TANDOS Commands

BACKUP

A disc copy utility
Syntax: BACKUP < CR >
Anatomy: Starts @ $B960. Ends @ $BB9B. Execs @ $B960.

This is a useful utility for single drive owners. It will copy whole disk contents without the need to do it file by file. It will first request the user inserts the source disc, then after reading as much data as available RAM allows, requests the user to insert the destination disc. This continues untill the whole disc has been copied. Caution as it will write over anything already on the destination disk.

Back to Index

COPY

A file copy utility
Syntax: COPY n:DESTFN.EXT< m:SOURCE.EXT [S P C M]< CR >
Anatomy: Starts @ $400. Ends @ $B10. Execs @ $530

The COPY command is a powerful command which permits copying of files from disc to disc. By using the various options it is possible to make copies of files on different discs, even if there is only one disc drive available. Options to delete old versions of files (supersede), merge files, change the write protection attribute are also available. Extensive wildcard facilities increase the usefulness of the Copy command even further.

Copying files

The simplest requirement is the transfer of a file from a disc on one drive to a disc on another drive. The command syntax for this is simply:-

COPY n:DESTN.EXT< m:SOURCE.EXT < CR >

The first file specification is the file destination and the second is the source. n and m may be the same, or omitted

(default is taken to be the current drive). If n and m are the same then the filenames must be different. In the event that you require the new file to have the same name as the old (must be different discs) then you may omit the destination filename e.g.

COPY 2: < 0:FILNAM.EXT< CR >

makes a copy of FILNAM.EXT on the disc in drive 2. The user is notified when a file has successfully been created on the destination disc.

Wildcards

Wildcards may occur only on the source (right hand) side of the '<' symbol. There are no other restrictions. Thus:-

COPY 2:< 0:*.* < CR >

will copy everything from drive 0 to drive 2. Once again, the user is notified whenever a file is created on the destination disc. This is especially useful when using wildcards as the full name of each file is printed permitting the user to follow the operation of the COPY instruction. Sometimes when using wildcards and you have true 'complex load-modules' on the disc you are copying from, the copy will fail and produce the message 'Insufficient space etc.' for and then a blank! This doesn't always happen but the reason it can't tell you what the title of the program is, is quite simply because it doesn't have one! What has probably happened is that it has failed on one of the load modules after the first one which is the only one really named by the directory. There is really no way around this problem because of the way COPY treats a 'load module' when in the process of copying but it is as well to be aware of it.

Superseding

It is sometimes necessary to delete an old version of a file and create a new version with the same name. This type of operation is simplified by the use of the S parameter. The command:

COPY m:JOE.EXT< n:FRED.EXT S< CR >

will generate a file named JOE.EXT on disc drive m from a source file FRED.EXT on disc n. If a file called JOE.EXT already exists on drive m then this will be deleted before the new file is created. The message JOE.EXT SUPERCEDED is displayed as a reminder. Files cannot be superseded if their write protect attribute is set (since this prevents their deletion).

Protection Attribute

Each file on a disc has a protection attribute. If this is set to 'N' (by default or by REN) then the file may be deleted. This status is shown in the directory listing by the absense of the 'P' attribute. If the attribute is set to 'write protect' ('P') then the file cannot be deleted by the DEL disc command or overwritten by any of the 'save' or copy disc commands (beware BACKUP, CREATE, FORMAT and INIT - these will completely erase a disc!). Copy normally assumes that files created by it should have the same protection attribute as the original file. If you wish to override this then you may specify N or P as parameters to any Copy command e.g.

COPY m:< n:*.* P < CR >

will copy all files from disc n to disc m and set the write protect attribute on all of them. It does not affect the write protect status of any files on disc n,

Single Disc Drive copying

COPY caters for operations which have to be performed on a single disc drive. It will prompt the user for Source and Destination discs to be loaded as necessary throughout the execution of any COPY command whenever the C parameter (C for 'Change Disc') is issued as part of that COPY command e.g.

COPY n:FRED< n:JOE C < CR >

Command execution proceeds in the following manner:-
1. The COPY program is loaded from the system disc,
2. The user is prompted: "LOAD SOURCE DISC & PRESS RETURN".
3. The source file is read.
4. The user is again prompted but with "LOAD DESTINATION DISC & PRESS RETURN".
5. The source file is written to disc.

There will be at least one read operation (from the source disc) and one write operation (to the destination disc) for every file copied. If the file is too long to fit in memory at one time then multiple read/write operations are needed. COPY tries hard to prevent you getting the discs confused. It checks the disc names (see INIT) every time you load a disc and warns you if it is not the disc it was expecting. This check is rendered useless if both discs have the same name, (it warns you about that too) hence the plea to use sensible names when INITing discs.

Merging Files

The user may occasionally wish to merge several files together into one larger file. This may readily be accomplished by using a Copy command with the M parameter. Other than disc size, there is no limit to the number of files which may be merged together. The command:

COPY n:FRED< m:JOE??? M < CR >

will merge all files with the first 3 letters of the filename being JOE on disc m into a single file called FRED on disc n. Note that merge operations require a filename on the destination side of the "<". Copy will inform the user of the progress of the command with a message each time the output file is written to (either its creation or for a file appended to it). FRED may or may not exist before the command is issued but it must not have the write protect attribute to set to 'P'. On completion of the above command COPY prompts with "NEXT:". If you want to append further files then simply type a new line specification at this point e.g.

NEXT: GEORGE.* < CR >

COPY will continue to merge and prompt with "NEXT:" until the user responds with only < CR >. One of the main uses for merged files is to create complex 'load modules' i.e. files which load and execute. You can use the DSAVE command to save various areas of memory separately (zero page, TANRAM, or even the VDU screen). See example below. After merging with copy then all of these areas will be loaded at one go merely by issuing the appropriate DLOAD or Auto-load command. Note that the transfer address (see DLOAD) is taken from the first part of the composite file only. It is perfectly legal to merge a file with itself (though it could be argued that it is rather a strange thing to do). COPY will perform correctly provided that the source file(s) fit in memory. If multiple copy operations are necessary then the operation will eventually fail with a message explaining that the disc is full (it won't be when you do a DIR - the partially formed file is automatically removed and no harm is done to the disc). This is an inevitable consequence of the source file getting longer every time you merge a bit onto it.

Merging Files Example

The idea of merging files is simply to be able to join together what are normally quite separate programs or files and run them as one. For example, to create the file VBASIC to auto-load BASIC using the VDU, it is necessary to merge 4 separate files together in order to create the whole thing at load time.

The first file is the old DBASIC (1 record). All this file does is to do the necessary initialising that the old GE2ED did with the addition of setting up disc buffers at the top of memory in case you want to use DBASIC disc file handling. These facilities are quite well explained on pages 4.1 to 4.5 of the manual. The only major difference between VBASIC's DBASIC (if you see what I mean) is that it also looks for a toolkit resident at $E800 as well and if there, powers it up. As long as the priority toolkit knows that the other is there, it is no problem at all to have both running quite happily together and doesn't require the use of USR()) at all. OK so that's the first file out of the way.

The next problem is that we want the BASIC interpreter and toolkit on disc to load into what is normally an illegal area of RAM between $C000 and $EFFF. Referring to the manual page 24 (RAM USAGE), you will find a label

entitled PAGDEF which resides between $B810-$B824. This is actually a copy of what is on the disc's system sector in bytes $8-$F (page 19). Let us assume that like the manual you have PAGE 0 set to $A1FF as the maximum RAM that the DOS is allowed to access. What we need to do is to create a single byte file which when loaded will change $B810 (page 0's definition) to read $EF which will then fool the DOS into allowing our BASIC interpreter+toolkit to be loaded quite happily. This is fairly easy as you simply change the byte $B81D to $EF by using the M command from the monitor! Then use either S or DSAVE as follows:-

S START B81D B81D

This is a waste of a sector but it serves a real purpose.

The next file to create on our disc is simply a dump of the BASIC INT with the toolkit already attached at SE800. So:-

S ROM C000 EFFF

will do the trick. Note that S or DSAVE will allow you to save from anywhere in memory regardless of what the PAGDEF says unlike DLOAD.

So we now have 3 files called DBASIC, START and ROM on disc but we still need a fourth to put back to what START did. DBASIC decides how much memory there is available on the system by looking at the byte at $B810 and not as normally happens when you power up BASIC (see manual page 16). If we leave $B810 to what START set it to then DBASIC when it runs, will get it sums wrong and chaos will reign! So our last file to be eventually merged into VBASIC will be like START but this time resetting $B810 back to what it should be. So END will be:-

S END B810 B8lD
remembering to change the byte before you save it!

Now according to the manual page 14 it ought to be a fairly simple task to use COPY to merge START, ROM and END onto DBASIC (renamed VBASIC) which should be on another disc first. Bearing in mind that VBASIC should be the only one to have a 'transfer' address set up, what should happen when we finish merging all the files together and then try and use our newly created VBASIC, is that BASIC should pose up as normal but having loaded all the other bits into memory first. Guess what? It will not work! The answer to why can be found on page 22 under the heading about 'LOAD MODULES'. The key to it all is byte 0 of the load module which should be $FF in a true load module. The trouble with COPY when merging is that it for some reason always sets this byte to $0 when merging a new file onto the last one. The end result is that you only get one true load module in your composite file ie the first one! This is completely contrary to what it says on page 14 about the copy merging facility and in fact COPY can NOT create 'complex load modules'! To prove a point, if you now dive into your newly created file called VBASIC which doesn't appear to want to work properly, with DZAP(SECFIX etc) and having found the first sectors of the 2nd, 3rd and 4th load modules, manually change the first bytes to $FF as it says on page 22 Now it will work.

Memory Usage

COPY uses memory from location $400 upwards (unlike nearly all the other TANDOS 65 commands). It will use almost all the available RAM during COPYing. If you have extension RAM (TANRAM) then copying long files will be quicker.

Back to Index

CREATE

Creates a new disc in Drive number n:
Syntax: CREATE n:
Anatomy: Starts @ $400. Ends @ $7E6. Execs @ $5A0.

This provides both the FORMAT and INIT utility operations in a single application. The Drive number must be

specified. In all cases the user will be prompted to insert the disc in the specified drive. A full listing of a version of CREATE created by Colin Nowell is available in TANDOC 3, Page 15.

Back to Index

DBASIC

Loads and runs disc-based BASIC
Syntax: DBASIC
Anatomy: Starts @ $B960. Ends @ $BA38. Execs @ $B960.

If the standard BASIC ROMs are installed, then the enhanced BASIC incorporating the various disc commands may be invoked by using the DBASIC command. This loads and runs a short initialization program from the system disc and jumps into the existing BASIC ROMs. The version on my Full System has been modified to first transfer the BASIC code and BASIC Toolkit from the ROMs located on the EPROM Storage Card (Combo Board) to the RAM location $C000-$EFFF. A second version named VBASIC has been created that does the same except it transfers the VDU Toolkit instead of the BASIC Toolkit (and other tweaks to ensure operability with the VDU 80/82 card). Disc BASIC adds a further 6 commands to the ROM version. No commands are lost. These commands are made available because DBASIC patches a RAM resident subroutine in the zero page.

Back to Index

DEL

Delete a file.
Syntax: DEL [n:]FILNAM.EXT < CR >
Anatomy: Starts @ $B960. Ends @ $BBDC. Execs @ $B980.

The delete command is used to erase files from the disc. The space previously occupied by the file itself and its directory information becomes available for use in storing other files. Delete will not operate on any disc which has the write protect tab fitted (delete is treated as a write operation). You will also be prevented from deleting any file with the 'Protect' attribute set (giving a 'P' in the directory listing after the file length). You may change the protection attribute with the REName command. Alternatively use the KILL utility.

Delete Single File.

If the delete command is used in conjunction with an explicit file specification (no wildcards) then only that file is deleted, (assuming it exists and is not protected, of course).

Wildcards

Delete may be used with wildcards. In this case the directory is searched for all files which match the specification. For each of these, the full filename is displayed and a yes/no prompt given. In the latest version, you do not have to press return after answering Y or N.

Back to Index

DIR

Directory Listing on disc in Drive n:
Syntax: DIR [n:]< CR > ....or.... DIR [n:]< LF >
Anatomy: Starts @ $B960. Ends @ $BB9A. Execs @ $B988.

The directory command is used to find out exactly what files are stored on a particular disc. It will also tell you how big each of your files are (in sectors - each sector is 256 bytes long. Although TANDOS 65 used 2 or more bytes per sector for its own purpose). You can tell if files are write protected and can see exactly how much room is left on the disc for more files. For directory listings of drive n: without page breaks, use the < LF > terminator.

Full Directory Listings

To obtain a full directory listing of the current disc you need only type DIR < CR > (current disc is 0 after a reset - see the DRV command for more information). A typical screen display is then:

```
DIR
 O:PAULK1 DIRECTORY PAGE 1
 PIMS. BDT 1 PIRATE. BAS 65
 STRTRK.BAS 34 LED. BAS 16
 PIMS. BAS 25 ELIZA. BAS 23
 ZODIAL. BAS 87 OTHELO.BAS 13
 FORWAR. OBJ 1 DIR 3P
 268 USED. 90 FREE OUT OF 358
```

The 'O:PAULK1'at top left is not a filename but a disc name (see INIT command). If you name your discs in an organised manner, you can use this feature to check that you have loaded the correct disc. The file names are the names of the files on the current disc. The numbers immediately to the right are the length of the files in sectors (in decimal notation). The 'P' following one of the numbers tell you that the file is protected from accidental erasure (it is "Write Protected" and hence any activity which attempts a write operation to that file will be blocked). The 'Used' and 'Free' refer to the total number of sectors used for files and which remain available for files. The 'Out of' tells you how many sectors are available for file use in total on the disc. This is not quite the same as the total number of sectors on the disc. TANDOS 65 uses 1 sector for the system sector and one directory sector for every 15 files on disc.

Paging

It is only possible to get the directory information for 24 files on the screen at once. If you have more than this number of files on the disc then DIR will provide two (or however many it needs) pages of directory information. It will not display the next page until you press < CR > in response to the >> prompt. < ESC > terminates the directory display. On older versions, whilst pressing < LF > instead of < CR > got rid of the >> prompt, it still caused gaps to appear on the listing which is undesirable on printer output. This has been taken care of in a re-write. Pressing < LF> will now simply give a continuous listing as it comes off the disc. This facility is also useful for those using the VDU screen with its extra line space which of course allows more files on the screen at once.

If you are using a printer, you may find it useful to avoid the next page prompt.

Selective Directories.

If you want the directory information for just one file (perhaps to check it is there or to see how long it is) you may not want to see the whole directory listing. You can obtain directory listings for any particular file by specifying the filename in the DIR command (in addition to the disc unit if necessary). e.g. DIR FILNAM.EXT. The display is exactly as before but only information about that particular file is shown (if there is no such file then no file information is displayed).

Using Wildcards.

Wildcards may be used when requesting a selective directory listing. Thus, to obtain a directory of all the Basic programs on the current disc, type DIR.BAS < CR > This file specification refers to any file with an extension BAS. Any legal combination of characters and wildcard characters may be used to give a powerful method of extracting the directory information needed.

Back to Index

DLOAD

Load file from disc
Syntax: DLOAD [n:]FILNAM.EXT [D N P S] < CR >
Anatomy: ROM Resident @ $B000

DLOAD is the most general way of loading files into memory - if the file is capable of execution TANDOS 65 will jump into it on load completion of the load operation. DLOAD transfers the specified file from the disc (either specified or the default 'current' unit) into memory. The addresses used to perform the transfer are stored as part of the file. An attempt to load to memory which the system does not believe to be present will result in an error message (NO MEMORY) and the load aborting.

DLOAD Parameters

DLOAD accepts various parameters to increase the flexibility of file loading.

D Display the start, end and transfer of control addresses. These are displayed in order in Hex beneath the DLOAD command (default is no display).

N No run. The file will not be executed on load completion - regardless of whether or not it would have been possible (the default is for executable files to be run on load completion).

Pn Page n. The file will be loaded to memory page n regardless of what page the file was saved from (which is the default).

SHHHH Start at location HHHH. H is a Hex number. The file will be loaded starting at address HHHH regardless of what location it was saved from (which is the default).

Auto Load

A simple version of the DLOAD command is made available by merely typing the file specification for the file you wish to load e.g. FRED < CR > will load (and if possible, run) FRED from the system disc. This is how disc resident system commands are implemented.

Parameters are not accepted for Auto Load (all the defaults apply). Under auto-load, anything following the file specification will be ignored by the load routines. The loaded program may use that data for its own purposes (just like commands for TANDOS 65 utilities).

Auto-load loads files from the system disc (as distinct from the current disc), unless you override this default by specifying the disc unit number you may auto-load from any disc e.g. 2:DIR will run the directory command from Disc 2:

The TANDOS Manual on Page 12 gives details of how to make full use of the auto-load facility.

All the TANDOS 65 commands (except DLOAD which is ROM resident) may be invoked in this way, as may programs you have written.

Back to Index

DRV

Select Drive specified
Syntax: DRV n < CR >
Anatomy: TANBUG/TUGBUG Command

The DRV command permits the user to select any disc unit number to be the default in subsequent disc commands.

DRV must be followed by any legal unit number (in the range 0 to 7). No check is made at this stage to ensure that the particular unit is available (that check is made by all TANDOS 65 commands which use the unit number).

The current drive number is reset to 0 (the system disc) whenever a reset is performed.

Note that programs executed by using the auto load command will always be loaded from the system disc (drive 0) unless the drive number is given explicitly. The current drive number is not used to define the auto load disc. Thus:

```
DRV 2
DIR
```

will run the copy of DIR from disc 0 and display a directory of disc 2.

Back to Index

DSAVE

Save memory as file
Syntax: DSAVE [n:]FILNAM.EXT [THHHH Pn]< CR >
Anatomy: Starts @ $B960. Ends @ $BB8C. Execs @ $B9AF.

DSAVE is the mechanism provided for saving areas of memory in disc files. The command prompts for two Hex addresses - Start and End. These are the addresses at which the save process should begin and end respectively. A check is made to ensure that End is > = Start.

The resultant file is not normally executable i.e. DLOAD will not attempt to cause the processor to jump into the loaded memory image. If you require an executable file then you must specify the T parameter.

DSAVE Parameters

THHHH Transfer address. HHHH specifies an address in Hexadecimal to which the computer should jump in order to execute the saved program. Default is 0 which DLOAD interprets as 'no transfer'.

Pn Page n. where n is a page number from 0 to 7. DSAVE will save the memory from page n. Default is Page 0. The page selection logic is left pointing to Page n on termination of the command.

You cannot DSAVE memory in the special disc board RAM at $B800 to $BBFF. This is because TANDOS 65 loads the DSAVE programs itself into that area which, of course, will overwrite the previous memory contents. Use the TANBUG memory copy command (Cx,y,z) to move the contents of $B800-$BBFF to any convenient location and then DSAVE from there.

Back to Index

DZAP

Latest Disc Sector Editor
Syntax: DZAP< CR >
Anatomy: Starts @ $B960. Ends @ $BB89. Execs @ $B960.

DZAP is the result of a much rewritten version of the original TANDOS utility called SECDMP and its re-written version SECFIX.

The main change from SECDMP to DZAP is that the screens (text input and sector output) are much more separated in the way they are controlled. In fact the graphics cursor that can roam about the lower half of the MICROTAN screen, is now controlled in real time and this control has no affect on the upper (text output) part of the display (except in the case of CONTROL O).

Command Summary

U(drive)

Select disc drive. Drive must be in the range 0-7. No check is made to ensure that the drive actually exists.

R(sector,track)

Sector read primitive. This downloads the specified sector, track into the lower half of the screen. Note that the order of entering the sector, track number has been reversed from SECDMP. The main reason for this is that it is now unnecessary to re-enter the track number if the sector required is on the same track as the last entry, ie R5 will read sector five from the same track as last time.

W(sector,track)

Sector write primitive. Comments as for sector read in every way except this is obviously the inverse function.

Exxxxx

Enter an ASCII string specified in "xxxxx" into the screen buffer (lower). Note that only what is on the screen line can be transferred so if the cursor 'falls" off to the next line, that text will NOT be transferred and the statement error "?" will be generated. The text will enter from the current buffer cursor position.

HFF(,FF,FF)

Transfer the Hex bytes specified to the screen buffer. Leading zeros are ignored and as many bytes it can fit on one screen line may be transferred at once. The bytes will be entered from the current buffer cursor position onwards and its position automatically updated.

L (0-FF)

Locate the screen buffer cursor at the specified position in the lower half of the screen. Note that the parameter is in Hex and that the display now includes an "index" line immediately above the lower screen in order to help with positioning. This command has largely been made redundant by the change in nature of cursor control which is not now as cumbersome and faster.

F

F will F(ill) the screen buffer with the specified ASCII character following the F on the screen. Note that F< CR > will clear the screen buffer under index line.

X

X will invoke a tidy exit from the program DZAP.

CONTROL COMMANDS

The rest of the commands from SECDPP are now no longer valid and you do not have to enter MRMRMR etc for 3 moves right of the cursor. The graphics cursor in the lower half of the screen is now controlled in real time by our old 'Control key' friends ^R, ^L, ^U, ^D for right, left, up and down respectively. This cursor control can occur at anytime and will NOT affect in any way what it going on in the upper text part of the screen. For example you may be in the middle of typing in a string of characters after the E command ready to transfer when you realise that the buffer cursor is in the wrong position. No problem. Just change the cursor position using the control keys and carry

on typing your text.

The biggest change from the old SECDMP program is probably in the P command which is no longer valid and has been replaced by ^O (O is for output). As in the cursor control functions this key is also 'live' in that it can be used at any time and from any position. On first use of ^0, the byte underneath the cursor in the buffer will be output to the upper text screen as a hex number followed by a ','. The cursor in the buffer moves right one position. You have now entered the output mode and will remain in it until a < CR > is issued from the keyboard. Further use of the ^O key will result in the next byte being output etc. This means that whole group of bytes may be output to a single line on the text screen for easier checking when applicable. Note that these CNTRL functions 'wrap around'. i.e. the graphic cursor will drop onto the next line if using ^R or ^0 etc.

General Comment

DZAP resides in the 'transient program area' of the RAM on the DOS card (as was SECFIX) and NOT in normal RAM so in most cases it will not upset or infringe on normal program operation. If the VDU 80/82 board is resident in the system and you are running under TUGBUG, then the VDU board will have its own screen cleared and then further output to it will be suppressed on start up of DZAP. The X command will reinstate the VDU board. Beware of errors occurring due to the DOS. In most cases DZAP will trap out any scrolling into the lower part of the screen by virtue of its own screen handling but should a DOS error occur then it is possible for the message that gets output by the DOS to upset the screen formatting, the easiest way around this is to issue a reset then type in GB960 for an orderly return to DZAP operation.

Back to Index

FORMAT

Formats the disc in the drive specified
Syntax: FORMAT n:< CR >
Anatomy: Starts @ $400. Ends @ $6BD. Execs @ $589.

The format utility program lays down the basic magnetic pattern on the disc surface so that future operations can find particular areas of the disc (broken down or 'mapped' into tracks and sectors). It is always necessary to format new discs before using them for anything at all (even non-TANDOS 65 things like DFORTH needs to use formatted discs. You should not need to use FORMAT again unless you have suffered a really catastrophic disc crash. Obviously, the FORMAT utility will destroy, beyond recall, any files or other information previously stored on the disc. FORMAT makes use of the information held in the system sector on the system disc to determine the number of tracks on the disc to be formatted. To format a disc, issue the command as in the above example. Note that you must give the disc unit explicitly (this is to try and avoid nasty accidents). The utility will be loaded from disc and will then issue a prompt to 'LOAD DISC n & PRESS RETURN'. You should then load the disc to be formatted, check that you have given the correct command and, if all is well, press < CR >. Formatting will then take place and after about 10 seconds (for a 40 track disc). A message will inform you that formatting is complete. If you wish to escape from formatting after issuing the TANDOS 65 command then you should press < ESC > when requested to 'LOAD DISC n & PRESS RETURN'. This will cause a return to Tanbug without writing to your disc. Note that FORMAT is one of the few TANDOS 65 commands to use memory from S400 upwards.

Back to Index

INIT

Initialise a disc in Drive n.
Syntax: INIT n:< CR >
Anatomy: Starts @$B960. Ends @ $BB46. Execs @ $B9CB.

Initialising the disc after formatting is as important because the disc will just appear to be completely empty to the

DOS and therefore will not be able to make any sense of it. This process involves two main activities. One is to set up the system definitions sector (Sector 0 of Track 1) and the other is to set up and define the 'free space chain'. All existing files are lost, regardless of their protect status. It is not, however, possible to INIT a disc with the write protect tab in place on the disc itself.

INIT loads from the system disc and then prompts the user to load the disc to be initialised. This provides a suitable opportunity, if needed, to remove the system disc and reload the drive with another disc. This is especially important for users with only a single disc drive. The user is next prompted for a disc name. This may be any combination of up to 9 alphanumeric characters. Try and use unique names for all your discs so that you can distinguish between them later. Again this is more important for users with single disc drives - under some circumstances TANDOS 65 attempts to check that the correct disc is loaded - it cannot do this if discs have the same or no name. After the disc name is terminated by < CR >, the initialisation process is started. This takes about 40 seconds for a single-sided single density, 40 track disc.

For double-sided discs, each side must be initialised separately.

The system sector information (amount of RAM, number and type of discs) on a newly initialised disc will be the same as the current system disc. Run SYS if you need to change this.

Make absolutely certain that you do not INIT a disc by accident. There is no way that the old files can be recovered since every disc sector is overwritten during the initialisation sequence. You may escape from INIT, without harming your disc by pressing < ESC > in response to the 'Disc Name' prompt.

The main difference between the latest INIT and the original INIT is that it can initialise 10 sectors but more than that it is also the way that it is achieved. The original INIT was exceedingly clumsy at the way that it achieved its aim and not very conducive to being adapted. The latest INIT changes all that and although outwardly the same, it now uses a look-up table for the sequence that it should use when setting up the free space chain. For the old 9 sector format, the access sequence was 1,4,7,2,5,8,3,6,9. With an extra sector and some trial and timing testing, it was found that overall disc access was significantly faster when 1,4,7,10,3,6,9,2,5,8 was used as opposed to the more obvious 1,4,7,10,2,5,9,3,6,9! The resultant time taken to INIT a disc was cut in half helped by the faster DOS and format. This flexibility will also help in the future when it is desired to investigate double density operation and its consequent change in access sequence. Note that the latest INIT uses the Transient Program Area (TPA) unlike the original.

Back to Index

KILL

Latest Delete file utility
Syntax: KILL [n:]FILNAM.EXT < CR >
Anatomy: Starts @ $B96O0. Ends @ $BBAE. Execs @ $B970

All comments about the DEL command also apply to this new command KILL. It was born simply as a short cut to enable the deletion of files with the protection status set SO BEWARE! Operation using wildcards is identical to DEL.

Back to Index

REN

Rename a file
Syntax: REN [n:]NEWNAM.EXT < [n:]OLDNAM.EXT [P N]< CR >
Anatomy: Starts $B960. Ends @ $BA69. Execs @ $B985

The Rename command is used for two different purposes.

The main use is simply to change the name of a file. However you cannot rename a file which is 'protected'.

The secondary use is to change the protection status of a file using P (protected) or N (non-protected) optional parameters.

Both the source and destination file specifications (i.e. those on either side of the '< ' symbol) must refer to the same disc unit. (It should be obvious that merely renaming a file cannot transfer it to a different disc).

Wildcards are not accepted by Rename.

Back to Index

S

An alternate to the DSAVE utility
Syntax: S [n:]FILNAM.EXT START END [THHHH RHHHH Pn]< CR >
Anatomy: Starts @ $B960. Ends @ $BBFC. Execs @ $B9A7

If the filename is already on the disc there is first an option to delete the old copy from the disc. Answer Y(es) or N(o). Then a second option allows the user to make the original version a backup by adding the extension 'BAK'. It will be deleted automatically if it exists. Answer B(ackup).

START and END are the start and end addresses of the data to be saved. These are mandatory. T, R, P are optional and may be entered in any order.

THHHH is the transfer address when the data is loaded the loader will jump to these address.

Px is the Page Number that the data will be saved from and subsequently re-loaded to.

RHHHH enables the data to be re-located when next loaded from the disc. This can be used for saving programs that have been relocated in order to assemble them.

When using 'R' the 'T' address must be the correct one. It is not adjusted by the save routine.

Errors

There are 3 errors that can occur:-
1) A syntax error in the command
2) There is not enough space on the disc
3) One of the files is write protected
The errors are reported as 1) SYN 2) NSP & 3) WRP

This new version of DSAVE first appeared in the TUG newsletter No.26 Page 16.

Unlike DSAVE, which will prompt for start and end addresses of the program or file to be saved on disc, S expects all its information including the title, on the same line and hence no core typing is required after that. This is why S is called simply 'S'!' There wouldn't be enough room for a longer name.

This isn't really the biggest advantage because under the simple exterior of this command is a re-locator. This is no ordinary machine code re-locator and cannot go through all your code changing all the 3 byte opcodes as necessary. The one thing that DSAVE would not allow you to do was to dump a program onto disc so that when it came back into memory would run in the Transient Program Area (TPA) ie. like any other of the utilities. It is possible using DLOAD with an 'S' attribute afterwards but you then had to manually start the program running ie. you couldn't AUTO LOAD it as in 'DIR < CR >'. S will allow you to assemble the code of your program with say, the DASM utility using an offset, to another area of memory but still generating 3 byte opcodes as if it were resident in the TPA. Then when you save to disc, the optional attribute will automatically change the main 'LOAD MODULE' (see

later) addresses so that when it next gets loaded by an auto load command, it will be loaded into the TPA and then RUN immediately. In other words you cay now create your own TANDOS utilities very simply.

But that is not all. S also allows you to specify an existing file as a saving module and gives you the choice of whether to delete the old file or to make it a backup by adding the suffix 'BAK' to the old one.

The rather terse error messages from 'S' (NSP-No space, SYN-Syntax and P-Write protected disc) are necessary because there is no more space available in the TPA to make the S program any bigger.

Back to Index

SECDMP/SECFIX

Early Disc Sector editors
Syntax: SECDMP < CR >
Anatomy: Starts $400. Ends @ $5CD. Execs @ $402
Syntax: SECFIX < CR >
Anatomy: Starts $B960. Ends @ $BB44. Execs @ $B960

Secdmp (SECtor DuMP) was written as a purely diagnostic/debugging tool for use during the development of TANDOS 65. It so far exceeded expectations, however, that it was included as part of the TANDOS 65 package. The program permits the user to examine freely, any sector, anywhere on any disc. Sectors may be modified and rewritten to disc anywhere. Whilst the experienced user will find the program useful, it is a relatively 'dangerous' program in that it is extremely easy to ruin the contents of a disc (you are effectively working outside the protection of the operating system). SECDMP can only be used with the Microtan screen as the output device.

SECFIX is a later version of SECDMP re-written to reside in the Transient Program Area (TPA). Both have been superceded by DZAP.

Running the Program

When loaded, SECDMP clears the screen and prompts for user input with a question mark.

User communication is echoed in the top half of the screen. The bottom half of the screen is used to display the Sector Buffer.

Available Commands

Only one command per line is permissible and all commands must be terminated by < CR > .

X Return to Tanbug the screen is cleared on exit,
Un Select disc unit n. n must be in the range of 0-7. No check is made to ensure that the unit actually exists.
Rt,s Read a sector at track t, sector s to the Sector Buffer. No check is made on the validity of either t or s.
Wt,s Write the contents of the Sector Buffer to the disc sector at track t, sector s. Again t,s are not checked for validity. (The cursor blinks out during the write operation).
Fc Fill the sector buffer with any ASCII character 'c'.
M[cursor string] Move the Sector Buffer cursor according to the directions in the cursor string. The following characters are understood as cursor movement characters:
L left
R right
U up
D down

'Wrapping' occurs at the edges of the displayed Sector Buffer (e.g. going Right when at the right hand edge leaves you at the left and down 1 line). The cursor string may be as long as desired but the entire command must fit on 1 screen line.

Lhh Locate the Sector Buffer cursor at byte hh. hh is interpreted as a Hexadecimal string. Only the last two characters are considered significant e.g. L0 will locate the cursor at the start of the buffer. LFF will locate it at the end as will L01 FF (the 01 is ignored since SECDMP will only look at the last two characters FE).

E[ascii string] This command enables ASCII characters to be written into the Sector Buffer. Writing starts at the current position of the cursor and the cursor is updated. The string may be as long as desired providing the command fits on a single screen line.

Hhh ,hh... Hex. This command enables hexadecimal values to be written to the Sector Buffer. Each hex value is written to the buffer starting at the cursor. The cursor position is updated. Hex values must be separated by commas. Only the last two characters of each hex value are considered relevant. Any number of hex values may be written to the buffer provided only that the entire command fits on one screen line.

P Print. This command causes the hex value of the character 'under' the cursor to be printed (next to the P). The cursor position is incremented. The command is particularly useful for looking at quantities which are not ASCII, particularly as it is not possible to distinguish between characters with and without bit 7 (MSB) set, when displayed on the screen.

Back to Index

SYS

Runs the system definition Utility on specified disc
Syntax: SYS n: < CR >
Anatomy: Starts $B960. Ends @ $BB7F. Execs @ $B9A1

The System definition utility provides a way for the user to keep the system informed of any changes to the RAM or disc configuration is use. TAN DOS 65 keeps a record of the amount of RAM in each page of memory and the number of tracks on each disc connected to the system. If the stored values do not correspond with what is actually present on your system then either your system performance will be reduced, or worse still, it may not work properly.

Back to Index