APPLE II FLOPPY DISK DUMPER

David T Craig : Revision 3 : 28 November 2002

AUTHOR

David T Craig
941 Calle Mejia # 1006
Santa Fe, NM 87501 USA
Phone: (505) 820-0358
eMail: shirlgato@cybermesa.com

PROGRAM PURPOSE

Dump (export) Apple II computer floppy disk images to an external storage device using the Apple II computer's I/O port.

Program transmits a disk image to the external storage device as a hexadecimal data dump containing all of the disk's raw data blocks. A block is a 512 byte chunk of data stored sequentially on the disk.

Supports 140K 5.25" disks containing 280 blocks numbered 0 to 279.
Supports 800K 3.5" disks containing 1600 blocks numbered 0 to 1599.

Program supports all Apple II compatible disk formats including Apple II DOS 3.3, Apple II ProDOS, Apple II Pascal, Apple II CP/M, and Apple III SOS. Copy protected disks are not supported, though these can be dumped but some blocks will most likely be unreadable.

REQUIRED EQUIPMENT

Apple II family computer with serial or parallel I/O port or card. Apple II, Apple II+, and Apple IIe computers will need an I/O card such as the Apple Super Serial Card, or a parallel card such as the Apple Parallel Interface Card. Apple IIc and Apple IIgs computers don't need a card since these computers have built-in serial ports.

Computer should also have a 140K 5.25" disk drive such as the Apple Disk][drive. Computer can also have 800K 3.5" disk drives.

A display monitor would also be very useful :-)

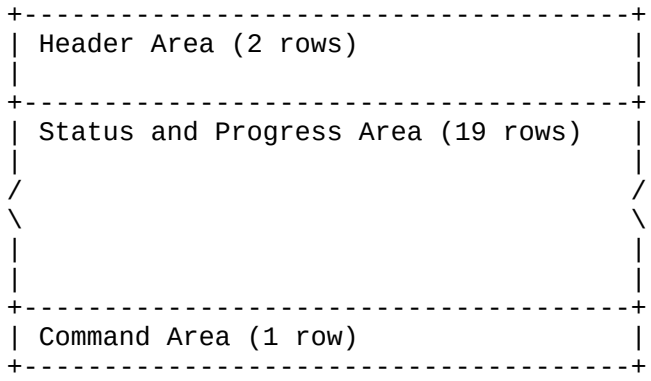
PROGRAM OPERATION

When program starts it determines what drives are present and if disks are present in each drive. User can insert or remove disks from any drive. User instructs the program to dump all the on-line disks to an external storage device.

Program supports up to 6 attached drives.

USER INTERFACE

Whole screen in 40 column by 24 row mode is used for the program. Screen displays all status and progress information for the user and lets the user issue commands to the program. Screen is divided into 3 areas as follows:



Header Area:
Displays program name, version number, and author information.

APPLE II FLOPPY DISK DUMPER v 1.0.0
Written by David T Craig (27 Nov 2002)

Command Area:
Lists all commands available to the user. Each command has a name whose first letter is what the user presses to execute the command. For example, to execute the List command, the user presses the "L" key.

L)ist D)ump V)erify S)creen P)refs Q)uit

List	Lists all of the on-line drives and whether they contain a disk and the disk size.
Dump	Dumps all the on-line disks to the current output device.
Verify	Verifies all or one disk by reading all of its data.
Screen	Dumps a single disk's data to the screen.
Prefs	Lets user specify various program preferences.
Quit	Quits the program.

Status and Progress Area:
Displays all status and progress information. Also prompts the user for additional command information.

COMMAND DETAILS

List
Lists all on-line drives as follows:

```
Drive 1 - 140K - Blocks:  280
Drive 2 - 140K - Blocks:  280
Drive 3 - EMPTY
Drive 4 - 800K - Blocks: 1600
Drive 5 - 140K - Blocks:  280
Drive 6 - EMPTY
```

Dump
Tests which disks are on-line. Asks user for descriptive information about each of the on-line disks, then dumps each on-line disk's blocks as follows:

```
Disk <x> information:

? DTC Apple II Disk # 12 - DOS 3.3
? Apple Presents the Apple //e tutorial
? Apple Computer Inc. 1983
? Product # 680-1234-01

(Press ESCAPE to stop disk dumping)

Dumping Disk # 1 ... Block dddd of dddd
Dumping Disk # 2 ... Block dddd of dddd
Dumping Disk # 4 ... Block dddd of dddd
Dumping Disk # 5 ... Block dddd of dddd
```

Four lines of text per disk are allowed. Disks blocks are dumped in a special format (see below for details).

Verify
Prompts user for the drive number of an on-line disk to verify. Verification consists of reading all the blocks and telling user if any blocks could not be read. Display is:

```
Drive to verify (0 for all) ? 4

Verifying 800K disk in drive 4 ...

(Press ESCAPE to stop verification)

   7 [-64]  128 [-32]  279 [-64]

Total bad blocks: 2
```

Notes: Each bad block is listed in format block-number [error-number].

Screen
Prompts user for drive number to dump to screen and then dumps the disk's blocks to the screen.

```
Drive to dump to screen ? 1

Drive 1 contains 280 blocks

Starting block number ([   0]) ? 10
```

```
Ending   block number ([ 279]) ? 12

DISK 1 - BLOCK   10 - CHECKSUM $AAFF

ADDR  0 1 2 3 4 5 6 7      ASCII
----  ----------------  --------
0000  1256AB44602058FC  ..A..9%$
           . . .
01F8  1256AB44602058FC  ..A..9%$
```

Prefs
Prompts user for program preferences:

```
Select the output device to use
for disk dumps:

Modem   - modem port
Printer - printer port

Output device (M or P) ? P

Enter disk owner name information:

? David T Craig / Santa Fe NM USA
```

Quit
Prompts user if quitting is desired.

```
Really quit (Y or N) ? Y

That's all folks ...
```

DISK BLOCK DUMP FORMAT

Each disk's data is dumped in a very special format whose goals are to be readable to people (i.e. text only, no binary information), be self describing, and be error free.

Each dumped disk's data has the following general format ($ represents a hexadecimal number, e.g. $FFFF):

```
Header
Disk Decription
Disk Information
Disk Block Checksums
Disk Blocks
Footer
```

Header:

```
******************************************************
* APPLE II FLOPPY DISK DUMPER v 1.0.0 (27 Nov 2002)
* Written by David T Craig (shirlgato@cybermesa.com)
******************************************************
```

Disk Decription:

```
>>> DISK DESCRIPTION

Description 1 : ...
Description 2 : ...
Description 3 : ...
Description 4 : ...

Disk owned by : ...
```

Disk Information:

```
>>> DISK PHYSICAL INFORMATION

Disk Size     : 140K
Disk Blocks   : 280
```

Disk Block Checksums:

```
>>> DISK BLOCK CHECKSUMS

Block    0      Checksum: $12AB ddddd
Block  ...
Block  128      Checksum: *** ERROR -64 ***
Block  ...
Block  279      Checksum: $FFFF ddddd
```

Disk Blocks:

Sequential list of all of the disk blocks starting at block 0 and ending at the disk's last disk block. Disk block data is stored in Pascal order which is the same as ProDOS disk order. Each block contains 512 bytes.

>>> DISK BLOCK DATA

BLOCK # $hhhh dddddd CHECKSUM $hhhh dddddd TOTAL BLOCKS $hhhh dddd

```
BLCK:ADDR 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F             ASCII
----:---- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- ----------------
0000:0000 hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh aaaaaaaaaaaaaaaa
...
0000:01F0 hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh aaaaaaaaaaaaaaaa
----:---- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- ----------------
BLCK:ADDR 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F             ASCII
```

h represents a hexadecimal digit (0-9, A-F).
d represents a decimal digit (0-9).
a represents an ASCII character (characters in range 0-32 and 127-255 print as "."s).

Notes:

If a disk block cannot be read, then the following appears after the BLOCK line (checksum will be 0 for these blocks):

```
*** ERROR: BLOCK COULD NOT BE READ
*** 64 - DEVICE ERROR
```

If a disk block contains the same byte, e.g. $00, then the following appears in place of the block's data dump:

```
*** NOTE: BLOCK CONTAINS SAME BYTE $hh (ddd) "a"
```

This feature should reduce the size of disk block text files since many disks will have unused blocks that tend to contain the same value, typically 0, if the block has not be used previously for a now deleted disk file.

Block checksum is a number whose purpose is to provide some degree of integrity checking that the disk block's dump data will not be corrupted by the block dump. For example, if the dumped block data was corrupted during transmission a reader of the dumped data would calculate their own checksum for the block's dumped data and if this checksum differed from the listed checksum, then the block data is corrupt.

Checksum is calculated as follows. Every byte of block data is exclusive-ORed (XOR) together starting at the first byte. A running XOR total is maintained as an 8-bit (1 byte) number. The XOR process is then re-done starting at the last block byte working toward the first byte. The 2 XOR totals are then merged together to get the final checksum value. The first checksum byte becomes the high byte of the 2 byte total, the second checksum becomes the low byte. A checksum running total byte is initialized to $A5 (165) for the forward checksum calculation. For the reverse checksum calculation, $5A (90) is used.

This algorithm should be portable with other systems and programming languages. It should also be simple to implement since it is byte based and no big-endian versus little-endian issues should arrise.

Checksum example:

Given a 3 byte block containing $01 $02 $FF, the checksum is calculated as follows:

```
Forward checksum:       $A5 XOR $01 -->   $A4
                              $A4 XOR $02 -->   $A6
                              $A6 XOR $FF -->   $59   (msb)

Reverse checksum:       $5A XOR $FF -->   $A5
                              $A5 XOR $02 -->   $A7
                              $A7 XOR $01 -->   $A6   (lsb)

Final checksum value:                           $59A6
```

Footer:

```
******************************************************
* END OF DISK DUMP
******************************************************
```

DISK DUMP USAGE

Disk dumps will be stored as text files by external systems that receive the output from this program.

These text files in themselves will not be too interesting. But these text files can be used to re-create a disk. For this to happen there will need to be another program that runs on the Apple II which reads these text files from an I/O port and writes the block data to a disk.

The next project here is obviously the creation of the dump-to-disk program.

Apple II emulation programs may also use this program's disk dump text files once these text files are converted to a format that the emulators support. There are currently many different emulators that support different disk image file formats. Three common disk image formats are:

ProDOS Order

> Each disk image file contains 143,360 bytes (140K) and has a file name suffix of ".PO" (e.g. "MyProDOSDisk.PO"). This program's text dump files correspond directly to this disk image format in that the disk block dumps when converted to binary will match the blocks that are stored in the ProDOS Order file format. A conversion program for this format needs only read the text dump files, convert each block to binary (i.e. 512 bytes), and save the converted block data to the image file. When done sequentially, the end result will be a ProDOS order image file.
>
> ProDOS Orders also matches the block format that the Apple II Pascal systems uses.

DOS 3.3 Order

> Each disk image file contains 143,360 bytes (140K) and has a file name suffix of ".DO" (e.g. "MyDOSDisk.DO").
>
> Converting this program's text dump files to DOS 3.3. Order files is somewhat complicated. Key here is to read each block and convert into into two 256 byte sectors corresponding to a specific sector in the DOS 3.3 Order image file. The block to sector conversion is fairly easy as long as you have a conversion table (the book "Beneath Apple ProDOS" has such a table).

2IMG Universal Format

> Each disk image file contains 819,200 bytes (800K) and has a file name suffix of ".2MG" (e.g. "My800KDisk.2MG"). This format is used for storing 800K disk images. Block order should match the ProDOS order.

Other disk image formats exist, but these three should cover the majority of emulator needs.

I plan to write at least the DOS 3.3 Order and ProDOS Order conversion programs for the Apple Macintosh computer. User will just run the programs in a folder containing dumped text files and the output will be the appropriate image file ready for emulator use.

COMPUTER DISK IMAGE TRANSFERS

Disk image transfers will be done using the Apple II serial or parallel card and a destination computer's serial or parallel port (or card).

For example, I plan to transfer my Apple II disk images from my Apple IIe (or IIc Plus) using the IIe Super Serial Card (or IIc Plus' built-in serial port) to my Macintosh and its serial port. On the Macintosh I will run a terminal program (e.g. GreatTerm) that will capture the Apple II's exported data and save it to a text file on the Macintosh. Several disk images will be exported per export session resulting in multiple images in a single Macintosh text capture file. I will extract each disk image from the Macintosh file and save each to their own text file for later use.

Other computers beside the Macintosh may be used, e.g. Windows PCs. The basic transfer and capture process will be similar.

Communication parameters such as baud rate will need to be set up between the Apple II and the destination computer. Doing a few exports from the Apple II should determine the best communication parameters.

PROGRAM DEVELOPMENT ENVIRONMENT

Program will be developed on an Apple II computer (either an Apple IIe or IIc Plus) using the Apple Pascal 1.3 programming environment. The program will be written in Apple Pascal 1.3. Program will contain only Pascal programming, no assembly programming will be used for portability purposes.

Modem port is Pascal's REMOUT: device. Printer port is Pascal's PRINTER: device.

Program will be named A2DISKDUMP.CODE on the program's distribution disk. This disk should be bootable on an Apple II computer (i.e. a turn-key system). All users need to do is insert the disk into an Apple II and then just turn on the computer. Disk will contain all the necessary system files that make the disk bootable.

Program can also be run directly from the Apple II Pascal environment by executing it from the Pascal command line interface.

Program should be developed in a modular fashion. This means the program will use Pascal units to segregate program modules and also make the main program source file fairly small.

Disk block checksum notes:

The disk block checksum should be calculated using tables for speed. Tables would contain the XOR results for 2 nibbles (half bytes) since Apple II Pascal does not include an XOR operator (this can be done in assembly language, but I want to avoid this language).

REFERENCES

"Apple 2 Floppy or Micro Disk Dump Program"
Handwritten note by David T Craig dated 23 November 2002.

"Apple II Emulator Resource Guide"
Web site with lots of good emulation information. Chapter 6 (Disk Image Formats & Conversion) is very useful for knowing about the different Apple II disk image formats.

www.zip.com/~alexm/faq/

###