for both the /// and the //.

- Saving non-140K disk drives such as the ProFile 5MB hard disk. This would allow users to dump the contents of these drives as a disk image. Recreating these images could be a problem that I need to check into further. The user could specify these devices via their device name such as .PROFILE. The user would need to know the number of blocks on these other devices.

  This feature would only exist on the /// and not on the //.

8.0 APPLE // DISK SUPPORT

This program should also be implemented so that it runs on the Apple // family of machines (e.g. Apple][, Apple][+, Apple //e, Apple //c, Apple //gs). The reason for this is the low-level disk format of the Apple /// 5.25" disks and the Apple // fmaily 5.25" disks are the same (the directory structures may differ but that is immaterial at the low-level block level).

This Apple // family version program should also fully support the sending and receiving of Apple /// disk image files.

Disk drive specifiers may be different on the // version of this program since the // family does not use .D1, .D2, etc for drives. I will need to check how disks are specified in the // world. Or, the /// drive specifiers could be used and the // program would translate them to what it really needs. See the above discussion for more about this.

This program should use the Apple // family serial I/O devices such as the Super Serial Card in the Apple //e or the built-in serial port of the Apple //c machine (a //GS machine could also be used if this has a serial port, but I'm not familiar with this machine so I can't say more at this time).

This program will most likely not support automatic disk formatting on an Apple //-class machine when a disk image is received from an external machine. I believe the Apple // family OS does not support a formatting driver such as the /// supports. I know you can get the formatter for Apple // Pascal use but I don't have it and don't want to bother at this time with this since the user can just pre-format a floppy for disk creation use using a program such as Apple's ProDOS utility.

9.0 PROGRAMMING DETAILS

The program should be programmed in a high-level language and not assembly language. Why? I'm too old to deal with assembly any more and assembly is just not needed. I plan on using Apple /// Pascal for the /// implementation. I will most likely create the /// version first and then use the ///'s sources for the // implementation. The source should have conditional compilation statements governing the destination platform so that only a single instance of the source code need to exist. For example, the first line could be:

```
{$SETC APPLE3 := TRUE}  { /// version use TRUE, // use FALSE }
```

Any areas of the program that are // or /// specific (e.g. disk formatting) would be bounded by the APPLE3 condition and do what is needed for that area.

Disk I/O will be achieved using the Pascal intrinsic commands UNITREAD and UNITWRITE to read and write 512 byte disk blocks respectively. Special disk operations such as disk formatting will be done using Pascal's UNITSTATUS command (Apple /// SOS calls may also need to be made to handle such tasks as getting a SOS device's Pascal unit number whcih the UNITREAD/WRITE commands require).

Serial I/O will be achieved using the Pascal commands WRITELN and READLN output and input serial port data. The serial ports will be accessed using Pascal's device names REMIN: and REMOUT: (from an Apple // perspective) or the .RS232 device (from an Apple /// perspective).

Serial I/O configuring will not be done by this program. The user will need to make certain that their /// (or //) serial I/O are configured correctly for the remote machine they are using. The /// distribution disk for this program will come with the program and the necessary OS files so that the user can just boot this disk and this program will work with pre-configured serial I/O (e.g. 9600 baud, XON-XOFF flow control, ...). For the Apple // world users will most likely need to setup their serial I/O card's DIP switches (ugh!). The distribution disk should also come with this specification and the source code for both the /// and // programs (I believe in making all of this program public so others with an interest in how it works or want to modify it can have everything they need).

The Apple // version of this program will be created with Apple /// Pascal using the {$SETC APPLE:=2} conditional compilation facility of the /// Pascal compiler (see page 109 of the Apple /// Pascal Prgorammer's Manual, volume 2 dated 1981). This allows a single source to exist for this program even though the same source will be used for two different target machines.

10.0 UTILITY PROGRAMS ON OTHER MACHINES

It may be useful to have some utility programs running on other machines (e.g. Macintosh) that deal with disk image files.

For example, there should exist a program that verifies text disk images files. Also, there may need to be a program that can convert a text disk image file to a binary disk image file for use by other programs that can deal with binary disk images (e.g. the various Apple // emulator programs seem to use binary images only). Likewise, there should be a utility that converts binary disk images to text files in case you want to send such a text image to my program to re-create a real 5.25" floppy disk.

These programs could all be the same program. I most likely will create such a utility for my Macintosh.

11.0 REFERENCES

Apple /// Owner's Guide
Apple Computer, 1981

Apple /// Plus Owner's Guide
Apple Computer, 1982

Apple /// Standard Device Drivers Manual
Apple Computer, 1981

Apple /// Pascal Programmer's Manual (volumes 1 and 2)
Apple Computer, 1981

Apple /// Pascal Technical Reference Manual
Apple Computer, 1983

/// Bits: John Jeppson's Guided Tour of Highway ///
Softalk magazine, May 1983

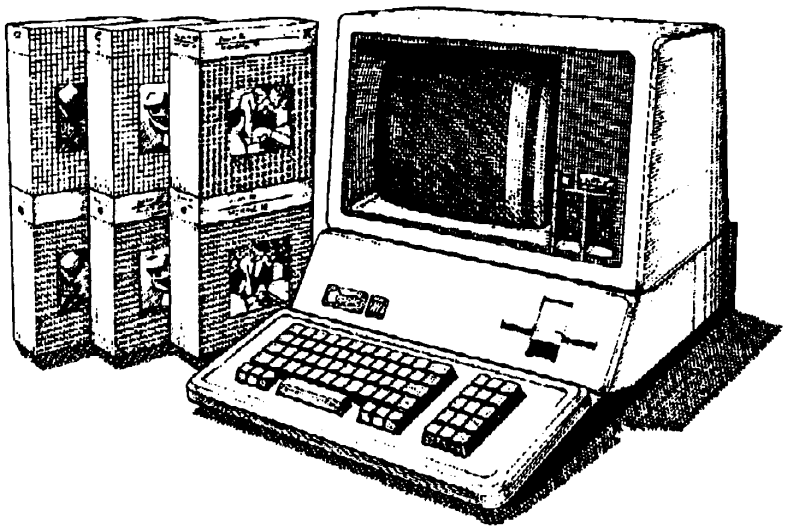
A Little SOS with your Pascal
Tim O'Konski, BYTE magazine, December 1982

Pascal source code to format SOS devices
James Vandermade, no date

Extraordinary Pi
Web site (www.users.globalnet.com.uk/~nickjh/pi.htm)
This is where I obtained the Pi digits.

F I N I S

Apple /// Computer Information



David Craig's Apple /// 5.25" Floppy Disk Image Maker Program Specification

Revision 4
11 March 1999

APPLE /// INFO

APPLE ///
5.25" 140K FLOPPY DISK
IMAGE MAKER PROGRAM
SPECIFICATION

Revision 4 -- David T. Craig -- 10 March 1999

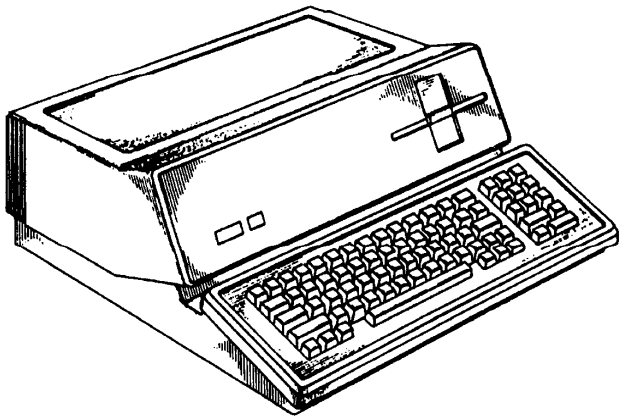


TABLE OF CONTENTS

APPLE /// INFO

0.0 REVISIONS

Date	Rev	Comments
1999-02-24	0	Created the preliminary spec.
1999-02-25	1	Added table of contents, spelling corrections.
1999-03-08	2	Added line checksum detail.
1999-03-10	3	Added comments about storing disk images on a CD-ROM. Changed checksum algorithm to not produce large numbers. Added D)evices command to command line. Improved reference section.
1999-03-11	4	Added Apple /// picture to first page. Corrected the checksum examples and added more checksum info. Centered all section headings, enlarged font size.

1.0 AUTHOR

David T. Craig
941 Calle Mejia # 1006
Santa Fe, NM 87501 USA

home phone: (505) 820-0358
email: 71533.606@compuserve.com

2.0 AUDIENCE

Apple /// computer users who want to preserve their collections of Apple /// 5.25" 140K floppy disks.

Apple // computer users will also be interested in this program since it will support Apple // 140K floppy disks.

3.0 INTRODUCTION

This specification describes a proposed Apple /// program that will let users make disk image files from Apple /// 5.25" 140K floppy disks and also re-create these disks from a disk image file.

Apple // computer 140K disks can also be made into disk images (see section APPLE // DISK SUPPORT).

These disk image files can be stored on other machines for safe-keeping. For example, one could use a CD-ROM to store around 2,000 disk images files (at around 300K each).

Comments about this specification are welcome.

4.0 SCHEDULE

This program should hopefully be finished in about 2 weeks. The reason for this deadline is I have borrowed some Apple /// disks that I need to return in around 2 weeks and I want to make disk image files from these disks before returning them.

I plan to implement this program first on the Apple /// since the ///'s development system, Apple /// Pascal, can produce Apple // Pascal compatible programs.

5.0 GOALS

The goals of this program are:

- Provide a simple to use Apple /// program that allows users to save 5.25" Apple /// floppy disks to external media in a reliable manner.
- Send disk data out the Apple /// computer's serial port to another computer which will save the disk image files as regular files on its disk media. This other computer would only need a terminal program that can receive files via its serial ports.
- Receive disk image files to an Apple /// computer via its serial port and recreate the original disk onto an Apple /// formatted or unformatted floppy disk.
- Image file sending and receiving will be on a floppy disk block basis. A disk block contains 512 bytes.
- Support Apple // 5.25" formatted disks also. See section APPLE // DISK SUPPORT at the end for more details.
- Copy protected disks will not be supported. If a disk's blocks cannot all be read then no disk image will be created by this program.

6.0 IMAGE FILE FORMAT

Disk image files have the following characteristics:

- Contain only human-readable characters (i.e. no control or Extended ASCII characters, except for the carriage return (ASCII 13) and line feed (ASCII 10) characters). Reason: So these files can easily be transferred between machines and over email networks without worrying about any special characters causing problems.

- Stored as text files with whatever machine operating system extension information is needed. For example, on PCs they would normally end in ".TXT", on Macs the internal document type should be "TEXT".

- Formatted on a line basis with each line terminated with carriage return and line feed characters. Reason: So Microsoft Windows-based machines and UNIX-based machines can view the files with their word processors. Macintosh machines should also be able to view these files though there will appear a space before all but the first line of the image file.

- Blank lines may exist and are ignored.

- Divided into disk block groups with each group headed by a comment specifying the block number starting with block 0.

- Disk blocks are divided into 32 byte chunks for a total of 16 chunks. Each 32 byte chunk appears as 64 characters with a leading address value and with a following ASCII representation of the bytes. Each 32 byte chunk is also terminated with a checksum value that is computed from the 32 bytes.

- File starts with a header and ends with a footer comment. All comments are stored on a line basis with each comment line starting with the ";" character marker (comment marker must start at position 1 of the line). Most header comments are not required but the footer comment is required. The first line of the file must be "APPLE 5.25 INCH DISK IMAGE FILE". The header comment specifies the following information:

Disk contents	e.g. Apple Writer /// 1.1
Disk format	e.g. Apple /// SOS
Disk block count	e.g. 280
Image creation date	e.g. 24 February 1999
Image program creator name	e.g. Apple3DiskImageMaker 1.0.0
Person who created image	e.g. David T. Craig
Address of person	e.g. 123 Main, City, State Zip USA
Person Contact info	e.g. 71533.606@compuserve.com
Misc. comment	e.g. Really neat program

A sample header is:

```
; APPLE 5.25 INCH DISK IMAGE FILE
```

APPLE /// INFO

```
;
; DISK_NAME:          Apple Writer /// 1.1
; DISK_FORMAT:        Apple /// SOS
; DISK_BLOCKS:        280
; DATE:               24 February 1999
; CREATED_BY:         Apple3DiskImageMaker 1.0.0
; CONTACT_NAME:       David T. Craig
; CONTACT_ADDRESS:    941 Calle Mejia #1006, Santa Fe, NM 87501 USA
; CONTACT_EMAIL:      71533.606@compuserve.com
; COMMENT:            Really neat program
```

The spaces after the tokens (e.g. "DISK_NAME") are not important but having all the token information line up makes the files much more readable to humans.

The footer comment contains the following lines:

```
; DISK_CHECKSUM: nnnnnnnn
;
; FINIS
```

The "FINIS" line MUST be the last line in the image file otherwise the disk restoration will see the file as corrupt. The "DISK_CHECKSUM" line must also exist.

- Missing header or footer information will cause the image file to be seen as corrupted and will not be processed. Incorrect checksums indicate disk images are corrupt and should not be used.

- Image files will be verified on a line basis. Verification will test that only human readable characters exist (line feed at start of a line is OK), line address is correct, data section is correct (has correct length, no spaces, correct hex characters), ASCII characters are correct (length, characters), and the line checksum is correct for the whole line.

- Each disk block begins with the following comment (the $ number is the hexadecimal number for the block followed by a "/" then the block number in decimal):

```
; BLOCK: $0081/129
```

Each block line contains the following sections:

```
block(hex) address(hex): data(hex) [ascii] CHKSUM: checksum(dec)
```

```
block(hex)      - block number in hex
addreses(hex)   - 4 hex digits teminated with ":"
data(hex)       - 64 hex digits
[ascii]         - 32 characters bound by "[" and "]"
                - control codes, spaces, DEL and extended are "."
checksum(dec)   - decimal checksum of whole line
```

A sample block with a few lines is:

```
APPLE /// 5.25" 140K FLOPPY DISK IMAGE MAKER PROGRAM SPECIFICATION
      Revision 4 -- David T. Craig -- 11 March 1999 -- 6 / 17
```

A sample block with a few lines is:

```
; BLOCK: $0000/0
;
0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 12345
0000 0020: ...
0000 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 12345
```

A sample line that could appear in an image file (and not one of the above made-up lines) follows:

```
0000 0000: 1236754ABF3428976EF678876CD67834
           0AAFF678234AECBBD897234879FE8796
           [ffdU YREHJ7 CJ.fdsjkhwre..23fw1%] CHKSUM: 22392
```

Note: I have broken this line into 3 lines so it would fit nicely in this document, it would appear as a single line in a real disk image file. This line also has a correct checksum value, 22392. See the next paragraph for checksum calculation deatails.

o The checksum is a decimal value that is calculated based on ALL of the characters in the line. The calculation is a repeated sum of all of the line character ASCII values multipled against a specific digit "key". The "key" here consists of the digits of the mathematical number Pi (3.14159265...). For example, if the line contains the following:

"DAVID"

then the checksum will be calculated as follows:

Pi to 255 digits:

```
3141592653 5897932384 6264338327 9502884197 1693993751
0582097494 4592307816 4062862089 9862803482 5342117067
9821480865 1328230664 7093844609 5505822317 2535940812
8481117450 2841027019 3852110555 9644622948 9549303819
6442881097 5665933446 1284756482 3378678316 5271201909
14564
```

```
ASCII "D" = 68       "3" = 3
ASCII "A" = 65       "1" = 1
ASCII "V" = 86       "4" = 4
ASCII "I" = 73       "1" = 1
ASCII "D" = 68       "5" = 5

Checksum = (68 x 3)+(65 x 1)+(86 x 4)+(73 x 1)+(68 x 5)
         = 204 + 65 + 344 + 73 + 340
         = 1026
```

Illustrative programming code in the 4th Dimension language by ACI of France for this checksum follows:

```
` 255 digits of the number Pi.  This means this routine can compute
` the checksum for strings with up to 255 characters.
`
` This string should be initialized at the beginning of a program
` so that it doesn't have to be initialized each time this routine
` is used therby making this routine run faster.

$pi :=

     "31415926535897932384626433832795028841971693993751058209749 4" +
     "45923078164062862089986280348253421170679821480865132823066 4" +
     "70938446095505822317253594081284811174502841027019385211055 5" +
     "96446229489549303819644288109756659334461284756482337867831 6" +
     "527120190914564"

Repeat
     $data := Request("Enter data to checksum:";"")

     If ($data # "")
          $checksum := 0

          For ($i;1;Length($data))
               $data_ascii     := Ascii( Substring($data;$i;1) )
               $pi_digit       := Ascii( Substring($pi;$i;1) ) - 48

               $checksum       := $checksum + ($data_ascii * $pi_digit)
          End for

          ALERT("Checksum = " + String($checksum))
     End if
Until ($data="")
```

The checksum is calculated from all of the data in a single line. For example, the above sample disk block line would have a checksum calculated on the following data which includes the spaces and other non-disk block data:

```
"0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67]"
```

(Side Note) Why is this algorithm used? I have used the more traditional checksum algorithms such as exclusive-ORs and CRCs, but wanted something different for this program. This algorithm should prove sufficient and besides that, I have an interest in the number Pi and wanted to use it in some real world situation.

Note: The header portion of the disk image file contains a CHECKSUM_VERSION line which tells programs that process these disk image files what checksum algorithm is used. This version information exists in case this algorithm changes in the future, e.g. to a more robust algorithm. The current checksum version is 1, the next will be 2, etc.

- The footer checksum is calculated based on the checksums of the individual lines. This checksum is just the sum of the weighted digit values of the digits in the line checksums. The weights here are the digits of the number pi. For example, if you have a file with 2 lines with the line checksums "12" and "3" then the footer checksum would be:

```
Footer Checksum      = (1 * 3) + (2 * 1) + (3 * 3)
                     = 3 + 2 + 9
                     = 14
```

Here's an ACI 4th Dimension program that calculates the footer checksum (this assumes $pi has been initialized):

```
Repeat
     $data := Request("Enter data to checksum:";"")

     If ($data # "")
          $checksum := 0

          For ($i;1;Length($data))
               $data_digit     := Ascii(Substring($data;$i;1))-48
               $pi_digit       := Ascii(Substring($pi;$i;1))-48

               $checksum := $checksum + ($data_digit * $pi_digit)
          End for

          ALERT("Checksum = "+String($checksum))
     End if
Until ($data="")
```

- Sample skeleton image file is:

```
; APPLE 5.25 INCH DISK IMAGE FILE
;
; DISK_NAME:          Apple Writer /// 1.1
; DISK_FORMAT:        Apple /// SOS
; DISK_BLOCKS:        280
; DATE:               24 February 1999
; CREATED_BY:         Apple3DiskImageMaker 1.0.0
; CONTACT_NAME:       David T. Craig
; CONTACT_ADDRESS:    941 Calle Mejia #1006, Santa Fe, NM 87501 USA
; CONTACT_EMAIL:      71533.606@compuserve.com
; COMMENT:            Really neat program

; CHECKSUM_VERSION: 1

; BLOCK: $0000/0
;
0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 344785
0000 0020: ...
0000 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 56873

...

; BLOCK: $0117/279
;
0117 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 437823
0117 0020: ...
0117 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 5623445

; DISK_CHECKSUM: 57856
; FINIS
```

7.0 USER INTERFACE

The disk image program will be named Apple3DiskImageMaker. When the user runs this program it asks the user what to do, mainly either send a disk image out a serial port or receive a disk image from an external machine. The user interface should be a simple command-line inteface (CLI) with the following contents (Apple /// and // Pascal [and Apple Lisa Workshop] users or P-System users in general will be familiar with this simple but very functional command interface):

S)end-disk R)eceive-image F)ormat V)erify-disk D)evices H)elp Q)uit ?

Disk drives are specified by numbers. For example, the number 1 means the first disk drive, number 2 the second disk drive, etc. For Apple /// users, drive 1 means .D1, drive 2 means .D2, etc. For Apple // users, drive 1 means unit 4, drive 2 unit 5, drive 3 unit 9, drive 4 unit 10. This is done so that this program has consistent disk drive references for both the Apple /// and the Apple // platforms. Apple /// users may be more used to drive references such as ".D1" but I think this standardized way is better.

Note: This program may also accept standard drive designations for each platform. But this feature is only an idea at this time and may not become real until this program has been used for real. This means that on the /// the user could type ".D1" and the program would accept this and use drive 1. On the // the user could type "#4" to mean drive 1 (I will need to check if this is the standard way of designating a drive in Apple // Pascal -- its been a L O N G time since I've delt with Apple // Pascal).

The program should provide as many defaults as possible to minimize user typing. For example, when the program asks for a disk drive the program should display a default drive specifier and if the user presses the Return key this default is used. Also, once an item is specified that item should retain that value when the value is prompted for again. For example, if the user enters 2 as the drive specifier, then this item should appear as the default for the next prompt that needs a drive specifier. Processing should show some type of indicator, for example a line of "*" characters when a disk's blocks are being sent.

Character case of inputs should be ignored (e.g. "x" = "X") and extra leading or trailing spaces should also be ignored (e.g. " david " = "david").

Here's a description of each command with sample command output:

S)end-disk

Purpose: Send a disk's image out a serial port. This command should ideally determine the number of blocks on a disk and use that instead of always using 280 blocks.

```
Disk Drive to Send Disk Image [1] ? 2
Disk Name [Apple Writer /// 1.1] ?
Disk Format [Apple /// SOS] ?
```

```
      Disk Blocks [280] ?
      Date [24 February 1999] ?
      Contact Name [David T. Craig] ?
      Contact Address [941 Calle Mejia #1006, Santa Fe, NM 87501 USA] ?
      Contact Email [71533.606@compuserve.com] ?
      Sending 280 disk blocks to the serial port ...
      Blocks Sent: ********************

R)eceive-image

      Purpose:  Receive a disk's image from a serial port and save to a floppy.  If
      the disk image specifies a disk size that differs from the receiving disk
      then the reception should not take place.

      Disk Drive to Receive Disk Image [2] ? 3
      Disk in drive 3 is not formatted, formatting disk ...
      Receiving disk image from serial port to 280 block disk ...
      Blocks Received: ********************

      Note:  If the disk in the drive to receive is already formatted then the
      following prompt should appear (the default for such destructive actions
      should always be No):

      WARNING -- Disk in drive 3 is already formatted.
      Overwrite this disk (Y/N) [N] ?

F)ormat

      Purpose:  Formats a disk, no directory information is put on the disk.

      Disk Drive to Format [2] ? 3
      Formatting disk in drive 3 ...

      Note:  If the disk in the drive to receive the image is already formatted
      then the following prompt should appear (the default for such destructive
      actions should always be No):

      WARNING -- Disk in drive 3 is already formatted.
      Overwrite this disk (Y/N) [N] ?

V)erify-disk

      Purpose:  Verify a disk's blocks by reading all the blocks.  The blocks may
      also be written after the read to make certain that the blocks can be both
      read and written.  The read and written data will be compared.

      Disk Drive to Verify [2] ? 3
      Verifying disk in drive 3 ...
      Blocks Verified: ********************

D)evices
```

APPLE /// INFO

Purpose: Lists the devices connected to your computer. For the Apple /// the SOS device numbers, device unit numbers, and device names will appear (for block devices the number of blocks will also be listed). For the Apple // most likely only the disk device unit numbers will be listed. May also list if a disk is write-protected.

H)elp

Purpose: Display a brief description of this program.

"This program's purpose is to yadda yadda yadda ..."

Q)uit

Purpose: Quits the program, but asks the user first.

Really Quit (Y/N) [Y] ?

When this program starts it will display information about itself such as the program name, version number, creation date, and author info. This information will be followed by the command line. For example,

Apple /// Disk Image File Sender and Receiver Utility
Version 1.0.0 -- 24 Feburary 1999
Written by David T. Craig

Future Features:

- **Saving disk images to other attached drives such as a hard drive or a 3.5" 800K floppy. This would allow users to make disk images on their machine and later transmit them to another machine either directly via disk or via their own telecommunications program such as Access ///. I don't envision this feature being part of the first release but a definite possibility for version 2.**

  This feature could exist for both the /// and the //.

- **Saving non-140K disk drives such as the ProFile 5MB hard disk. This would allow users to dump the contents of these drives as a disk image. Recreating these images could be a problem that I need to check into further. The user could specify these devices via their device name such as .PROFILE. The user would need to know the number of blocks on these other devices.**

  This feature would only exist on the /// and not on the //.

8.0 APPLE // DISK SUPPORT

This program should also be implemented so that it runs on the Apple // family of machines (e.g. Apple][, Apple][+, Apple //e, Apple //c, Apple //gs). The reason for this is the low-level disk format of the Apple /// 5.25" disks and the Apple // fmaily 5.25" disks are the same (the directory structures may differ but that is immaterial at the low-level block level).

This Apple // family version program should also fully support the sending and receiving of Apple /// disk image files.

Disk drive specifiers may be different on the // version of this program since the // family does not use .D1, .D2, etc for drives. I will need to check how disks are specified in the // world. Or, the /// drive specifiers could be used and the // program would translate them to what it really needs. See the above discussion for more about this.

This program should use the Apple // family serial I/O devices such as the Super Serial Card in the Apple //e or the built-in serial port of the Apple //c machine (a //GS machine could also be used if this has a serial port, but I'm not familiar with this machine so I can't say more at this time).

This program will most likely not support automatic disk formatting on an Apple //-class machine when a disk image is received from an external machine. I believe the Apple // family OS does not support a formatting driver such as the /// supports. I know you can get the formatter for Apple // Pascal use but I don't have it and don't want to bother at this time with this since the user can just pre-format a floppy for disk creation use using a program such as Apple's ProDOS utility.

9.0 PROGRAMMING DETAILS

The program should be programmed in a high-level language and not assembly language. Why? I'm too old to deal with assembly any more and assembly is just not needed. I plan on using Apple /// Pascal for the /// implementation. I will most likely create the /// version first and then use the ///'s sources for the // implementation. The source should have conditional compilation statements governing the destination platform so that only a single instance of the source code need to exist. For example, the first line could be:

```
{$SETC APPLE3 := TRUE}  { /// version use TRUE, // use FALSE }
```

Any areas of the program that are // or /// specific (e.g. disk formatting) would be bounded by the APPLE3 condition and do what is needed for that area.

Disk I/O will be achieved using the Pascal intrinsic commands UNITREAD and UNITWRITE to read and write 512 byte disk blocks respectively. Special disk operations such as disk formatting will be done using Pascal's UNITSTATUS command (Apple /// SOS calls may also need to be made to handle such tasks as getting a SOS device's Pascal unit number whcih the UNITREAD/WRITE commands require).

Serial I/O will be achieved using the Pascal commands WRITELN and READLN output and input serial port data. The serial ports will be accessed using Pascal's device names REMIN: and REMOUT: (from an Apple // perspective) or the .RS232 device (from an Apple /// perspective).

Serial I/O configuring will not be done by this program. The user will need to make certain that their /// (or //) serial I/O are configured correctly for the remote machine they are using. The /// distribution disk for this program will come with the program and the necessary OS files so that the user can just boot this disk and this program will work with pre-configured serial I/O (e.g. 9600 baud, XON-XOFF flow control, ...). For the Apple // world users will most likely need to setup their serial I/O card's DIP switches (ugh!). The distribution disk should also come with this specification and the source code for both the /// and // programs (I believe in making all of this program public so others with an interest in how it works or want to modify it can have everything they need).

The Apple // version of this program will be created with Apple /// Pascal using the {$SETC APPLE:=2} conditional compilation facility of the /// Pascal compiler (see page 109 of the Apple /// Pascal Prgorammer's Manual, volume 2 dated 1981). This allows a single source to exist for this program even though the same source will be used for two different target machines.

10.0 UTILITY PROGRAMS ON OTHER MACHINES

It may be useful to have some utility programs running on other machines (e.g. Macintosh) that deal with disk image files.

For example, there should exist a program that verifies text disk images files. Also, there may need to be a program that can convert a text disk image file to a binary disk image file for use by other programs that can deal with binary disk images (e.g. the various Apple // emulator programs seem to use binary images only). Likewise, there should be a utility that converts binary disk images to text files in case you want to send such a text image to my program to re-create a real 5.25" floppy disk.

These programs could all be the same program. I most likely will create such a utility for my Macintosh.

11.0 REFERENCES

Apple /// Owner's Guide
Apple Computer, 1981

Apple /// Plus Owner's Guide
Apple Computer, 1982

Apple /// Standard Device Drivers Manual
Apple Computer, 1981

Apple /// Pascal Programmer's Manual (volumes 1 and 2)
Apple Computer, 1981

Apple /// Pascal Technical Reference Manual
Apple Computer, 1983

Apple /// SOS Reference Manual (volumes 1 and 2)
Apple Computer, 1982

/// Bits: John Jeppson's Guided Tour of Highway ///
Softalk magazine, May 1983

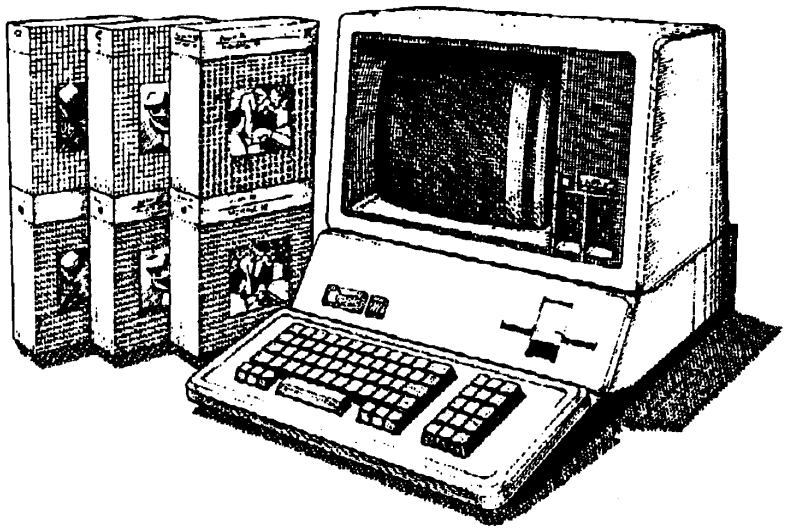
A Little SOS with your Pascal
Tim O'Konski, BYTE magazine, December 1982

Pascal source code to format SOS devices
James Vandermade, no date

Extraordinary Pi
Web site (www.users.globalnet.com.uk/~nickjh/pi.htm)
This is where I obtained the Pi digits.

F I N I S

Apple /// Computer Information



David Craig's Apple /// 5.25" Floppy Disk Image Maker Program Specification

Revision 5
18 March 1999 (matrix)

"_066.PICT" 166 KB 2001-08-25 dpi: 300h x 300v pix: 2472h x 3213v

APPLE /// INFO

Apple ///
5.25" 140K Floppy Disk
Image Maker Program Specification

Revision 5 -- David T. Craig -- 18 March 1999

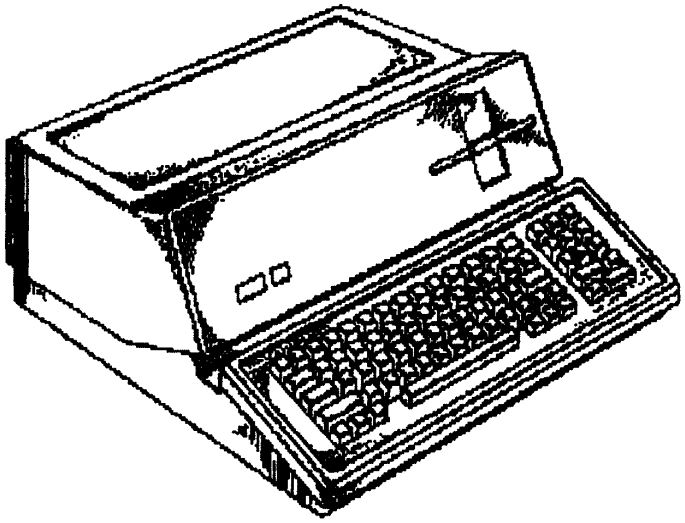


TABLE OF CONTENTS

"_067.PICT" 156 KB 2001-08-25 dpi: 300h x 300v pix: 2472h x 3213v

APPLE /// INFO

0.0 REVISIONS

Date	Rev	Comments
1999-02-24	0	Created the preliminary spec.
1999-02-25	1	Added table of contents, spelling corrections.
1999-03-08	2	Added line checksum detail.
1999-03-10	3	Added comments about storing disk images on a CD-ROM. Changed checksum algorithm to not produce large numbers. Added D)evices command to command line. Improved reference section.
1999-03-11	4	Added Apple /// picture to first page. Corrected the checksum examples and added more checksum info. Centered all section headings, enlarged font size.
1999-03-18	6	Added U)tilities command to main comman line. Added various disk utilities in their own command line. Added discussion about the Mac utility and image catalogs. Added image file header item SOURCE. Added image file header item DISK_IMAGE_NUMBER. Added image file header item HEADER_CHECKSUM. Added graphics for send and receive image operations. Used "(" instead of ")" for CLI since "(" is what Apple uses for its Apple /// and // Pascal CLIs.

1.0 AUTHOR

David T. Craig
941 Calle Mejia # 1006
Santa Fe, NM 87501 USA

home phone: (505) 820-0358
email: 71533.606@compuserve.com

2.0 AUDIENCE

Apple /// computer users who want to preserve their collections of Apple /// 5.25" 140K floppy disks.

Apple // computer users will also be interested in this program since it will support Apple // 140K floppy disks.

3.0 INTRODUCTION

This specification describes a proposed Apple /// program that will let users make disk image files from Apple /// 5.25" 140K floppy disks and also re-create these disks from a disk image file.

Apple // computer 140K disks can also be made into disk images (see section APPLE // DISK SUPPORT).

These disk image files can be stored on other machines for safe-keeping. For example, one could use a 600MB CD-ROM to store around 2,000 disk images files (at around 300K per disk image file).

This program also provides a decent suite of disk verification tools such as verifying that all the blocks of a disk are readable and writable. The purpose of these tools is to tell you if a disk media is correct or has problems that will interfere with sending or receiving disk images. Various verification tools exist that let you also format a disk and test the reading and writing of either a single disk or all the disks in various ways.

Comments about this specification are welcome.

4.0 SCHEDULE

This program should hopefully be finished in about 2 weeks. The reason for this deadline is I have borrowed some Apple /// disks that I need to return in around 2 weeks and I want to make disk image files from these disks before returning them.

I plan to implement this program first on the Apple /// since the ///'s development system, Apple /// Pascal, can produce Apple // Pascal compatible programs.

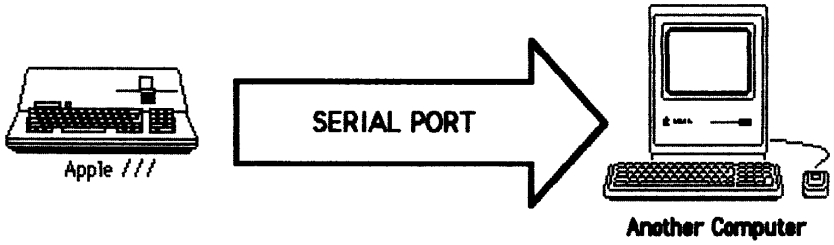
5.0 GOALS

The goals of this program are:

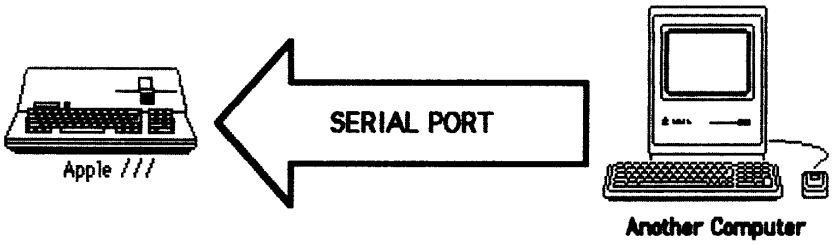
- Provide a simple to use Apple /// program that allows users to save 5.25" Apple /// floppy disks to external media in a reliable manner.
- Send disk data out the Apple /// computer's serial port to another computer which will save the disk image files as regular files on its disk media. This other computer would only need a terminal program that can receive files via its serial ports.
- Receive disk image files to an Apple /// computer via its serial port and recreate the original disk onto an Apple /// formatted or unformatted floppy disk.
- Image file sending and receiving will be on a floppy disk block basis. A disk block contains 512 bytes.
- Support Apple // 5.25" formatted disks also. See section APPLE // DISK SUPPORT at the end for more details.
- Copy protected disks will not be supported. If a disk's blocks cannot all be read then no disk image will be created by this program.

Graphically, one can view this program's goals as follows:

SENDING A DISK IMAGE FROM THE APPLE /// TO ANOTHER COMPUTER



RECEIVING A DISK IMAGE FROM ANOTHER COMPUTER TO THE APPLE ///



APPLE /// 5.25" 140K FLOPPY DISK IMAGE MAKER PROGRAM SPECIFICATION
Revision 5 -- David T. Craig -- 18 March 1999 -- 6 / 30

6.0 IMAGE FILE FORMAT

Disk image files have the following characteristics:

- Contain only human-readable characters (i.e. no control or Extended ASCII characters, except for the carriage return (ASCII 13) and line feed (ASCII 10) characters). Reason: So these files can easily be transferred between machines and over email networks without worrying about any special characters causing problems.

- Stored as text files with whatever machine operating system extension information is needed. For example, on PCs they would normally end in ".TXT", on Macs the internal document type should be "TEXT". The disk image files should follow a standard naming convention. See section UTILITY PROGRAMS ON OTHER MACHINES for recommended names.

- Formatted on a line basis with each line terminated with carriage return and line feed characters. Reason: So Microsoft Windows-based machines and UNIX-based machines can view the files with their word processors. Macintosh machines should also be able to view these files though there will appear a space before all but the first line of the image file.

- Blank lines may exist and are ignored.

- Divided into disk block groups with each group headed by a comment specifying the block number starting with block 0.

- Disk blocks are divided into 32 byte chunks for a total of 16 chunks. Each 32 byte chunk appears as 64 characters with a leading address value and with a following ASCII representation of the bytes. Each 32 byte chunk is also terminated with a checksum value that is computed from the 32 bytes.

- File starts with a header and ends with a footer comment. All comments are stored on a line basis with each comment line starting with the ";" character marker (comment marker must start at position 1 of the line). Most header comments are not required but the footer comment is required. The first line of the file must be "APPLE 5.25 INCH DISK IMAGE FILE". Each header command line contains an item and its data. The item names are fixed, the data can vary and the data should not exceed 80-100 characters in length. The header comment specifies the following information:

Disk contents	e.g. Apple Writer /// 1.1
Disk format	e.g. Apple /// SOS

Disk block count	e.g. 280
Source of the disk	e.g. WAP disk "Business BASIC - 03"
Disk image number	e.g. 1
Image creation date	e.g. 24 February 1999
Image program creator name	e.g. A3DIM 1.0.0
Person who created image	e.g. David T. Craig
Address of person	e.g. 123 Main, City, State Zip USA
Person Contact info	e.g. 71533.606@compuserve.com
Misc. comment	e.g. Really neat program
Header checksum	e.g. 123456

Here's information about some of the header items that may be somewhat obscure:

Source of the disk: This is a description of where this disk originated. For example, if you received this disk from a society such as the Washington Apple Pi (WAP) group then list it as the source. You should also list the source's catalog number (if any) for the disk. This allows someone to know where you got the disk.

Disk image number: This is a number that identifies the disk image file. You may also want to include your initials in front of this number so that it will be serialized to you. For example, I plan to set this number to "DTCn" such as "DTC001", "DTC002", etc. This numbering scheme allows different people to use the same numbers but their initials delimit each person's disks from the other person's disks. This number could also be used by a disk image cataloging program to create a disk image file name in case the disk image file name needs to be standardized (see UTILITY PROGRAMS ON OTHER MACHINES for more on this).

Header checksum: This is a checksum of the header tokens and their data, excludes this checksum field itself. This checksum should be calculated on a line basis using the same checksum algorithm as the disk block line checksum.

A sample header is:

```
; APPLE 5.25 INCH DISK IMAGE FILE
;
; DISK_NAME:           Apple Writer /// 1.1
; DISK_FORMAT:         Apple /// SOS
; DISK_BLOCKS:         280
; DISK_SOURCE:         WAP Business BASIC 5
; DISK_IMAGE_NUMBER:   DTC001
; DATE:                24 February 1999
; CREATED_BY:          A3DIM 1.0.0
```

```
; CONTACT_NAME:        David T. Craig
; CONTACT_ADDRESS:     941 Calle Mejia #1006, Santa Fe, NM 87501 USA
; CONTACT_EMAIL:       71533.606@compuserve.com
; COMMENT:             Really neat program
; HEADER_CHECKSUM:     12345
```

The spaces after the tokens (e.g. "DISK_NAME") are not important but having all the token information line up makes the files much more readable to humans.

The footer comment contains the following lines:

```
; DISK_CHECKSUM: nnnnnnnn
;
; FINIS
```

The "FINIS" line MUST be the last line in the image file otherwise the disk restoration will see the file as corrupt. The "DISK_CHECKSUM" line must also exist.

- Missing header or footer information will cause the image file to be seen as corrupted and will not be processed. Incorrect checksums indicate disk images are corrupt and should not be used.

- Image files will be verified on a line basis. Verification will test that only human readable characters exist (line feed at start of a line is OK), line address is correct, data section is correct (has correct length, no spaces, correct hex characters), ASCII characters are correct (length, characters), and the line checksum is correct for the whole line.

- Each disk block begins with the following comment (the $ number is the hexadecimal number for the block followed by a "/" then the block number in decimal):

```
; BLOCK: $0081/129
```

Each block line contains the following sections:

```
block(hex) address(hex): data(hex) [ascii] CHKSUM: checksum(dec)
```

```
block(hex)        - block number in hex
addreses(hex)     - 4 hex digits teminated with ":"
data(hex)         - 64 hex digits
[ascii]           - 32 characters bound by "[" and "]"
                  - control codes, spaces, DEL and extended are "."
checksum(dec)     - decimal checksum of whole line
```


"_075.PICT" 204 KB 2001-08-25 dpi: 300h x 300v pix: 2472h x 3213v

A sample block with a few lines is:

```
; BLOCK: $0000/0
;
0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 12345
0000 0020: ...
0000 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 12345
;
; BLOCK_CHECKSUM: 23475
```

Note the BLOCK_CHECKSUM. Every block must have a single checksum value, in addition to the individual line checksums. The block checksum is just a sum of all of the line checksums. For example, if a block contained 3 lines with the checksums 10, 20, and 30, then the block checksum would be 60 (10+20+30).

A sample line that could appear in an image file (and not one of the above made-up lines) follows:

```
0000 0000: 1236754ABF3428976EF678876CD67834
           0AAFF678234AECBBD897234879FE8796
           [ffdU YREHJ7 CJ.fdsjkhwre..23fw1%] CHKSUM: 22392
```

Note: I have broken this line into 3 lines so it would fit nicely in this document, it would appear as a single line in a real disk image file. This line also has a correct checksum value, 22392. See the next paragraph for checksum calculation deatails.

o The checksum is a decimal value that is calculated based on ALL of the characters in the line. The calculation is a repeated sum of all of the line character ASCII values multipled against a specific digit "key". The "key" here consists of the digits of the mathematical number Pi (3.14159265...). For example, if the line contains the following:

"DAVID"

then the checksum will be calculated as follows:

Pi to 255 digits:

3141592653 5897932384 6264338327 9502884197 1693993751
0582097494 4592307816 4062862089 9862803482 5342117067
9821480865 1328230664 7093844609 5505822317 2535940812
8481117450 2841027019 3852110555 9644622948 9549303819
6442881097 5665933446 1284756482 3378678316 5271201909

14564

ASCII "D" = 68 "3" = 3
ASCII "A" = 65 "1" = 1
ASCII "V" = 86 "4" = 4
ASCII "I" = 73 "1" = 1
ASCII "D" = 68 "5" = 5

Checksum = (68 x 3)+(65 x 1)+(86 x 4)+(73 x 1)+(68 x 5)
= 204 + 65 + 344 + 73 + 340
= 1026

Illustrative programming code in the 4th Dimension language by ACI of France for this checksum follows:

```
` 255 digits of the number Pi.  This means this routine can compute
` the checksum for strings with up to 255 characters.
`
` This string should be initialized at the beginning of a program
` so that it doesn't have to be initialized each time this routine
` is used therby making this routine run faster.

$pi :=

      "31415926535897932384626433832795028841971693993751058209749445" +
      "45923078164062862089986280348253421170679821480865132823066470" +
      "70938446095505822317253594081284811174502841027019385211055596" +
      "96446229489549303819644288109756659334461284756482337867831652" +
      "527120190914564"

Repeat
      $data := Request("Enter data to checksum:";"")

      If ($data # "")
            $checksum := 0

            For ($i;1;Length($data))
                  $data_ascii      := Ascii( Substring($data;$i;1) )
                  $pi_digit  := Ascii( Substring($pi;$i;1) ) - 48

                  $checksum  := $checksum + ($data_ascii * $pi_digit)
            End for

            ALERT("Checksum = " + String($checksum))
      End if
Until ($data="")
```

The checksum is calculated from all of the data in a single line. For example, the above sample disk block line would have a checksum calculated on the following data which includes the spaces and other non-disk block data:

"0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67]"

(Side Note) Why is this algorithm used? I have used the more traditional checksum algorithms such as exclusive-ORs and CRCs, but wanted something different for this program. This algorithm should prove sufficient and

besides that, I have an interest in the number Pi and wanted to use it in some real world situation.

Note: The header portion of the disk image file contains a CHECKSUM_VERSION line which tells programs that process these disk image files what checksum algorithm is used. This version information exists in case this algorithm changes in the future, e.g. to a more robust algorithm. The current checksum version is 1, the next will be 2, etc.

o The footer checksum is calculated based on the checksums of the individual lines. This checksum is just the sum of the weighted digit values of the digits in the line checksums. The weights here are the digits of the number pi. For example, if you have a file with 2 lines with the line checksums "12" and "3" then the footer checksum would be:

```
Footer Checksum  = (1 * 3) + (2 * 1) + (3 * 3)
                 = 3 + 2 + 9
                 = 14
```

Here's an ACI 4th Dimension program that calculates the footer checksum (this assumes $pi has been initialized):

```
Repeat
     $data := Request("Enter data to checksum:";"")

     If ($data # "")
          $checksum := 0

          For ($i;1;Length($data))
               $data_digit      := Ascii(Substring($data;$i;1))-48
               $pi_digit        := Ascii(Substring($pi;$i;1))-48

               $checksum := $checksum + ($data_digit * $pi_digit)
          End for

          ALERT("Checksum = "+String($checksum))
     End if
Until ($data="")
```

o Sample skeleton image file is:

```
; APPLE 5.25 INCH DISK IMAGE FILE
;
; DISK_NAME:            Apple Writer /// 1.1
; DISK_FORMAT:          Apple /// SOS
; DISK_BLOCKS:          280
; DISK_SOURCE:          WAP Business BASIC 5
; DISK_IMAGE_NUMBER:    DTC001
; DATE:                 24 February 1999
; CREATED_BY:           A3DIM 1.0.0
; CONTACT_NAME:  David T. Craig
; CONTACT_ADDRESS:      941 Calle Mejia #1006, Santa Fe, NM 87501 USA
; CONTACT_EMAIL:        71533.606@compuserve.com
; COMMENT:              Really neat program

; CHECKSUM_VERSION: 1

; BLOCK: $0000/0
;
0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 344785
0000 0020: ...
0000 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 56873
;
; BLOCK_CHECKSUM: 23475

...

; BLOCK: $0117/279
;
0117 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 437823
0117 0020: ...
0117 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 5623445
;
; BLOCK_CHECKSUM: 23475

; DISK_CHECKSUM: 57856
;
; FINIS
```

7.0 USER INTERFACE

The disk image program will be named A3DIM (Apple 3 Disk Image Maker). When the user runs this program it asks the user what to do, mainly either send a disk image out a serial port or receive a disk image from an external machine. The user interface should be a simple command-line inteface (CLI) with the following contents (Apple /// and // Pascal [and Apple Lisa Workshop] users or P-System users in general will be familiar with this simple but very functional command interface):

S(end-disk R(eceive-image D(evices U(tilities H(elp Q(uit ?

Disk drives are specified by numbers. For example, the number 1 means the first disk drive, number 2 the second disk drive, etc. For Apple /// users, drive 1 means .D1, drive 2 means .D2, etc. For Apple // users, drive 1 means unit 4, drive 2 unit 5, drive 3 unit 9, drive 4 unit 10. This is done so that this program has consistent disk drive references for both the Apple /// and the Apple // platforms. Apple /// users may be more used to drive references such as ".D1" but I think this standardized way is better.

Note: This program may also accept standard drive designations for each platform. But this feature is only an idea at this time and may not become real until this program has been used for real. This means that on the /// the user could type ".D1" and the program would accept this and use drive 1. On the // the user could type "#4" to mean drive 1 (I will need to check if this is the standard way of designating a drive in Apple // Pascal -- its been a L O N G time since I've delt with Apple // Pascal).

The program should provide as many defaults as possible to minimize user typing. For example, when the program asks for a disk drive the program should display a default drive specifier and if the user presses the Return key this default is used. Also, once an item is specified that item should retain that value when the value is prompted for again. For example, if the user enters 2 as the drive specifier, then this item should appear as the default for the next prompt that needs a drive specifier. Processing should show some type of indicator, for example a line of "*" characters when a disk's blocks are being sent.

Character case of inputs should be ignored (e.g. "x" = "X") and extra leading or trailing spaces should also be ignored (e.g. " david " = "david").

Here's a description of each command with sample command output:

S(end-disk

Purpose: Send a disk's image out a serial port. This command should ideally determine the number of blocks on a disk and use that instead of always using 280 blocks.

```
Disk Drive to Send Disk Image [1] ? 2
Disk Name [Apple Writer /// 1.1] ?
Disk Format [Apple /// SOS] ?
Disk Blocks [280] ?
Disk Block Range [0-279] ?
Date [24 February 1999] ?
Contact Name [David T. Craig] ?
Contact Address [941 Calle Mejia #1006, Santa Fe, NM 87501 USA] ?
Contact Email [71533.606@compuserve.com] ?
Sending 280 disk blocks to the serial port ...
Verifying Blocks: **************************
Blocks Sent:      ********************
```

Note: A block verification is done first on the disk to be sent to make certain it can be fully read before the sending is really done. This will detect media problems such as bad blocks and also copy-protected disks which this utility does not support. If any blocks cannot be read then the following message appears and the disk sending is not done:

```
Disk send failed because some blocks could not be read:
      34    78    156
```

If bad blocks are found you may want to verify the disk with the Verify or Test commands in this utility's Utilities section.

Note the "Disk Block Range" prompt. This prompt allows you to specify a range of disk blocks to send. The default is the whole disk block range, e.g. 0-279 for a 280 block disk. This can be very handy if you want to re-send a selection of already sent disk blocks for an image (the received disk image could have had some transmission errors for a few blocks and you don't want to resend the whole disk image again). If you use this feature you will need to manually merge the new disk image with an older image so you can can have a single complete disk image file.

R(eceive-image

Purpose: Receive a disk's image from a serial port and save to a floppy. If the disk image specifies a disk size that differs from the receiving disk then the reception should not take place.

```
Disk Drive to Receive Disk Image [2] ? 3
Disk in drive 3 is not formatted, formatting disk ...
Receiving disk image from serial port to 280 block disk ...
Blocks Received: ********************
```

Note: If the disk in the drive to receive is already formatted then the following prompt should appear (the default for such destructive actions should always be No):

```
WARNING -- Disk in drive 3 is already formatted.
Overwrite this disk (Y/N) [N] ?
```

D(evices

Purpose: Lists the devices connected to your computer. For the Apple /// the SOS device numbers, device unit numbers, and device names will appear (for block devices the number of blocks will also be listed). For the Apple // most likely only the disk device unit numbers will be listed. May also list if a disk is write-protected. May also list the Apple // disk block size and maybe even the disk type (e.g. DOS 3.3, Pascal, ProDOS).

U(tilities

Purpose: Display another command line that lets the user perform various disk-based utilities. The purpose of these utilities is to let the user perform various actions on their disk drives and disks. These utilities should provide the user with drive and disk reliability information. The command line is:

F(ormat V(erify E(xamine C(ompare D(uplicate T(est L(ist S(um R(eturn

Each utility command does the following:

F(ormat

Purpose: Formats a disk, no directory information is put on the disk.

```
Disk Drive to Format [2] ? 3
Formatting disk in drive 3 ...
```

Note: If the disk in the drive to receive the image is already formatted then the following prompt should appear (the default for such destructive actions should always be No):

```
WARNING -- Disk in drive 3 is already formatted.
Overwrite this disk (Y/N) [N] ?
```

Note: If the formatting fails it will try one more time. You may also want to try the formatting in another drive in case the drive you used for the first format is defective in some way (e.g. the drive's disk speed needs adjustment).

V(erify

Purpose: Verify a disk's blocks by reading all the blocks. If a block cannot be read then its block number will be listed after the "Blocks Verified" line. Verifies a single time, see the Test command for an extensive disk testing facility.

```
Disk Drive to Verify [2] ? 3
Verifying disk in drive 3 ...
Blocks Verified: ********************
```

If you type "?" in reply to the "Disk Drive to Verify" prompt then you are given the following verification options:

```
Verify How (Sequential/Random) [Sequential] ?
```

Typing "S" means you want to verify the disk blocks by accessing them in a sequential fashion starting at the first to the last block, then from the last block to the first. "R" means access the blocks in a random fashion with no block accessed more than once (a card shuffle algorithm will be used here to achieve this access strategy).

You will then be asked:

```
Read or Write Access (Read/Write/Both) [Read] ?
```

Typing "R" means you want to only read the disk blocks. "W" means you want to just write the disk blocks. "B" means you want to both read and write the disk blocks. The "W" option should only be used when you don't care what is on the disk since its contents will be destroyed.

After these special options are prompted for, you will be shown the "Disk Drive to Verify" prompt where you can enter the disk drive number to

verify. You can also enter at this prompt "A" (means "ALL") which means verify all the mounted disks. The program will then verify the first disk drive, then the second, etc. You can also enter "AR" (means "ALL RANDOM") here which means the program will randomly verify all of your disk drives simultaneously. For example, if you have 4 floppy disk drives (like I do on my ///) then it would access one block from one drive, then access another block on another drive, etc. This feature, when combined with the Both and Random options should really stress-test your disk system since all drives should tend to be on and the random accesses will cause all the drives to be working at the same time. If your drives or system have any problems then this test should hopefully find them.

If you want to test just a subset of your mounted drives you can also type just their drive numbers. For example, "1 3" (or "13") means test just drives 1 and 3.

Note: If a disk is write-protected, then the disk will not be written to even if you specify disk writing.

E(xamine

Purpose: Displays the contents of a disk's blocks on the screen. Useful if you want to see what is on one of your disks. Prompts are:

```
Disk Drive to Examine [2] ? 3
Block [0] ? 112
```

Then 32 lines of information appear on your screen in the following format:

```
000: 42789FB07A4EC8263F939ECCBB874380 [SDF.wr5^#35954f.]
```

16 lines appear first, then you are prompted with "Press RETURN or SPACE to continue" to show the last 16 lines of the block (this is needed since the screen shows only 24 lines and I want you to be able to see all of the block data without having to pause the screen in some archaic fashion).

The "000" at the start contains the byte offset of the first byte of the line. The middle group contains 16 bytes of data in hexadecimal. The last group contains the 16 bytes in ASCII with control characters and extended characters (including DEL) as a "." character.

C(ompare

Purpose: Compares 2 disks in 2 disk drives. Prompts are:

```
Disk Drive 1 to Compare [2] ? 3
Disk Drive 2 to Compare [1] ? 1
Comparing Blocks: ************************
```

If any blocks mismatch then they will be listed, e.g.:

```
Mismatched blocks ...
    1    5    27
```

Note: Comparison should be done on a track basis, or at least a track from each disk will be read but the comparison would be on a block basis within each track (a track contains 8 blocks for 4K total).

D(uplicate

Purpose: Duplicates one disk to another. Prompts are:

```
Source Disk Drive      [3] ? 2
Destination Disk Drive [1] ? 1
Duplicating Blocks: *********
```

Note: Duplication should be done on a track basis for faster speed.

T(est

Purpose: Tests all of the mounted disks by reading and writing them. This command is very similar to the Verify command but this command does not prompt you for any options and it works until you tell it to stop (just press the ESCAPE key). All of the tests that Verify supports are done. Prompts are:

```
Format disks also (Y/N) [N] ?
```

If you answer "Y" here then all of the mounted disks are formatted.

Sample output is (this sample assumes you have only 1 disk drive):

```
Testing drive 1:  Formatting ...
Testing drive 1:  Sequential Read
**********************************************
Testing drive 1:  Sequential Write
**********************************************
Testing drive 1:  Sequential Read and Write
**********************************************
Testing drive 1:  Random Read
**********************************************
Testing drive 1:  Random Write
**********************************************
Testing drive 1:  Readom Read and Write
**********************************************
```

When you quit this test a summary appears:

```
Summary:
      Passes ..................... 23
      Drives accessed ............ 3
      Blocks read ................ 12,567
      Blocks written ............. 8,752
      Block errors ............... 5
      Format failures ............ 2
      Drives with errors ......... 1 2
```

Note: Each pass of this test performs all of the tests including the formatting (if specified).

Note: If a format fails then the format will be done again.

L(ist

Purpose: Lists all the mounted disks.

This can be handy if you want to see if the computer knows if a disk is in a drive.

S(um

Purpose: Compute a checksum for a disk or for a specific block range. Prompt is:

```
Disk Drive to Checksum [2] ? 3
Disk Block Range [0-279] ?
```

Uses the same checksum as the disk image checksum if the whole disk is to be checksummed. Otherwise, uses the block checksum algorithm which is just a sum of the individual line checksums.

R(eturn

Purpose: Return to the main command line.

Note: One command that is not here and which would be a great command to have is a disk speed command. This is an important aspect of disk drives since slow or fast disk drives can produce disks that are unreadble by other computers. This command is not present here since I don't have access to programming code that does this in a simple fashion (I could pull this code out of the Apple /// Disk /// formatter driver source listing but that seems rather complicated to me). If anyone has a code library with such a routine that I can link to a Pascal program that would be very welcome.

H(elp

Purpose: Display a brief description of this program.

"This program's purpose is to yadda yadda yadda ..."

Q(uit

Purpose: Quits the program, but asks the user first.

Really Quit (Y/N) [Y] ?

When this program starts it will display information about itself such as the program name, version number, creation date, and author info. This information will be followed by the command line. For example,

```
Apple /// Disk Image File Sender and Receiver Utility
Version 1.0.0 -- 24 Feburary 1999
Written by David T. Craig
```

Future Features:

- Saving disk images to other attached drives such as a hard drive or a 3.5" 800K floppy. This would allow users to make disk images on their machine and later transmit them to another machine either directly via disk or via their own telecommunications program such as Access ///. I don't envision this feature being part of the first release but a definite possibility for version 2.

  This feature could exist for both the /// and the //.

- Saving non-140K disk drives such as the ProFile 5MB hard disk. This would allow users to dump the contents of these drives as a disk image. Recreating these images could be a problem that I need to check into further. The user could specify these devices via their device name such as .PROFILE. The user would need to know the number of blocks on these other devices.

  This feature would only exist on the /// and not on the //.

8.0 APPLE // DISK SUPPORT

This program should also be implemented so that it runs on the Apple // family of machines (e.g. Apple][, Apple][+, Apple //e, Apple //c, Apple //gs). The reason for this is the low-level disk format of the Apple /// 5.25" disks and the Apple // fmaily 5.25" disks are the same (the directory structures may differ but that is immaterial at the low-level block level).

This Apple // family version program should also fully support the sending and receiving of Apple /// disk image files.

Disk drive specifiers may be different on the // version of this program since the // family does not use .D1, .D2, etc for drives. I will need to check how disks are specified in the // world. Or, the /// drive specifiers could be used and the // program would translate them to what it really needs. See the above discussion for more about this.

This program should use the Apple // family serial I/O devices such as the Super Serial Card in the Apple //e or the built-in serial port of the Apple //c machine (a //GS machine could also be used if this has a serial port, but I'm not familiar with this machine so I can't say more at this time).

This program will most likely not support automatic disk formatting on an Apple //-class machine when a disk image is received from an external machine. I believe the Apple // family OS does not support a formatting driver such as the /// supports. I know you can get the formatter for Apple // Pascal use but I don't have it and don't want to bother at this time with this since the user can just pre-format a floppy for disk creation use using a program such as Apple's ProDOS utility.

9.0 PROGRAMMING DETAILS

The program should be programmed in a high-level language and not assembly language. Why? I'm too old to deal with assembly any more and assembly is just not needed. I plan on using Apple /// Pascal for the /// implementation. I will most likely create the /// version first and then use the ///'s sources for the // implementation. The source should have conditional compilation statements governing the destination platform so that only a single instance of the source code need to exist. For example, the first line could be:

```
{$SETC APPLE:=2}  { /// system use 3, // use 2 }
```

This is a built-in Apple /// Pascal conditional that can either be 2 or 3 (The default is 2). When set to 2 the /// compiler automatically generates Apple // Pascal codefiles. Any areas of the program that are // or /// specific (e.g. disk formatting) would be bounded by the APPLE conditional compilation test and do what is needed for that area.

Disk I/O will be achieved using the Pascal intrinsic commands UNITREAD and UNITWRITE to read and write 512 byte disk blocks respectively. Special disk operations such as disk formatting will be done using Pascal's UNITSTATUS command (Apple /// SOS calls may also need to be made to handle such tasks as getting a SOS device's Pascal unit number whcih the UNITREAD/WRITE commands require). Disk reading should be done eithe ron a block basis or a track basis (a track contains 8 blocks and is 4K in size). The reason for the track access is so the UNITREAD call can access disks very quickly. For the send-disk feature of this program the track reading method may not provide any benefits since the serial I/O will most likely be the bottleneck. But for the program's disk access utility features such as verifying a disk then the track reading may make this feature very fast (I envision being able to read one track per second since that is what one Apple // program can do -- means a disk should be verifiable in around 35 seconds since a 140K disk contains 35 tracks).

Serial I/O will be achieved using the Pascal commands WRITELN and READLN output and input serial port data. The serial ports will be accessed using Pascal's device names REMIN: and REMOUT: (from an Apple // perspective) or the .RS232 device (from an Apple /// perspective).

Serial I/O configuring will not be done by this program. The user will need to make certain that their /// (or //) serial I/O are configured correctly for the remote machine they are using. The /// distribution disk for this program will come with the program and the necessary OS files so that the user can just boot this disk and this program will work with pre-configured serial I/O (e.g. 9600

baud, XON-XOFF flow control, ...). For the Apple // world users will most likely need to setup their serial I/O card's DIP switches (ugh!). The distribution disk should also come with this specification and the source code for both the /// and // programs (I believe in making all of this program public so others with an interest in how it works or want to modify it can have everything they need).

The Apple // version of this program will be created with Apple /// Pascal using the {$SETC APPLE:=2} conditional compilation facility of the /// Pascal compiler (see page 109 of the Apple /// Pascal Prgorammer's Manual, volume 2 dated 1981). This allows a single source to exist for this program even though the same source will be used for two different target machines.

10.0 UTILITY PROGRAMS ON OTHER MACHINES

It may be useful to have some utility programs running on other machines (e.g. Macintosh) that deal with disk image files.

For example, there should exist a program that verifies text disk images files. Also, there may need to be a program that can convert a text disk image file to a binary disk image file for use by other programs that can deal with binary disk images (e.g. the various Apple // emulator programs seem to use binary images only). Likewise, there should be a utility that converts binary disk images to text files in case you want to send such a text image to my program to re-create a real 5.25" floppy disk.

The utility program should also provide a cataloging capability. I envision this reading the header of a disk image file and generating a text file with fields for each header line. This text file could be just an exact reproduction of the header information, or it could be a tab-delimited text file that a program such as a database or a spreadsheet could use and print in a nice fashion.

The utility program should also provide a disk image file renaming facility. I think this would be handy so that if the disk image files are not named in a standard fashion this utility would standardize the names. For example, I think the disk image file names should be "Apple3DiskImage_iiinnn.TXT" where "iiinnn" is the disk image number preceded by a person's initials. This information would originate in the disk image file header and the utility would extract this info to make the disk image file's new name. For example, if a disk image file has a disk image number of "DTC001" then the disk image file would be renamed to "Apple3DiskImageFile_DTC001.TXT". This name format is compatible with the Apple Macintosh and MS Windows machines. If the disk images will be stored on a machine with lesser naming cababilities, then a shorter naming standard could be used. For example, under some UNIX systems the names can be no longer than 14 characters. I recommend a naming standard of "A3D_iiinnn.TXT" for UNIX. For MS-DOS systems with the standard 8.3 naming convention I recommend "A3iiinnn.TXT". I think the utility program should ask the user for the disk name prefix and just append the disk image number and ".TXT" to the disk name. The user would then have the control here (the default prefix should be "Apple3DiskImageFile_").

One feature of this utility program that may be very useful is to write the files from a disk image file to native files on the host system. This would allow someone to easily obtain the files from a SOS (or Apple // Pascal) disk without having to run some type of special file transfer program). For example, the utility would know the directory and file formats of the disk and create corresponding files on the host machine. To make this simple I recommend not

recreating the disk image's file directories but instead just write the files out with a leading sequence number. For example, if a disk image has the directory and file organization:

```
Disk
     Directory A
          File A1
          File A2
     Directory B
          File B1
          File B2
          Directory B2
               File B21
```

The utility would create 5 files:

0001_A1 0002_A2 0003_B1 0004_B2 0005_B21

Note that the actual file names will be present as suffixes on the host machine files. Therefore, if a disk image file was named "SARA.HIST.TEXT" then the host file name would be "0001_SARA.HIST.TEXT".

I recommend that at least Apple /// SOS disk formats be recognized. It may also be useful to recognize Apple // Pascal disk formats since these are easy to parse (no subsdirectories exist and the file block allocation is very simple compared to SOS's multiple directory format with discontigious file block allocation).

I also recommend that SOS text files be converted to a more regular form when saved to a host machine. By this I mean remove the text file header (2 blocks) and save each file page (2 blocks) without the special text formatting stuff that is a remnant of the old UCSD P-System days.

I also think it would be neat if this utility program also generated a report for a disk image detailing the gory file system details of the disk. For example, list how many directories are on the disk, the characterists of each directory and file, and the blocks where each file resides. This could give people (such as this utility's author) a very good insight into the organization of a SOS disk. This should also support UCSD Pascal disks that the Apple // Pascal system creates.

The user interface for this utility program should also be a single command line. This makes using this program and if implemented on various machines this interface would be easier for different people to use since it would be standard across platforms.

These programs could all be the same program. I most likely will create such a utility for my Macintosh.

11.0 REFERENCES

Apple /// Owner's Guide
Apple Computer, 1981

Apple /// Plus Owner's Guide
Apple Computer, 1982

Apple /// Standard Device Drivers Manual
Apple Computer, 1981

Apple /// Pascal Programmer's Manual (volumes 1 and 2)
Apple Computer, 1981

Apple /// Pascal Technical Reference Manual
Apple Computer, 1983

Apple /// SOS Reference Manual (volumes 1 and 2)
Apple Computer, 1982

/// Bits: John Jeppson's Guided Tour of Highway ///
Softalk magazine, May 1983

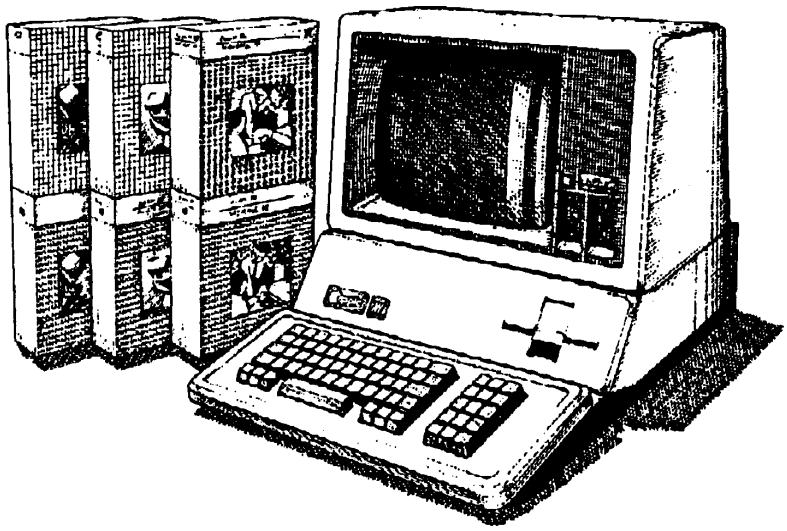
A Little SOS with your Pascal
Tim O'Konski, BYTE magazine, December 1982

Pascal source code to format SOS devices
James Vandermade, no date

Extraordinary Pi
Web site (www.users.globalnet.com.uk/~nickjh/pi.htm)
This is where I obtained the Pi digits.

F I N I S

Apple /// Computer Information



David Craig's Apple /// 5.25" Floppy Disk Image Maker Program Specification

Revision 5
18 March 1999 (laser)

APPLE /// INFO

Apple ///
5.25″ 140K Floppy Disk
Image Maker Program Specification

Revision 5 -- David T. Craig -- 18 March 1999

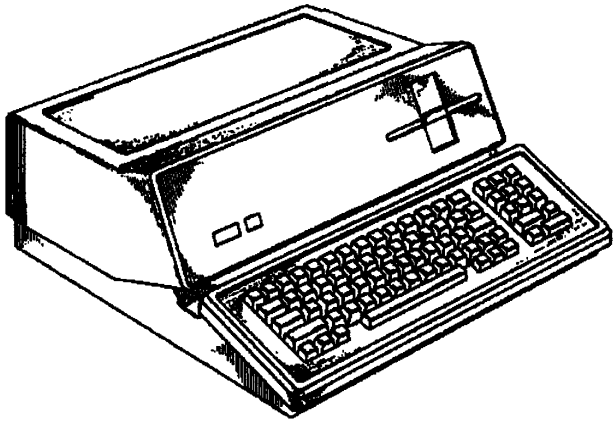


TABLE OF CONTENTS

"_098.PICT" 123 KB 2001-08-25 dpi: 300h x 300v pix: 2472h x 3213v

APPLE /// INFO

0.0 REVISIONS

Date	Rev	Comments
1999-02-24	0	Created the preliminary spec.
1999-02-25	1	Added table of contents, spelling corrections.
1999-03-08	2	Added line checksum detail.
1999-03-10	3	Added comments about storing disk images on a CD-ROM. Changed checksum algorithm to not produce large numbers. Added D)evices command to command line. Improved reference section.
1999-03-11	4	Added Apple /// picture to first page. Corrected the checksum examples and added more checksum info. Centered all section headings, enlarged font size.
1999-03-18	6	Added U)tilities command to main comman line. Added various disk utilities in their own command line. Added discussion about the Mac utility and image catalogs. Added image file header item SOURCE. Added image file header item DISK_IMAGE_NUMBER. Added image file header item HEADER_CHECKSUM. Added graphics for send and receive image operations. Used "(" instead of ")" for CLI since "(" is what Apple uses for its Apple /// and // Pascal CLIs.

1.0 AUTHOR

David T. Craig
941 Calle Mejia # 1006
Santa Fe, NM 87501 USA

home phone: (505) 820-0358
email: 71533.606@compuserve.com

2.0 AUDIENCE

Apple /// computer users who want to preserve their collections of Apple /// 5.25" 140K floppy disks.

Apple // computer users will also be interested in this program since it will support Apple // 140K floppy disks.

APPLE /// INFO

3.0 INTRODUCTION

This specification describes a proposed Apple /// program that will let users make disk image files from Apple /// 5.25" 140K floppy disks and also re-create these disks from a disk image file.

Apple // computer 140K disks can also be made into disk images (see section APPLE // DISK SUPPORT).

These disk image files can be stored on other machines for safe-keeping. For example, one could use a 600MB CD-ROM to store around 2,000 disk images files (at around 300K per disk image file).

This program also provides a decent suite of disk verification tools such as verifying that all the blocks of a disk are readable and writable. The purpose of these tools is to tell you if a disk media is correct or has problems that will interfere with sending or receiving disk images. Various verification tools exist that let you also format a disk and test the reading and writing of either a single disk or all the disks in various ways.

Comments about this specification are welcome.

4.0 SCHEDULE

This program should hopefully be finished in about 2 weeks. The reason for this deadline is I have borrowed some Apple /// disks that I need to return in around 2 weeks and I want to make disk image files from these disks before returning them.

I plan to implement this program first on the Apple /// since the ///'s development system, Apple /// Pascal, can produce Apple // Pascal compatible programs.

APPLE /// INFO

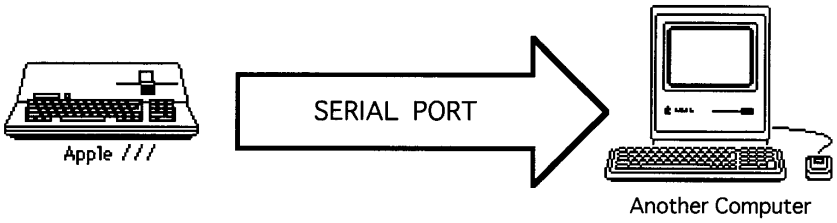
5.0 GOALS

The goals of this program are:

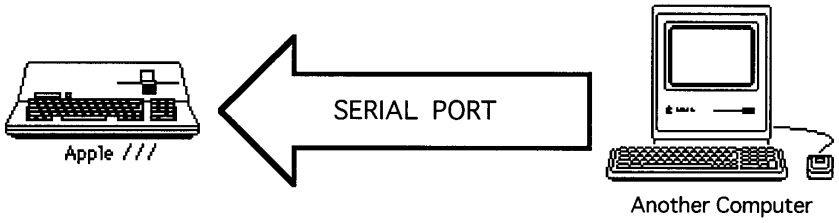
- Provide a simple to use Apple /// program that allows users to save 5.25" Apple /// floppy disks to external media in a reliable manner.
- Send disk data out the Apple /// computer's serial port to another computer which will save the disk image files as regular files on its disk media. This other computer would only need a terminal program that can receive files via its serial ports.
- Receive disk image files to an Apple /// computer via its serial port and recreate the original disk onto an Apple /// formatted or unformatted floppy disk.
- Image file sending and receiving will be on a floppy disk block basis. A disk block contains 512 bytes.
- Support Apple // 5.25" formatted disks also. See section APPLE // DISK SUPPORT at the end for more details.
- Copy protected disks will not be supported. If a disk's blocks cannot all be read then no disk image will be created by this program.

Graphically, one can view this program's goals as follows:

SENDING A DISK IMAGE FROM THE APPLE /// TO ANOTHER COMPUTER



RECEIVING A DISK IMAGE FROM ANOTHER COMPUTER TO THE APPLE ///



APPLE /// 5.25" 140K FLOPPY DISK IMAGE MAKER PROGRAM SPECIFICATION
Revision 5 -- David T. Craig -- 18 March 1999 -- 4 / 23

"_101.PICT" 155 KB 2001-08-25 dpi: 300h x 300v pix: 2472h x 3213v

APPLE /// INFO

6.0 IMAGE FILE FORMAT

Disk image files have the following characteristics:

- Contain only human-readable characters (i.e. no control or Extended ASCII characters, except for the carriage return (ASCII 13) and line feed (ASCII 10) characters). Reason: So these files can easily be transferred between machines and over email networks without worrying about any special characters causing problems.

- Stored as text files with whatever machine operating system extension information is needed. For example, on PCs they would normally end in ".TXT", on Macs the internal document type should be "TEXT". The disk image files should follow a standard naming convention. See section UTILITY PROGRAMS ON OTHER MACHINES for recommended names.

- Formatted on a line basis with each line terminated with carriage return and line feed characters. Reason: So Microsoft Windows-based machines and UNIX-based machines can view the files with their word processors. Macintosh machines should also be able to view these files though there will appear a space before all but the first line of the image file.

- Blank lines may exist and are ignored.

- Divided into disk block groups with each group headed by a comment specifying the block number starting with block 0.

- Disk blocks are divided into 32 byte chunks for a total of 16 chunks. Each 32 byte chunk appears as 64 characters with a leading address value and with a following ASCII representation of the bytes. Each 32 byte chunk is also terminated with a checksum value that is computed from the 32 bytes.

- File starts with a header and ends with a footer comment. All comments are stored on a line basis with each comment line starting with the ";" character marker (comment marker must start at position 1 of the line). Most header comments are not required but the footer comment is required. The first line of the file must be "APPLE 5.25 INCH DISK IMAGE FILE". Each header command line contains an item and its data. The item names are fixed, the data can vary and the data should not exceed 80-100 characters in length. The header comment specifies the following information:

Disk contents	e.g. Apple Writer /// 1.1
Disk format	e.g. Apple /// SOS
Disk block count	e.g. 280
Source of the disk	e.g. WAP disk "Business BASIC - 03"
Disk image number	e.g. 1
Image creation date	e.g. 24 February 1999
Image program creator name	e.g. A3DIM 1.0.0
Person who created image	e.g. David T. Craig
Address of person	e.g. 123 Main, City, State Zip USA
Person Contact info	e.g. 71533.606@compuserve.com
Misc. comment	e.g. Really neat program
Header checksum	e.g. 123456

Here's information about some of the header items that may be somewhat obscure:

Source of the disk: This is a description of where this disk originated. For example, if you received this disk from a society such as the Washington Apple Pi (WAP) group then list it as the source. You should also list the source's catalog number (if any) for the disk. This allows someone to know where you got the disk.

Disk image number: This is a number that identifies the disk image file. You may also want to include your initials in front of this number so that it will be serialized to you. For example, I plan to set this number to "DTCn" such as "DTC001", "DTC002", etc. This numbering scheme allows different people to use the same numbers but their initials delimit each person's disks from the other person's disks. This number could also be used by a disk image cataloging program to create a disk image file name in case the disk image file name needs to be standardized (see UTILITY PROGRAMS ON OTHER MACHINES for more on this).

Header checksum: This is a checksum of the header tokens and their data, excludes this checksum field itself. This checksum should be calculated on a line basis using the same checksum algorithm as the disk block line checksum.

A sample header is:

```
; APPLE 5.25 INCH DISK IMAGE FILE
;
; DISK_NAME:              Apple Writer /// 1.1
; DISK_FORMAT:            Apple /// SOS
; DISK_BLOCKS:            280
; DISK_SOURCE:            WAP Business BASIC 5
; DISK_IMAGE_NUMBER:      DTC001
; DATE:                   24 February 1999
; CREATED_BY:             A3DIM 1.0.0
; CONTACT_NAME:           David T. Craig
; CONTACT_ADDRESS:        941 Calle Mejia #1006, Santa Fe, NM 87501 USA
; CONTACT_EMAIL:          71533.606@compuserve.com
; COMMENT:                Really neat program
; HEADER_CHECKSUM:        12345
```

The spaces after the tokens (e.g. "DISK_NAME") are not important but having all the token information line up makes the files much more readable to humans.

The footer comment contains the following lines:

```
; DISK_CHECKSUM: nnnnnnnn
;
; FINIS
```

The "FINIS" line MUST be the last line in the image file otherwise the disk restoration will see the file as corrupt. The "DISK_CHECKSUM" line must also exist.

- Missing header or footer information will cause the image file to be seen as corrupted and will not be processed. Incorrect checksums indicate disk images are corrupt and should not be used.

- Image files will be verified on a line basis. Verification will test that only human readable characters exist (line feed at start of a line is OK), line address is correct, data section is correct (has correct length, no

spaces, correct hex characters), ASCII characters are correct (length, characters), and the line checksum is correct for the whole line.

o Each disk block begins with the following comment (the $ number is the hexadecimal number for the block followed by a "/" then the block number in decimal):

```
; BLOCK: $0081/129
```

Each block line contains the following sections:

```
block(hex) address(hex): data(hex) [ascii] CHKSUM: checksum(dec)

block(hex)      - block number in hex
addreses(hex)   - 4 hex digits teminated with ":"
data(hex)       - 64 hex digits
[ascii]         - 32 characters bound by "[" and "]"
                - control codes, spaces, DEL and extended are "."
checksum(dec)   - decimal checksum of whole line
```

A sample block with a few lines is:

```
; BLOCK: $0000/0
;
0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 12345
0000 0020: ...
0000 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 12345
;
; BLOCK_CHECKSUM: 23475
```

Note the BLOCK_CHECKSUM. Every block must have a single checksum value, in addition to the individual line checksums. The block checksum is just a sum of all of the line checksums. For example, if a block contained 3 lines with the checksums 10, 20, and 30, then the block checksum would be 60 (10+20+30).

A sample line that could appear in an image file (and not one of the above made-up lines) follows:

```
0000 0000: 1236754ABF3428976EF678876CD67834
             0AAFF678234AECBBD897234879FE8796
             [ffdU YREHJ7 CJ.fdsjkhwre..23fw1%] CHKSUM: 22392
```

Note: I have broken this line into 3 lines so it would fit nicely in this document, it would appear as a single line in a real disk image file. This line also has a correct checksum value, 22392. See the next paragraph for checksum calculation deatails.

o The checksum is a decimal value that is calculated based on ALL of the characters in the line. The calculation is a repeated sum of all of the line character ASCII values multipled against a specific digit "key". The "key" here consists of the digits of the mathematical number Pi (3.14159265...). For example, if the line contains the following:

"DAVID"

then the checksum will be calculated as follows:

Pi to 255 digits:

3141592653 5897932384 6264338327 9502884197 1693993751
0582097494 4592307816 4062862089 9862803482 5342117067
9821480865 1328230664 7093844609 5505822317 2535940812
8481117450 2841027019 3852110555 9644622948 9549303819
6442881097 5665933446 1284756482 3378678316 5271201909
14564

ASCII "D" = 68 "3" = 3
ASCII "A" = 65 "1" = 1
ASCII "V" = 86 "4" = 4
ASCII "I" = 73 "1" = 1
ASCII "D" = 68 "5" = 5

Checksum = (68 x 3)+(65 x 1)+(86 x 4)+(73 x 1)+(68 x 5)
= 204 + 65 + 344 + 73 + 340
= 1026

Illustrative programming code in the 4th Dimension language by ACI of France for this checksum follows:

```
` 255 digits of the number Pi.  This means this routine can compute
` the checksum for strings with up to 255 characters.
`
` This string should be initialized at the beginning of a program
` so that it doesn't have to be initialized each time this routine
` is used therby making this routine run faster.

$pi :=

     "31415926535897932384626433832795028841971693993751058209749 4"  +
     "45923078164062862089986280348253421170679821480865132823066 4"  +
     "70938446095505822317253594081284811174502841027019385211055 5"  +
     "96446229489549303819644288109756659334461284756482337867831 6"  +
     "527120190914564"

Repeat
     $data := Request("Enter data to checksum:";"")

     If  ($data # "")
          $checksum := 0

          For  ($i;1;Length($data))
               $data_ascii     := Ascii( Substring($data;$i;1) )
               $pi_digit       := Ascii( Substring($pi;$i;1) ) - 48

               $checksum       := $checksum + ($data_ascii * $pi_digit)
          End for

          ALERT("Checksum = " + String($checksum))
     End if
Until  ($data="")
```

The checksum is calculated from all of the data in a single line. For example, the above sample disk block line would have a checksum calculated on the following data which includes the spaces and other non-disk block data:

"0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67]"

(Side Note) Why is this algorithm used? I have used the more traditional checksum algorithms such as exclusive-ORs and CRCs, but wanted something different for this program. This algorithm should prove sufficient and besides that, I have an interest in the number Pi and wanted to use it in some real world situation.

Note: The header portion of the disk image file contains a CHECKSUM_VERSION line which tells programs that process these disk image files what checksum algorithm is used. This version information exists in case this algorithm changes in the future, e.g. to a more robust algorithm. The current checksum version is 1, the next will be 2, etc.

o The footer checksum is calculated based on the checksums of the individual lines. This checksum is just the sum of the weighted digit values of the digits in the line checksums. The weights here are the digits of the number pi. For example, if you have a file with 2 lines with the line checksums "12" and "3" then the footer checksum would be:

```
     Footer Checksum      = (1 x 3) + (2 x 1) + (3 x 3)
                          = 3 + 2 + 9
                          = 14
```

Here's an ACI 4th Dimension program that calculates the footer checksum (this assumes $pi has been initialized):

```
Repeat
     $data := Request("Enter data to checksum:";"")

     If ($data # "")
          $checksum := 0

          For ($i;1;Length($data))
               $data_digit    := Ascii(Substring($data;$i;1)) - 48
               $pi_digit      := Ascii(Substring($pi;$i;1)) - 48
               $checksum      := $checksum + ($data_digit * $pi_digit)
          End for

          ALERT("Checksum = "+String($checksum))
     End if
Until ($data="")
```

- Sample skeleton image file is:

```
; APPLE 5.25 INCH DISK IMAGE FILE
;
; DISK_NAME:              Apple Writer /// 1.1
; DISK_FORMAT:            Apple /// SOS
; DISK_BLOCKS:            280
; DISK_SOURCE:            WAP Business BASIC 5
; DISK_IMAGE_NUMBER:      DTC001
; DATE:                   24 February 1999
; CREATED_BY:             A3DIM 1.0.0
; CONTACT_NAME:           David T. Craig
; CONTACT_ADDRESS:        941 Calle Mejia #1006, Santa Fe, NM 87501 USA
; CONTACT_EMAIL:          71533.606@compuserve.com
; COMMENT:                Really neat program

; CHECKSUM_VERSION: 1

; BLOCK: $0000/0
;
0000 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 344785
0000 0020: ...
0000 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 56873
;
; BLOCK_CHECKSUM: 23475

...

; BLOCK: $0117/279
;
0117 0000: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 437823
0117 0020: ...
0117 01E0: 1212 ... EF [.dsfijh. ... *^e8.67] CHKSUM: 5623445
;
; BLOCK_CHECKSUM: 23475

; DISK_CHECKSUM: 57856
;
; FINIS
```

APPLE /// INFO

7.0 USER INTERFACE

The disk image program will be named A3DIM (Apple 3 Disk Image Maker). When the user runs this program it asks the user what to do, mainly either send a disk image out a serial port or receive a disk image from an external machine. The user interface should be a simple command-line inteface (CLI) with the following contents (Apple /// and // Pascal [and Apple Lisa Workshop] users or P-System users in general will be familiar with this simple but very functional command interface):

S(end-disk R(eceive-image D(evices U(tilities H(elp Q(uit ?

Disk drives are specified by numbers. For example, the number 1 means the first disk drive, number 2 the second disk drive, etc. For Apple /// users, drive 1 means .D1, drive 2 means .D2, etc. For Apple // users, drive 1 means unit 4, drive 2 unit 5, drive 3 unit 9, drive 4 unit 10. This is done so that this program has consistent disk drive references for both the Apple /// and the Apple // platforms. Apple /// users may be more used to drive references such as ".D1" but I think this standardized way is better.

Note: This program may also accept standard drive designations for each platform. But this feature is only an idea at this time and may not become real until this program has been used for real. This means that on the /// the user could type ".D1" and the program would accept this and use drive 1. On the // the user could type "#4" to mean drive 1 (I will need to check if this is the standard way of designating a drive in Apple // Pascal -- its been a L O N G time since I've delt with Apple // Pascal).

The program should provide as many defaults as possible to minimize user typing. For example, when the program asks for a disk drive the program should display a default drive specifier and if the user presses the Return key this default is used. Also, once an item is specified that item should retain that value when the value is prompted for again. For example, if the user enters 2 as the drive specifier, then this item should appear as the default for the next prompt that needs a drive specifier. Processing should show some type of indicator, for example a line of "*" characters when a disk's blocks are being sent.

Character case of inputs should be ignored (e.g. "x" = "X") and extra leading or trailing spaces should also be ignored (e.g. " david " = "david").

Here's a description of each command with sample command output:

S(end-disk

Purpose: Send a disk's image out a serial port. This command should ideally determine the number of blocks on a disk and use that instead of always using 280 blocks.

```
Disk Drive to Send Disk Image [1] ? 2
Disk Name [Apple Writer /// 1.1] ?
Disk Format [Apple /// SOS] ?
Disk Blocks [280] ?
Disk Block Range [0-279] ?
Date [24 February 1999] ?
Contact Name [David T. Craig] ?
Contact Address [941 Calle Mejia #1006, Santa Fe, NM 87501 USA] ?
Contact Email [71533.606@compuserve.com] ?
```

```
Sending 280 disk blocks to the serial port ...
Verifying Blocks: ************************
Blocks Sent:      ********************
```

Note: A block verification is done first on the disk to be sent to make certain it can be fully read before the sending is really done. This will detect media problems such as bad blocks and also copy-protected disks which this utility does not support. If any blocks cannot be read then the following message appears and the disk sending is not done:

```
Disk send failed because some blocks could not be read:
     34   78   156
```

If bad blocks are found you may want to verify the disk with the Verify or Test commands in this utility's Utilities section.

Note the "Disk Block Range" prompt. This prompt allows you to specify a range of disk blocks to send. The default is the whole disk block range, e.g. 0-279 for a 280 block disk. This can be very handy if you want to re-send a selection of already sent disk blocks for an image (the received disk image could have had some transmission errors for a few blocks and you don't want to resend the whole disk image again). If you use this feature you will need to manually merge the new disk image with an older image so you can can have a single complete disk image file.

R(eceive-image

Purpose: Receive a disk's image from a serial port and save to a floppy. If the disk image specifies a disk size that differs from the receiving disk then the reception should not take place.

```
Disk Drive to Receive Disk Image [2] ? 3
Disk in drive 3 is not formatted, formatting disk ...
Receiving disk image from serial port to 280 block disk ...
Blocks Received: ********************
```

Note: If the disk in the drive to receive is already formatted then the following prompt should appear (the default for such destructive actions should always be No):

```
WARNING -- Disk in drive 3 is already formatted.
Overwrite this disk (Y/N) [N] ?
```

D(evices

Purpose: Lists the devices connected to your computer. For the Apple /// the SOS device numbers, device unit numbers, and device names will appear (for block devices the number of blocks will also be listed). For the Apple // most likely only the disk device unit numbers will be listed. May also list if a disk is write-protected. May also list the Apple // disk block size and maybe even the disk type (e.g. DOS 3.3, Pascal, ProDOS).

U(tilities

Purpose: Display another command line that lets the user perform various disk-based utilities. The purpose of these utilities is to let the user perform various actions on their disk drives and disks. These utilities should provide the user with drive and disk reliability information. The command line is:

F(ormat V(erify E(xamine C(ompare D(uplicate T(est L(ist S(um R(eturn

Each utility command does the following:

F(ormat

Purpose: Formats a disk, no directory information is put on the disk.

```
Disk Drive to Format [2] ? 3
Formatting disk in drive 3 ...
```

Note: If the disk in the drive to receive the image is already formatted then the following prompt should appear (the default for such destructive actions should always be No):

```
WARNING -- Disk in drive 3 is already formatted.
Overwrite this disk (Y/N) [N] ?
```

Note: If the formatting fails it will try one more time. You may also want to try the formatting in another drive in case the drive you used for the first format is defective in some way (e.g. the drive's disk speed needs adjustment).

V(erify

Purpose: Verify a disk's blocks by reading all the blocks. If a block cannot be read then its block number will be listed after the "Blocks Verified" line. Verifies a single time, see the Test command for an extensive disk testing facility.

```
Disk Drive to Verify [2] ? 3
Verifying disk in drive 3 ...
Blocks Verified: ******************
```

If you type "?" in reply to the "Disk Drive to Verify" prompt then you are given the following verification options:

```
Verify How (Sequential/Random) [Sequential] ?
```

Typing "S" means you want to verify the disk blocks by accessing them in a sequential fashion starting at the first to the last block, then from the last block to the first. "R" means access the blocks in a random fashion with no block accessed more than once (a card shuffle algorithm will be used here to achieve this access strategy).

You will then be asked:

```
Read or Write Access (Read/Write/Both) [Read] ?
```

Typing "R" means you want to only read the disk blocks. "W" means you want to just write the disk blocks. "B" means you want to both read and write

the disk blocks. The "W" option should only be used when you don't care what is on the disk since its contents will be destroyed.

After these special options are prompted for, you will be shown the "Disk Drive to Verify" prompt where you can enter the disk drive number to verify. You can also enter at this prompt "A" (means "ALL") which means verify all the mounted disks. The program will then verify the first disk drive, then the second, etc. You can also enter "AR" (means "ALL RANDOM") here which means the program will randomly verify all of your disk drives simultaneously. For example, if you have 4 floppy disk drives (like I do on my ///) then it would access one block from one drive, then access another block on another drive, etc. This feature, when combined with the Both and Random options should really stress-test your disk system since all drives should tend to be on and the random accesses will cause all the drives to be working at the same time. If your drives or system have any problems then this test should hopefully find them.

If you want to test just a subset of your mounted drives you can also type just their drive numbers. For example, "1 3" (or "13") means test just drives 1 and 3.

Note: If a disk is write-protected, then the disk will not be written to even if you specify disk writing.

E(xamine

Purpose: Displays the contents of a disk's blocks on the screen. Useful if you want to see what is on one of your disks. Prompts are:

```
Disk Drive to Examine [2] ? 3
Block [0] ? 112
```

Then 32 lines of information appear on your screen in the following format:

```
000: 42789FB07A4EC8263F939ECCBB874380 [SDF.wr5^#35954f.]
```

16 lines appear first, then you are prompted with "Press RETURN or SPACE to continue" to show the last 16 lines of the block (this is needed since the screen shows only 24 lines and I want you to be able to see all of the block data without having to pause the screen in some archaic fashion).

The "000" at the start contains the byte offset of the first byte of the line. The middle group contains 16 bytes of data in hexadecimal. The last group contains the 16 bytes in ASCII with control characters and extended characters (including DEL) as a "." character.

C(ompare

Purpose: Compares 2 disks in 2 disk drives. Prompts are:

```
Disk Drive 1 to Compare [2] ? 3
Disk Drive 2 to Compare [1] ? 1
Comparing Blocks: *************************
```

If any blocks mismatch then they will be listed, e.g.:

```
Mismatched blocks ...
    1    5    27
```

Note: Comparison should be done on a track basis, or at least a track from each disk will be read but the comparison would be on a block basis within each track (a track contains 8 blocks for 4K total).

D(uplicate

Purpose: Duplicates one disk to another. Prompts are:

```
Source Disk Drive       [3] ? 2
Destination Disk Drive [1] ? 1
Duplicating Blocks: *********
```

Note: Duplication should be done on a track basis for faster speed.

T(est

Purpose: Tests all of the mounted disks by reading and writing them. This command is very similar to the Verify command but this command does not prompt you for any options and it works until you tell it to stop (just press the ESCAPE key). All of the tests that Verify supports are done. Prompts are:

```
Format disks also (Y/N) [N] ?
```

If you answer "Y" here then all of the mounted disks are formatted.

Sample output is (this sample assumes you have only 1 disk drive):

```
Testing drive 1:  Formatting ...
Testing drive 1:  Sequential Read
*******************************************
Testing drive 1:  Sequential Write
*******************************************
Testing drive 1:  Sequential Read and Write
*******************************************
Testing drive 1:  Random Read
*******************************************
Testing drive 1:  Random Write
*******************************************
Testing drive 1:  Readom Read and Write
*******************************************
```

When you quit this test a summary appears:

```
Summary:
     Passes ..................... 23
     Drives accessed ............ 3
     Blocks read ................ 12,567
     Blocks written ............. 8,752
     Block errors ............... 5
     Format failures ............ 2
     Drives with errors ......... 1 2
```

Note: Each pass of this test performs all of the tests including the formatting (if specified).

Note: If a format fails then the format will be done again.

L(ist

Purpose: Lists all the mounted disks.

This can be handy if you want to see if the computer knows if a disk is in a drive. Lists the disk number, number of blocks, and if the disk is write-protected.

S(um

Purpose: Compute a checksum for a disk or for a specific block range. Prompt is:

```
Disk Drive to Checksum [2] ? 3
Disk Block Range [0-279] ?
```

Uses the same checksum as the disk image checksum if the whole disk is to be checksummed. Otherwise, uses the block checksum algorithm which is just a sum of the individual line checksums.

R(eturn

Purpose: Return to the main command line.

Note: One command that is not here and which would be a great command to have is a disk speed command. This is an important aspect of disk drives since slow or fast disk drives can produce disks that are unreadble by other computers. This command is not present here since I don't have access to programming code that does this in a simple fashion (I could pull this code out of the Apple /// Disk /// formatter driver source listing but that seems rather complicated to me). If anyone has a code library with such a routine that I can link to a Pascal program that would be very welcome.

H(elp

Purpose: Display a brief description of this program.

"This program's purpose is to yadda yadda yadda ..."

Q(uit

Purpose: Quits the program, but asks the user first.

Really Quit (Y/N) [Y] ?

When this program starts it will display information about itself such as the program name, version number, creation date, and author info. This information will be followed by the command line. For example,

```
Apple /// Disk Image File Sender and Receiver Utility
Version 1.0.0 -- 24 Feburary 1999
Written by David T. Craig
```

Future Features:

- Saving disk images to other attached drives such as a hard drive or a 3.5" 800K floppy. This would allow users to make disk images on their machine and later transmit them to another machine either directly via disk or via their own telecommunications program such as Access ///. I don't envision this feature being part of the first release but a definite possibility for version 2.

  This feature could exist for both the /// and the //.

- Saving non-140K disk drives such as the ProFile 5MB hard disk. This would allow users to dump the contents of these drives as a disk image. Recreating these images could be a problem that I need to check into further. The user could specify these devices via their device name such as .PROFILE. The user would need to know the number of blocks on these other devices.

  This feature would only exist on the /// and not on the //.

8.0 APPLE // DISK SUPPORT

This program should also be implemented so that it runs on the Apple // family of machines (e.g. Apple][, Apple][+, Apple //e, Apple //c, Apple //gs). The reason for this is the low-level disk format of the Apple /// 5.25" disks and the Apple // fmaily 5.25" disks are the same (the directory structures may differ but that is immaterial at the low-level block level).

This Apple // family version program should also fully support the sending and receiving of Apple /// disk image files.

Disk drive specifiers may be different on the // version of this program since the // family does not use .D1, .D2, etc for drives. I will need to check how disks are specified in the // world. Or, the /// drive specifiers could be used and the // program would translate them to what it really needs. See the above discussion for more about this.

This program should use the Apple // family serial I/O devices such as the Super Serial Card in the Apple //e or the built-in serial port of the Apple //c machine (a //GS machine could also be used if this has a serial port, but I'm not familiar with this machine so I can't say more at this time).

This program will most likely not support automatic disk formatting on an Apple //-class machine when a disk image is received from an external machine. I believe the Apple // family OS does not support a formatting driver such as the /// supports. I know you can get the formatter for Apple // Pascal use but I don't have it and don't want to bother at this time with this since the user can just pre-format a floppy for disk creation use using a program such as Apple's ProDOS utility.

9.0 PROGRAMMING DETAILS

The program should be programmed in a high-level language and not assembly language. Why? I'm too old to deal with assembly any more and assembly is just not needed. I plan on using Apple /// Pascal for the /// implementation. I will most likely create the /// version first and then use the ///'s sources for the // implementation. The source should have conditional compilation statements governing the destination platform so that only a single instance of the source code need to exist. For example, the first line could be:

```
{$SETC APPLE:=2}  { /// system use 3, // use 2 }
```

This is a built-in Apple /// Pascal conditional that can either be 2 or 3 (The default is 2). When set to 2 the /// compiler automatically generates Apple // Pascal codefiles. Any areas of the program that are // or /// specific (e.g. disk formatting) would be bounded by the APPLE conditional compilation test and do what is needed for that area.

Disk I/O will be achieved using the Pascal intrinsic commands UNITREAD and UNITWRITE to read and write 512 byte disk blocks respectively. Special disk operations such as disk formatting will be done using Pascal's UNITSTATUS command (Apple /// SOS calls may also need to be made to handle such tasks as getting a SOS device's Pascal unit number whcih the UNITREAD/WRITE commands require). Disk reading should be done eithe ron a block basis or a track basis (a track contains 8 blocks and is 4K in size). The reason for the track access is so the UNITREAD call can access disks very quickly. For the send-disk feature of this program the track reading method may not provide any benefits since the serial I/O will most likely be the bottleneck. But for the program's disk access utility features such as verifying a disk then the track reading may make this feature very fast (I envision being able to read one track per second since that is what one Apple // program can do -- means a disk should be verifiable in around 35 seconds since a 140K disk contains 35 tracks).

Serial I/O will be achieved using the Pascal commands WRITELN and READLN output and input serial port data. The serial ports will be accessed using Pascal's device names REMIN: and REMOUT: (from an Apple // perspective) or the .RS232 device (from an Apple /// perspective).

Serial I/O configuring will not be done by this program. The user will need to make certain that their /// (or //) serial I/O are configured correctly for the remote machine they are using. The /// distribution disk for this program will come with the program and the necessary OS files so that the user can just boot this disk and this program will work with pre-configured serial I/O (e.g. 9600 baud, XON-XOFF flow control, ...). For the Apple // world users will most likely need to setup their serial I/O card's DIP switches (ugh!). The distribution disk should also come with this specification and the source code for both the /// and // programs (I believe in making all of this program public so others with an interest in how it works or want to modify it can have everything they need).

The Apple // version of this program will be created with Apple /// Pascal using the {$SETC APPLE:=2} conditional compilation facility of the /// Pascal compiler (see page 109 of the Apple /// Pascal Prgorammer's Manual, volume 2 dated 1981). This allows a single source to exist for this program even though the same source will be used for two different target machines.

10.0 UTILITY PROGRAMS ON OTHER MACHINES

It may be useful to have some utility programs running on other machines (e.g. Macintosh) that deal with disk image files.

For example, there should exist a program that verifies text disk images files. Also, there may need to be a program that can convert a text disk image file to a binary disk image file for use by other programs that can deal with binary disk images (e.g. the various Apple // emulator programs seem to use binary images only). Likewise, there should be a utility that converts binary disk images to text files in case you want to send such a text image to my program to re-create a real 5.25" floppy disk.

The utility program should also provide a cataloging capability. I envision this reading the header of a disk image file and generating a text file with fields for each header line. This text file could be just an exact reproduction of the header information, or it could be a tab-delimited text file that a program such as a database or a spreadsheet could use and print in a nice fashion.

The utility program should also provide a disk image file renaming facility. I think this would be handy so that if the disk image files are not named in a standard fashion this utility would standardize the names. For example, I think the disk image file names should be "Apple3DiskImage_iiinnn.TXT" where "iiinnn" is the disk image number preceded by a person's initials. This information would originate in the disk image file header and the utility would extract this info to make the disk image file's new name. For example, if a disk image file has a disk image number of "DTC001" then the disk image file would be renamed to "Apple3DiskImageFile_DTC001.TXT". This name format is compatible with the Apple Macintosh and MS Windows machines. If the disk images will be stored on a machine with lesser naming cababilities, then a shorter naming standard could be used. For example, under some UNIX systems the names can be no longer than 14 characters. I recommend a naming standard of "A3D_iiinnn.TXT" for UNIX. For MS-DOS systems with the standard 8.3 naming convention I recommend "A3iiinnn.TXT". I think the utility program should ask the user for the disk name prefix and just append the disk image number and ".TXT" to the disk name. The user would then have the control here (the default prefix should be "Apple3DiskImageFile_").

One feature of this utility program that may be very useful is to write the files from a disk image file to native files on the host system. This would allow someone to easily obtain the files from a SOS (or Apple // Pascal) disk without having to run some type of special file transfer program). For example, the utility would know the directory and file formats of the disk and create corresponding files on the host machine. To make this simple I recommend not recreating the disk image's file directories but instead just write the files out with a leading sequence number. For example, if a disk image has the directory and file organization:

```
Disk
     Directory A
          File A1
          File A2
     Directory B
          File B1
          File B2
          Directory B2
```


"_118.PICT" 251 KB 2001-08-25 dpi: 300h x 300v pix: 2472h x 3213v

File B21

The utility would create 5 files:

0001_A1 0002_A2 0003_B1 0004_B2 0005_B21

Note that the actual file names will be present as suffixes on the host machine files. Therefore, if a disk image file was named "SARA.HIST.TEXT" then the host file name would be "0001_SARA.HIST.TEXT".

I recommend that at least Apple /// SOS disk formats be recognized. It may also be useful to recognize Apple // Pascal disk formats since these are easy to parse (no subsdirectories exist and the file block allocation is very simple compared to SOS's multiple directory format with discontigious file block allocation).

I also recommend that SOS text files be converted to a more regular form when saved to a host machine. By this I mean remove the text file header (2 blocks) and save each file page (2 blocks) without the special text formatting stuff that is a remnant of the old UCSD P-System days.

I also think it would be neat if this utility program also generated a report for a disk image detailing the gory file system details of the disk. For example, list how many directories are on the disk, the characterists of each directory and file, and the blocks where each file resides. This could give people (such as this utility's author) a very good insight into the organization of a SOS disk. This should also support UCSD Pascal disks that the Apple // Pascal system creates.

The user interface for this utility program should also be a single command line. This makes using this program and if implemented on various machines this interface would be easier for different people to use since it would be standard across platforms.

These programs could all be the same program. I most likely will create such a utility for my Macintosh.

11.0 REFERENCES

Apple /// Owner's Guide
Apple Computer, 1981

Apple /// Plus Owner's Guide
Apple Computer, 1982

Apple /// Standard Device Drivers Manual
Apple Computer, 1981

Apple /// Pascal Programmer's Manual (volumes 1 and 2)
Apple Computer, 1981

Apple /// Pascal Technical Reference Manual
Apple Computer, 1983

Apple /// SOS Reference Manual (volumes 1 and 2)
Apple Computer, 1982

/// Bits: John Jeppson's Guided Tour of Highway ///
Softalk magazine, May 1983

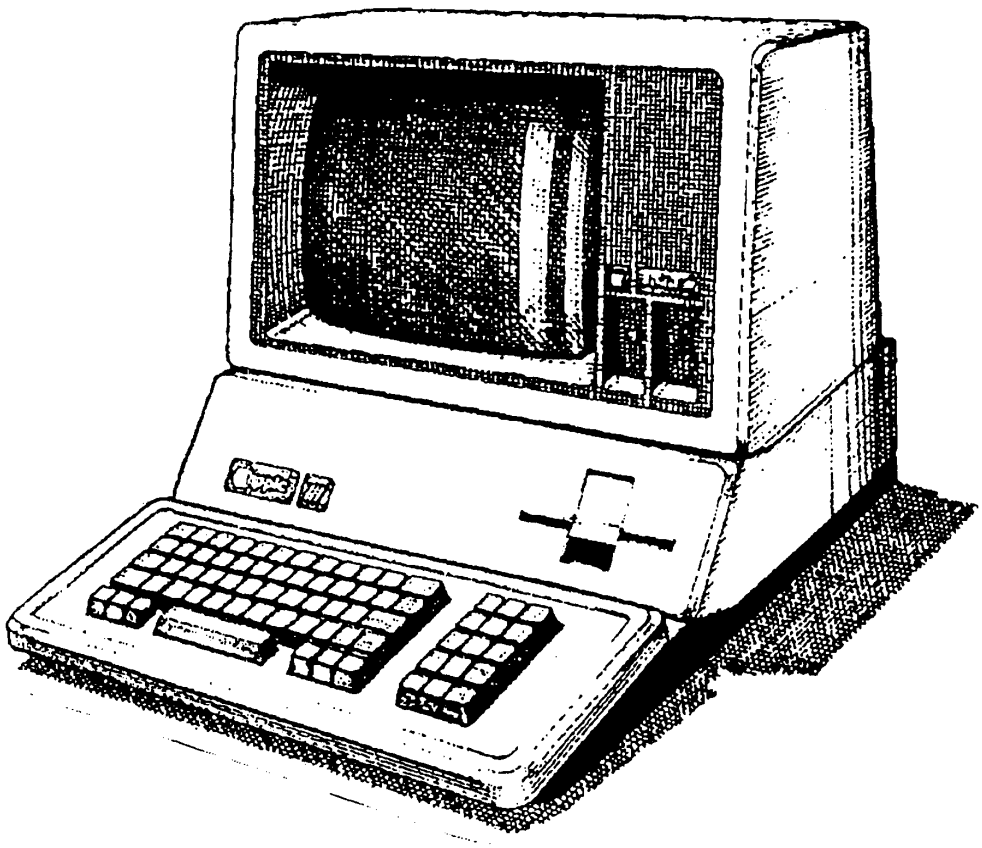
A Little SOS with your Pascal
Tim O'Konski, BYTE magazine, December 1982

Pascal source code to format SOS devices
James Vandermade, no date

Extraordinary Pi
Web site (www.users.globalnet.com.uk/~nickjh/pi.htm)
This is where I obtained the Pi digits.

F I N I S

Apple III Computer Information



End of Document

"_121.PICT" 178 KB 2001-08-25 dpi: 300h x 300v pix: 2472h x 3213v