TANDOS 65

Contents

INTRODUCTION

TANDOS 65 is an extension to the Tanbug/Xbug ROM resident monitor for the Microtan system. It provides a friendly environment for the computer user who wishes to interface floppy discs with his system.

What it does

TANDOS 65 lets you manipulate files held on floppy disc with speed and simplicity. You can do everything on disc which used to be limited to cassette. One advantage is the enormous improvement in speed (a second or two to load a typical program) and reliability.

Another big advantage is in the vastly increased flexibility. Commands exist in TANDOS 65 for:-

Obtaining file firectories (list of filenames, length and status)

Copying files from disc to disc.

Renaming files

Protecting files from accidental erasure.

Deleting files.

Loading a file to memory.

Saving memory as a file.

and more besides.

The result of all this is that you can use your Microtan system more quickly, reliably and powerfully than before.

Chapter 1

Installation Notes

To install your TANDOS 65 disc system you should have at least the following:

1. Microtan system with system motherboard.
2. Disc controller card. This may have the GPIB option added if specified when ordering.
3. Disc drive/s.
4. Disc connecting cable.
5. TANDOS 65 distribution disc.
6. This manual (if you do not have this manual you probably won't be reading this bit anyway!).
7. Blank discs for copying purposes.

Assuming that you have the above items you should now perform the following steps.

a. Make sure your system is switched OFF.

b. Remove the Tanbug 3 Eprom from its holder on the disc controller card. Install it in place of your Tanbug Eprom on the Microtan board. Make sure that you insert it the correct way round.

c. Plug the disc controller card into one of the additional slots on the system motherboard. Do NOT use the DOS slot. The disc controller card is set up for 5¼ " drives. If your drives differ from this please contact Microtanic for the link positions.

d. Connect the disc drive cable to your drive/s as specified in the disc drive instruction sheet. Don't forget to fit a mains plug to your drives.

e. Plug the other end of the cable into the socket on the disc controller card. Note the polarising lug which prevents you plugging in the cable incorrectly.

f. Your disc system is now ready for use. Switch the power on and insert the TANDOS 65 distribution disc into drive 0 as illustrated below (if you only have one drive then that will be drive 0).

Chapter 2

Description and Terminology

TANDOS 65 is actually a collection of eight or so programs. Each program performs a particular type of action e.g.

List a directory

Delete a file

Almost all of these programs reside on disc, thus avoiding cluttering up an enormous chunk of memory. When needed, the routines in the ROM will load the appropriate program into memory and then execute it.

This process is entirely automatic and not something you need to worry about (or even know about!). It manifests itself as a short pause between typing a command and execution of that command (while the program is fetched from disc).

Disc Numbering

You may have up to four disc drives on your system. Any or all of these may be double-sided. So far as the hardware is concerned, a two sided disc is the same as two single sided discs. To make the system consistent, a numbering scheme for discs has been adopted which gives a unique number to each disc surface. Thus, the first side of the first disc is Ø, the second side of the first disc is 1 etc. up to the second side of the fourth disc which will be 7. If you have only single-sided discs then you will not use any of the odd numbers (a typical two drive system will contain units Ø and 2 only).

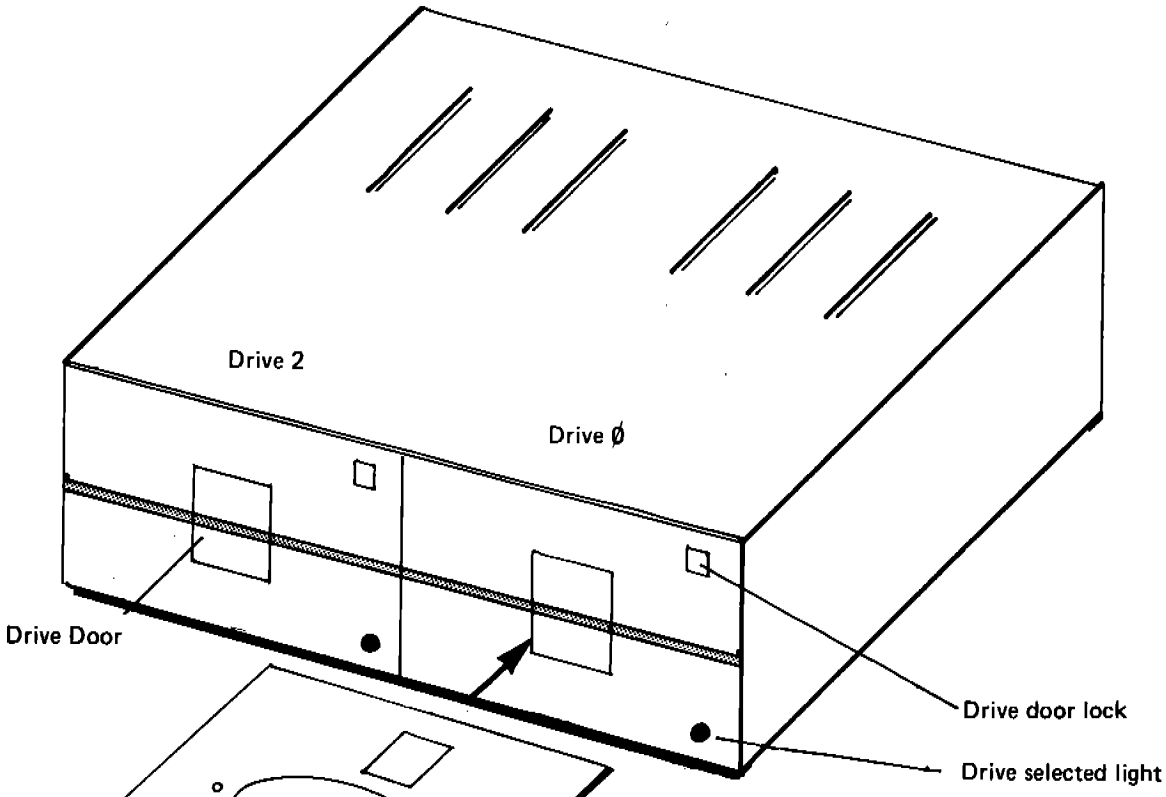


Fig 1: Correct positioning of disc for drive insertion.
Note label uppermost.

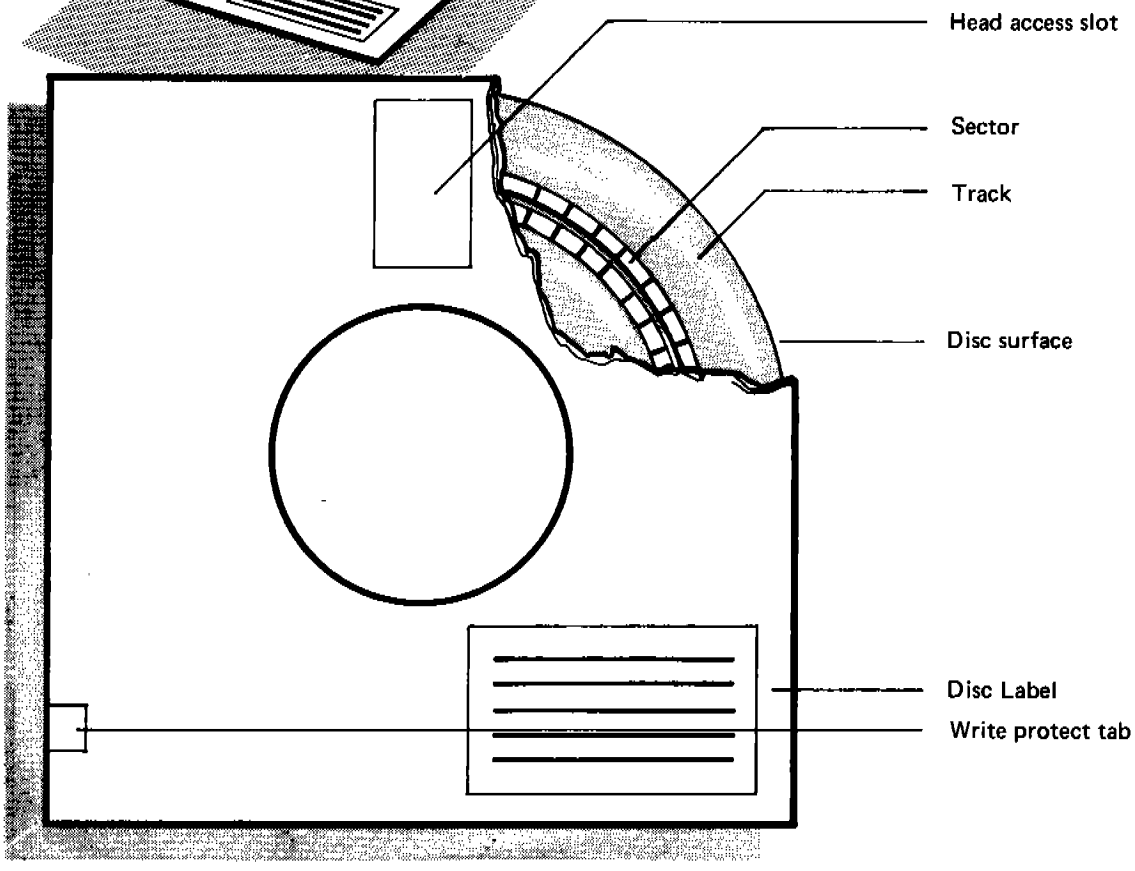


Fig 2: Main features of Floppy Disc.

Filenames.

Each file on a disc has a name. This name has up to 9 characters and consists of up to 6 characters of 'extension', e.g. FILNAM.EXT is a legal filename. So also is X . The dot is not part of the filename but is used when printing it to separate the name from the extension (otherwise you could not tell ABC from ABC but A.BC and AB.C are easily identifiable).

The characters used in making up the filename must be alphanumeric (i.e. A–Z, Ø–9 inclusive).

It is conventional, but by no means mandatory, to use the extension to indicate the general class of file, e.g. .TXT for a text file, bland extension for an executable program. .SRC for source file, .BAS for a Basic source file etc.

Wildcards.

This strange name, originating from the days of punched cards, is used to indicate a special character which replaces all others and is a sort of 'don't care' character. TANDOS 65 permits the use of wildcards in several commands and they considerably enhance the power of the system when operating on several files at once. TANDOS 65 has two wildcard characters, '?' and '*'. '?' replaces a single character, so referring to a filename JOE? .BAS could equally well refer to JOE.BAS, JOE1.BAS, JOEY.BAS etc. You can have any number of ? wildcards in a filename. FRED.? ? ? matches any file with name FRED regardless of the extension.

The '*' provides a Shorthand way of saying the same thing:
FRED.*.

Likewise, *.BAS refers to any file with the extension BAS. *.* refers to any filename at all.

Using wildcards with the TANDOS 65 commands simplifies such tasks as (we give only the file specification here):—

Copying all your Basic programs from one disc to another	*.BAS
Deleting all files beginning with XYZ	XYZ? ? ? .*
Obtaining a directory listing for all files having the character '5' in the 3rd position of the name and an extension of '123'	? ? 5 ? ? ? .123

Disc Units .

If you need to specify a particular disc because the file you want is there, then the complete specification for your file becomes:-

Unit:FILNAM.EXT

Unit is the disc number in the range Ø to 7 inclusive. Legal file specifications include:-

Ø:FRED.BAS
3:MYFILE.TAN
2:X

etc.

The colon is used to indicate that the preceding number is a disc unit and not part of the filename.

Directories.

Each disc is capable of holding many files (about 350 very short files for single–sided, single density 40 track discs). The operating system needs to know a certain amount of information about each of these files. In particular: what it is called (filename), where it is kept on the disc and how big it is. All this information is held in the directory. In fact the directory is nothing more than another file on the disc which just happens to contain information about every other file.

The operating system uses the directory in much the same way as you would use the index to a book. Using the directory, the system has rapid access to file information.

Formatting and Initialising

The disc hardware needs a basic pattern of information stored on a disc before it can do anything useful. This information largely consists of positional data so that the hardware can check where the head is currently positioned over the disc surface.

Brand new discs do not have this information recorded on them. The process of writing this information is called formatting. A TANDOS 65 utility is provided for this job. It is the formatting process that sets up the disc for single or double density and it also determines the size of each sector (always 256 bytes in TANDOS 65).

A sector is the smallest unit of information which may be stored on a disc. On a single-sided, single density, 40 track disc, TANDOS 65 expects 360 sectors each of 256 bytes. Double density operation would increase that figure to 640.

The 'information' content of a freshly formatted disc is entirely nonsense. TANDOS 65 needs the contents of the sectors to be organised in a particular way to ensure that the directory is 'empty' and that it will be able to allocate space to future files in the correct way. A TANDOS 65 command is provided for this process.

The System Sector

The very first sector on each disc (track Ø, sector 1) is reserved for special system information. This sector is referred to as the System Sector (SS).

The information held in the SS is in two parts: System related and disc related.

The system related information consists of details of discs connected to the system and the amount of memory in each page.

The disc information includes such things as: where the directory is stored, how much free space is available, where it is and the name of the disc. The disc name is a string of up to 9 alphanumeric characters, provided by the user during initialisation. It is useful in identifying a particular disc.

The System S Disc

The disc in Unit Ø will be treated as the system disc. The information relating to discs and memory available will be read from this disc only. TANDOS 65 expects the files relating to the various commands to be on this disc. The first TANDOS 65 command following a reset will read the system information from the system disc. It will not normally read it again unless reset is pressed (or the equivalent in software is performed e.g. GFCØØ from Tanbug).

If you change your system configuration and as a consequence update the system sector information using the SYS command then the new information is not used by the system until the computer is reset. This allows you to set up a system disc for configurations other than that currently in use.

The TANDOS 65 command 'SYS' lets you edit the system information on any disc. The disc initialisation command copies the system information for you. This means that freshly initialised discs are safe to use as system discs without any special action (though you will probably need to copy the various TANDOS 65 files relating to commands before you can do anything useful).

Disc Compatibility

TANDOS 65 is fully compatible with any mixture of up to four floppy disc drives. These drives may be double or single sided and have any number of tracks from 35 to 80 inclusive.

TANDOS 65 is fully compatible with both single and double density formats. Speed restrictions on Microtan however, limit revision 1 firmware to single density only. It is hoped to make a new ROM available which will permit double density operation.

Memory Requirements

TANDOS 65 assumes that at least 8K of RAM is available (i.e. Microtan and Tanex). It will operate correctly with any additional amount of memory up to the maximum 64K. It will however 'steal' 5K of memory at locations $A8ØØ to $BBFF for its own use (this is the top 5K of TANRAM).

The operating system uses a mixture of ROM and RAM at these locations. The necessary hardware is entirely contained on the DISC/GPIB board so that it is not necessary to purchase TANRAM (though the use of TANRAM will significantly improve your systems performance).

Chapter 3.

Getting Started.

Assuming that you have got the hardware sorted out in the manner described in the instructions in Chapter 1 and that you have checked some of the simpler instructions in Tanbug – just to make certain that everything is OK, then you can try some of the TANDOS 65 commands.

Your distribution disc will have come with a silver coloured tab over the write protection notch. This is to prevent accidental erasure of the files. Leave it on!

Insert your distribution disc in Unit 0 (see your disc drive manual on how to do this if you are unsure).

Press Reset

type the command DIR <CR>

The disc should start up and after a moment you should get the following display on your T.V. screen (teletype users will receive the same information – but more slowly!).

```
    O:MASTER      DIRECTORY PAGE 1
COPY              8P TEXT –       2P
    DIR           3P DEL  –       3P
    DSAVE         3P REM  –       2P
    SYS           3P INIL –       3P
    FORMAT        3P DEBASIC      1P
    SECIMP        2P

        33 USED. 325 FREE OUT OF 358
```

The display shows the name of all the files on your distribution disc. The numbers tell you how long the files are (in sectors). The letter P after the numbers shows that the file is protected from erasure by a 'don't delete' flag in the directory.

Now that you are sure that your system is working, it would be a good idea to follow the procedure outlined in 'creating a system disc' before trying out all the various commands.

Creating a System Disc from the Distribution Disc

The distribution disc supplied assumes that you have a minimum system i.e. 8K of memory and one single-sided drive. Regardless of how much memory or how many drives you really have, the supplied system cannot use other than the minimum. This means that the very first system disc you genrate may be a rather more tedious job than necessary.

The following actions must be executed in the order given. Wait for completion of each action before proceeding (some actions take several seconds).

Load distribution disc in Unit Ø. (Don't remove the silver tab).
Type FORMAT Ø:<CR>
When prompted to do so – exchange the distribution and new discs.
Wait for approximately 10 seconds until formatting completed.
Again load the distribution disc.
Type INIT Ø:<CR>
When prompted to do so – exchange discs as before.
Enter the required disc name and press CR .
Wait for approximately 40 seconds until initialisation completed.
Again load the distribution disc.
Type COPY <* C <CR> .
You will be prompted to change discs from time to time and the system will keep you informed of which files have been created.

When the copying process is completed, your new disc is ready to use. The system information stored on it will be the same as on the distribution disc. If your system is in any way different (more discs or memory) then use the SYS command. (see Section SYS). Store your distribution disc in a safe place – you may need it again.

You may generate more system discs from your own master system disc created from the distribution disc. If you have more than one disc drive, the process will be quicker and less prone to operator error, if you generate your new disc on a second disc drive (i.e. for disc 2 use FORMAT 2: ; INIT 2: ; COPY 2: < *C see relevant sections for full explanation).

```
FORMAT 2:
LOAD DISC 2 AND PRESS RETURN
FORMAT COMPLETE
INIT 2:
LOAD DISC 2 AND PRESS RETURN
DISC NAME: PAULKØ2
DISC INITIALISATION COMPLETE
```

```
COPY 2: < *.*
COPY     - CREATED
TEXT     - CREATED
DIR      - CREATED
DEL      - CREATED
DSAVE    - CREATED
REN      - CREATED
SYS      - CREATED
INIT     - CREATED
FORMAT   - CREATED
DBASIC   - CREATED
SECDMP   - CREATED
```

DIRectory

DIR <CR>	Full directory of current disc
DIR n: <CR>	Full directory of disc n
DIR n:FILNAM.EXT<CR>	Directory information for specified disc/file only

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.

Full Directory Listings

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

```
DIR
Ø:PAULKØ1          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 'Ø:PAULKØ1' 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 remainder of the display is fairly obvious in meaning. 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 hency 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 at least one for directories (for every 15 files on disc, TANDOS 65 uses 1 directory sector).

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. (Escape terminates the directory display).

If you are using a printer, you may find it useful to avoid the next page prompt. Terminate the DIR command by <LF> – no next page prompts will be given (DIR will still paginate the directory information).

Different Discs.

If you want a full directory listing of a disc other than the 'current' unit, modify the DIR command to DIR n: <CR> where n is any valid unit number from Ø to 7.

This functions exactly as before but gives you the information about disc unit n.

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 your 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 (see the section 'wildcards' in Chapter 2) to give you a powerful method of extracting the directory information you need.

DELete.

DEL FILNAM.EXT <CR>	Delete FILNAM.EXT from current drive.
DEL n:FILNAM.* <CR>	Delete all files with name 'FILNAM' from drive no.

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 RENname command.

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. If you want this particular file to be deleted then you must respond with Y <CR>.

Programmed Requests.

Advanced users may wish to know that it is possible to delete files from disc by issuing subroutine calls and incorporating some assembly language code in your program. (See Advanced Users Guide).

Directory Tidy Up

You may occasionally notice that disc activity and execution time for a DEL command is slightly extended. This is perfectly normal and occurs whenever file deletion releases sufficient space in the directory for a sector to be removed from the directory and returned to free space.

The directory is completely re—organised during this process so you may well find that files are listed in a different place on the directory display. This does not affect the files themselves. The directory tidy up is only executed after the DEL command is issued from the keyboard.

REName

REN FILNAM.EXT< FILNUM.EXT <CR>	Rename FILNUM as FILNAM
REN FILNAM.EXT P<CR>	Change attribute to Protected.

The Rename command is used for two different purposes. The main use is simply to change the name of a file.

The secondary use is to change the protection code of a file.

You may not rename a file which is write protected.

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.

SYStem

SYS <CR>	Runs the system definition Utility on current disc.
SYS n: <CR>	As above but on specified disc.

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.

TANDOS 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.

RAM pages

The Microtan and Tanex system supports a partially paged RAM structure. You may have up to 8 (numbered Ø to 7) pages, each of up to 42K bytes, of additional RAM in the form of TANRAM boards.

The minimum RAM available in any page is that set by Tanex. TANDOS 65 assumes a full complement of RAM in Tanex so that the minimum memory must be 8K bytes. Those pages (if any) where TANRAM boards are fitted, will have rather more. Typically, 24K, 40K, 41K, 42K depending on how well populated your TANRAM is.

SYS responds with the page and the number of Kbytes it believes to be available in that page If you wish to change this then type the new number of Kbytes (in decimal) and press return. Pressing return only, indicates 'no change'. Continue until all pages Ø – 7 have been presented for editing. SYS will then proceed to present information relating to discs.

An attempt to enter a silly amount of memory will be trapped out and the user re-prompted.

Please remember that the DOS 65 ROMs take up what was previously RAM space. These begin at $A800 which means your maximum RAN space will be 42K.

Disc Tracks

Discs may have various numbers of tracks from 35 to 80. Typically 40.

TANDOS 65 keeps a record of discs present by listing how many tracks are available on each possible disc unit (Ø to 7). A value of 0 inidates that the unit is not present. The actual number of tracks is only used by the INIT and FORMAT commands.

The SYS utility presents the number of tracks per disc in the same way for the memory pages. Again 〈CR〉 indicates no change and a decimal number followed by 〈CR〉 will update the entry. An attempt to enter 'Ø' as the number of tracks on drive Ø (which would indicate that drive Ø was not present) is not allowed. Drive Ø must always be present.

INITialise.

INIT Ø: Initialise a disc in drive Ø.

The INIT command is used to initialise the file and directory structure onto a disc. 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. You should 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 INITed disc will be the same as the current system disc. Run SYS if you need to change this.

Please 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 'escape' in response to the 'Disc Name' prompt.

DLOAD

DLOAD FILNAM.EXT 〈CR〉	Load file from disc – if file is capable of execution TANDOS 65 will jump into it on load completion of the load operation.
DLOAD n:FILNAM.EXT ABC 〈CR〉	Load from specified disc with parameters ABC.

DLOAD is the most general way of loading files into memory. It is the disc equivalent of the F,FILNAM command in Tanbug.

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 other than the file specification will be ignored by the load routines. Thus the loaded program may use that data for its own purposes (just like commands to 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 advanced Users guide 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 program you have written.

Programmed Requests

File loading is also available as a subroutine call. This permits the advanced user to make full use of overlays, program chaining etc.

The DLOAD (and auto–load facility) command is the only TANDOS 65 command which is entirely ROM resident. Other than variable usage and sector buffering it makes no use of the TANDOS 65 command which is entirely ROM resident. Other than variable usage and sector buffering it makes no use of the TANDOS 65 RAM area at $B8ØØ – $BBFF.

DSAVE

DSAVE FILNAM.EXT <CR> Save memory as file – not executable.
DSAVE n:FILNAM.EXT THHHH<CR> Save memory as executable file with transfer of control to HHHH.

DSAVE is the mechanism provided for saving areas of memory in disc files. It is the disc equivalent of the Tanbug command Dxxxx,xxxx,NAME.

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 Ø which DLOAD interprets as 'no transfer'.

Pn Page n, where n is a page number from Ø to 7. DSAVE will save the memory from page n. Default is page Ø. 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 SB8ØØ to SBBFF. 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 SB8ØØ–SBBFF to any convenient location and then DSAVE from there.

COPY

COPY DESTFN.EXT<SOURCE.EXT<CR>	Make a copy of SOURCE.EXT called DESTFN.EXT'

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 though there is only one disc drive available. Options to delete old versions of files (supercede), merge two files, change the write protection attribute etc. are also available. Extensive wildcard facilities increase the usefulness of the Copy command even further.

Copying files from one disc to another

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:<Ø: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: <Ø:*.* <CR>

will copy everything from drive Ø 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.

Superceding

It is often 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 superceded 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. No special indication is given of this status in the directory listing.

If the attribute is set to 'write protect' ('P') then the file cannot be deleted by any of the normal disc commands (beware INIT and FORMAT -- 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.

Merging Files together

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).

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 read 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 rubbish 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.

Single Disc Drive

COPY caters for operations which must 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 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 copies.

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) everytime 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.

Memory Usage

COPY uses memory from location S4ØØ 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.

FORMAT

```
   FORMAT Ø: <CR>
n: FORMAT 2: <CR>
```

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 TAN-FORTH 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 'return'. 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 'escape' 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 S4ØØ upwards.

DRIVE

DRV 2 <CR> Select drive 2 as current drive

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

DRV may be followed by any legal unit number (in the range Ø 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 Ø (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 Ø) 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 Ø and display a directory of disc 2.

BASIC – Using the TANDOS 65 BASIC adaptor

DBASIC ⟨CR⟩

Assuming that you already have the standard BASIC ROMs installed, then the enhanced BASIC, incorporating the various disc commands, may be invoked by using the DBASIC command. This loads and runs a short initialisation program from the system disc and then jumps into the existing BASIC ROMs.

Disc BASIC adds a further 6 'commands' to the ROM version. No commands or facilities are lost (yes, you can still save/load to cassette). These commands are made available to you because DBASIC patches a RAM resident subroutine in the zero page. Owners of the various proprietary 'Toolkit' packages available will not easily be able to use these with disc BASIC due to this patch. However, there are still ways of accessing 'Toolkit' and other similar ROMs, see section on 'linking to other ROMS'.

WARm Start

This Tanbug command works as described in the Tanbug manual but you are cautioned to avoid using TANDOS commands which cause significant disc activity (e.g. DIR, DSAVE etc) as these use locations which will corrupt the values needed to warm start BASIC. All the Tanbug single letter commands (e.g. I, L etc) are 'safe'.

RAM Usage

DBASIC requests 'memory size' in the usual way. If you opt to take the default by pressing CR then the system will determine the amount of memory available. Unlike the old BASIC, DBASIC extracts this information from the system disc when loading DBASIC. Subsequently, there is no delay and the default may be changed by using the TANDOs 65 SYS utility. If an error occurs when responding to 'memory size' then BASIC is reset to the ROM version.

The necessary buffers for disc I/o are positioned at the top of RAM. There are four 256 bytes buffers in total which means that you will effectively have 1K less memory when running DBASIC than when running BASIC.

New Facilities

There are three new commands and seven new statements available in DBASIC. They are (with example filenames):

DSAVE	"n:FRED"	⟨CR⟩	
DLOAD	"n:JOE"	⟨CR⟩	
DELFNP	"n:PROG"		
DELFND	"n:DATA"		
DOPENR	"n:DATAIN"		program mode only
DOPENW	"n:DATOUT		
DCLOSER			
DCLOSEW			
DINPUT			
DPRINT			

All filename specifications must be expressed as a valid BASIC string, either by enclosing the filenames in quotes or by specifying a string variable containing the filename e.g.

```
A$="2:FRED"
DSAVE A$
```

will work correctly. String expressions may also be used.

DSAVE

This command is used to save the entire BASIC program as a disc file. It requires a disc filename specification (and optional drive number) expressed in the usual format **except** that the extension is given by the system and will always be '.BAS'. you will get a syntax error if you try to specify your own file extension. DSAVE may also be used from within a program.

A typical example would be:

```
DSAVE "2:MYFILE" ⟨CR⟩
```

DLOAD

This command is the complement of the DSAVE command and is used to load a new program into memory from disc. Like DSAVE it needs a file specification in order to know which file to load.

DLOAD may also be used from within a program. e.g. DLOAD "0:PIRATE" ⟨CR⟩

DOPENx

There are two versions of the DOPENx statement – DOPENR and DOPENW: The DOPENx statements are used to indicate to the system which files will be read from/written to by the DINPUT and DPRINT statements.

e.g.

```
DOPEN "2:ADDRESS" or A$="2:ADDRESS"
                     DOPENR A$
```

indicates that DINPUT statements should read the contents of the file 2:ADDRES,BDT

and DOPENW "Ø:EXAMPL" or A$="Ø:EXAMPL"
DOPENW A$

will cause DPRINT statements to write to the file Ø:EXAMPL.BDT.

As with DLOAD and DSAVE, only the filename, not the extension, is needed. The system supplies the extension '.BDT' (BASIC DaTa) for you. Any attempt to specify an extension will result in a syntax error.

DCLOSEx

Again there are two versions of this statement – DCLOSER and DCLOSEW. They are the complements of DOPENX statements. The correct use of DCLOSE is especially important for files being written by the DPRINT statement as the DCLOSEW causes the system to write out any remaining data left unwritten in the output buffer and then tidies up the disc file and entres the relevant information into the disc directory. The effect of failing to issue the DCLOSEW will be that all the data written to the file will be lost and the file will not exist on the disc (this does not harm the other information on the disc). DCLOSEx will return successfully even if a file has not been opened. It is not possible to issue a second DOPENR or DOPENW until the appropriate DCLOSEx has been issued. The effect of this is that there can be only two disc files open simultaneously; one for input and one for output.

On input you can effectively 'rewind' a file back to the beginning by closing and then re-opening the same file. DOPENR, DCLOSER and DOPENW do not write to the disc. DCLOSEW will only write to the disc if a DRINT statement has been issued since the DOPENW (it is not possible to generate empty files). Both of the DOPENx statements read from disc.

DELFNx

There are two versions of DELFN, DELFNP and DELFND. These are used to delete Programs or Data files created by BASIC (in fact any file having the '.BAS' or '.BDT' extension).

e.g.

```
DELFND "2:DATA" or A$="2:DATA"
                    DELFND A$
DELFNP "2:PROG" of B$="2:PROG"
                    DELFNP B$
```

An open file is automatically closed before being deleted. DELFNx may be issued in immediate mode or may be included in a program.

DINPUT and DPRINT

These two statements operate in exactly the same way as INPUT and PRINT with the sole exception that I/O is with the disc and not the keyboard/VDU. Reference should be made to the BASIC manual for more information. Note that the same requirements regarding separators (commas) apply to disc I/O as they do to cassette I/O.

As for cassette I/O only sequential files are currently supported by TANDOS 65 V1.Ø.

The DPRINT statement makes use of the user programmable output facility (Tanbug V2 onwards) whereby the address of a user provided output routine is placed in the slow interrupt vector. Consequently, slow interrupts and user programmed output are not available from Disc BASIC.

Example Programs

The first program generates a file with a single column of numbers from 1 to 10.

```
1Ø    DOPENW "Ø:EXØ1"
2Ø    FOR I = 1 to 1Ø
3Ø    DPRINT I
4Ø    NEXT I
5Ø    DCLOSEW
```

If you want to see the numbers as they are written to disc then insert the line 35 PRINT 1.

This second example reads the column of numbers from the first program and adds a second column of numbers equal to 1ØØ times the first column. This is intended to demonstrate, in very simple terms, the common 'input - process - output' type of application.

```
NEW
1Ø    DOPENR "Ø:EXØ1"
2Ø    DOPENW "Ø:EXØ2"
3Ø    FOR I = 1 to 1Ø
4Ø    DINPUT J
5Ø    DPRINT J,",",1ØØ*)
6Ø    NEXT I
7Ø    DCLOSEW
8Ø    DCLOSER
```

Adding line 45 PRINT J will display the values read from file.

This last program reads the second data file we generated (with two columns of data) and displays them on the screen.

```
NEW
1Ø    DOPENR "Ø:EXØ2"
2Ø    FOR I = 1 to 1Ø
3Ø    DINPUT J,K
4Ø    PRINT J,K
5Ø    NEXT I
6Ø    DCLOSER
```

Use of machine code subroutines with DBASIC

If you use machine code subroutines with ROM BASIC, they are normally put at the top of RAM, and 'MEMORY SIZE' entered appropriately. The same applies with DBASIC, but you will find that the available memory is 1K less than you specified, due to space taken by the disc buffers.

Linking to other ROMs

DBASIC is linked to ROM BASIC in the same way as for example, Toolkit and the Hi-Res graphics ROMs. This makes it tedious to use these ROMs with DBASIC, as additional code is required to reset the relevant locations in memory. These locations are 235 to 249 (HEB to HF9). The problem is that you cannot write BASIC code to reset this area of memory as it is almost continuously executed by BASIC. The solution is to write a machine code subroutine to do it for you.

To find out what you need to put in these locations for, say, the Hi-Res graphics ROM, this is what you do (assuming the ROM is installed!).

1. Run normal BASIC and follow the normal procedure to make available the Hi—Res commands.
2. Using a FOR. . NEXT loop and the PEEK instruction, note down the contents of location 235 to 249.
3. Write a machine code subroutine which loads the values you noted into their respective locations.
4. Run DBASIC.
5. Repeat steps 2 and 3 to produce a second subroutine.
6. Run DBASIC again.
7. Use the USR(X) statement to call your first subroutine. You can now access the Hi—Res graphics.
8. Use the USR(X) statement to call your second sub-routine. You can now use the disc commands.
9. Alternate between steps 7 and 8 as appropriate.

Chapter 5

Advanced User's Guide

The information contained in this section of the manual is intended for those users who wish to delve more deeply into the internal mysteries of TANDOS 65.

This section is aimed at users with a reasonable background of computer 'know—how' and is not intended as a tutorial text.

Disc Organisation and File Structures

System Sector

Track Ø, Sector 1 is reserved for system use on all TANDOS 65 discs. It is referred to as the 'system sector'.

Bytes 0—7	'DSCDEF' These 8 bytes define the number of tracks available on each of the 8 disc drives permissible. Byte Ø refers to disc Ø etc. A value of H28 indicates a 40 track disc drive. 'DSCDEF' is loaded into RAM from the system disc (drive Ø) and is used to determine which devices are available. (Ø tracks indicate no drive).
Bytes 8—15 (H8—HF)	'PAGDEF' These 8 bytes define the maximum amount of memory available on each TANRAM page. Byte 8 refers to TANRAM page Ø etc. The value contained in the byte is the high byte of the maximum address on the page i.e. if the maximum address is HA1FF then the relevant byte in PAGDEF will contain HA1. The least significant byte is always assumed to be HFF.
Bytes 16—17 (H10—H11)	Free Space Pointer Byte 16 contains the sector number and Byte 17 the track number of the start of the free space chain. On a freshly INITialised disc, this will be track Ø, sector 7. This pointer will be updated whenever sectors are allocated to files or directory use, and when files are deleted.
Bytes 18—19 (H12—H13)	Directory Pointer Byte 18 contains the sector number and Byte 19 the track number of the first sector of the disc directory. This will usually be track 0, sector 4 but may point to anywhere on the disc (particularly after a long period of heavy use creating and deleting files). A sector number of Ø Which is an illegal number, as far as the disc hardware is concerned) indicates that there is no directory on the disc.

Bytes 20–21 (H14–H15)	Free Space Count These bytes together (byte 2Ø least significant) contain the total number of sectors remaining unallocated. This number will be the total number of sectors on the disc minus two on a freshly INITialised disc and is updated whenever sectors are allocated to files or directory.
Bytes 22–23 (H16–H17)	Used Space Count These bytes together (Byte 22 least significant) contain the total number of sectors allocated to files on the disc. The number will be Ø on a freshly INITialised disc and is updated whenever sectors are allocated to file usage. The count is not changed when sectors are allocated to directory use.
Bytes 24–32 (H17–1F)	Disc Name This is a nine character ASCII string which is used to store an arbitrary disc name. This name is specified by the user during INITialisation. It is used by DIRectory and COPY.
Bytes 33–63 (H2Ø–H3F)	Spare.
Bytes 64- 95 (H4Ø–H5F)	Version Identifier This area is reserved for an ASCII string identifying the particular version of TANDOS 65 used to create the disc. It is written by INITialise.
Bytes 96–255 (H6Ø–HFF)	Spare.

Directory Sectors

The disc directory consists of a number of directory sectors (often only 1). The first directory sector is pointed to by the directory pointer in the system sector. Two bytes in each directory sector are reserved as pointers to the next directory sector. The last directory sector is indicated by a Ø in the next sector byte. By this means the directory maybe as long, or as short as required.

Each directory sector may hold up to 15 file entries.

Each directory sector is organised into three parts:–

Next directory Sector Pointer	Bytes Ø,1 these point to the next directory sector. Byte Ø is the track and byte 1 is the actual sector. A value of Ø in byte 1 indicates that the current directory sector is also the last. The remaining 7 bits are undefined but care should be taken to leave them unchanged when modifying the Write Protect bit to preserve compatibility with future systems.

Free Space

Unused sectors on the disc are linked together to form a 'free–space chain'. Thus, the first free sector is pointed to by the 'free space pointer' in the system sector. The first sector points to the second: the second to the third, and so on.

This organisation is created initially by the INITialise command. During this process, all unused sectors are filled with zeroes (except the two bytes used as pointers).

INITialise creates the free space chain in such a way that the pointers never point to an adjacent sector but to a sector about 1/3rd of a disc revolution 'later'. This considerably reduces the time to access files as there is just sufficient time to process each sector and request the next before that sector appears under the head. If the sectors were adjacent on the disc, then the 'next' sector would have gone past too quickly and the system would have to wait for an entire disc revolution.

This optimum arrangement is gradually eroded as a disc has files created and deleted in a random way.

During normal TANDOS 65 operations, sectors are used for files and then, perhaps, returned to the free space chain. Under these circumstances the sector contents will be undefined.

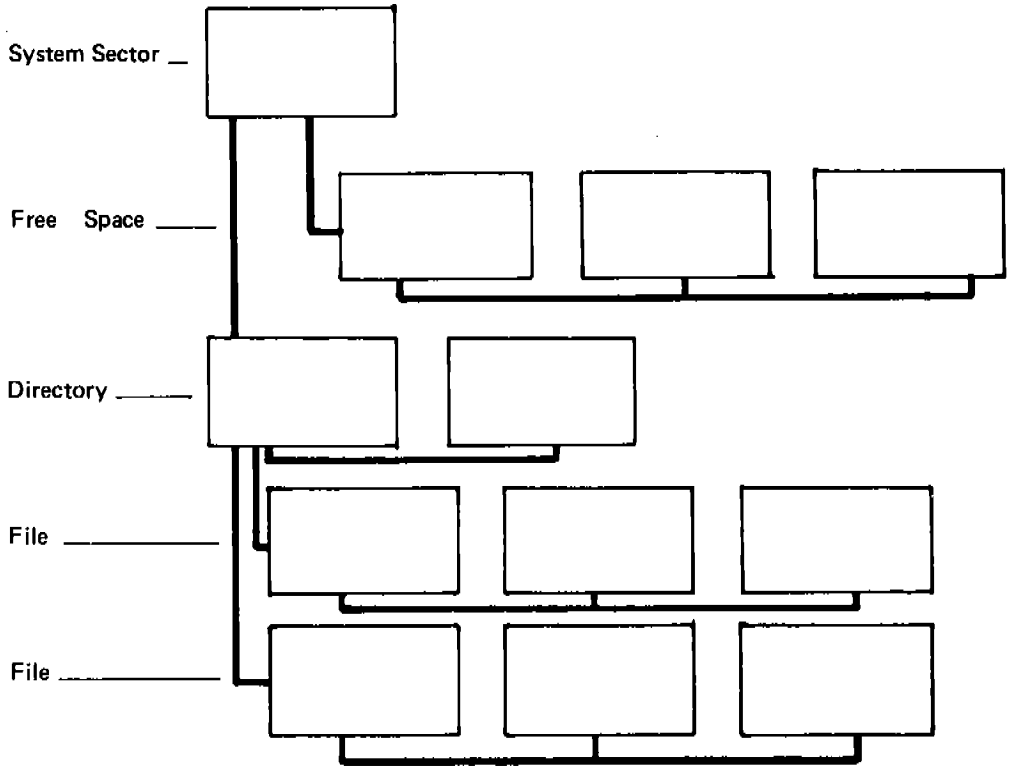
The pointers in each sector of free space are contained in Bytes Ø,1. Byte Ø is the track and Byte 1, the sector number of the next sector in free space. A value of Ø in Byte 1 indicates that the current sector is also the last sector.

File Organisation

The sectors which form an individual file are linked together into a chain (in a similar way to the free-space). The first sector of a file is pointed to by information contained in the File Information Block in its directory entry. The first sector contains information which points to the second sector and so on.

The pointers are contained in Bytes ∅ and 1 of each file sector. Byte ∅ is the track and Byte 1 the sector number of the next sector in the file. A value of ∅ in Byte 1 indicates that the current sector is also the last sector of the file.

Disc Structure



This diagram shown how the various data structures interact to complete the disc file structure.

The System Sector points to both the Free Space chain and the Directory. The Free Space maybe as long or short as in appropriate (to suit the disc size and the number of sectors already used). The Directory points to each file (both start and end in fact, although only start shown above – this makes deleting a file a more straightforward and reliable task). Each file may be as long or short as necessary and available space permits.

File Structure

All files created under TANDOS 65 conform to the same structure. Essentially, this is record oriented, with a maximum record length of 255 (HFF) bytes. In general, records may cross sector boundaries (i.e. a particular record may start in one sector and end in another).

Files intended to be loaded using DLOAD or the autoload derivative of that command may not contain records which cross sector boundaries. however. Certain special records containing address information are also expected by DLOAD.

The record structure for all files is very simple:

> Byte Ø of the record contains the record length (excludes the record length byte itself – thus a record of 1Ø bytes would have a 1Ø in byte Ø of the record).

The remainder of the record consists entirely of data.

Records of Ø length are permissible – these consist simply of the record length byte (of value Ø, of course). Such records are treated as padding and are never passed to the user. (The user may generate them but they will be skipped when reading them back). Zero length records are used to fill sectors at the end of load modules (created by DSAVE).

Reading and writing of records is supported by the I/O sub-routines in the Disc ROM.

Load Modules

Load modules are files, usually created by the DSAVE command, which are intended to be loaded into memory.

The first record in the file is always an address record. Address records in load modules are non-standard in that the record length byte is HFF but the record is only seven bytes long. Further address records may occur anywhere in a load module. (hence the requirement for some special indicator).

Address Records

Byte Ø	Record length – always HFF
Byte 1	Page – TANRAM Page number to which the load module should be loaded.
Bytes 2–3	Start Address – the address of the first byte of data in the load module. Byte 2 is least significant.
Bytes 4–5	End Address – the address of the last byte of data in the load module. Byte 4 is least significant.
Bytes 6–7	Transfer Address – the address to which control should be transferred on completion of loading. Byte 6 is least significant. If both bytes 6 and 7 are zero then control will not be transferred to the load module but will continue with TANBUG on completion of loading.

The data from subsequent records is loaded at the start address upwards. until either the data is exhausted or a new address record is encountered.

DLOAD will not load files correctly with records which cross sector boundaries.

DSAVE generates load modules with no more than one data record per sector.

Text Files

TANDOS 65 standard files will adhere to the following format:

Record structure compatible with the I/O routines in the DOS ROM.

Space compression will not be used.
All control characters will be included in the record.
Lines will normally end with carriage–return.

SECDMP

n:SECDMP ⟨CR⟩

Secdmp (SECtor DuMP) was written as a purely diagnostic/debugging tool for use during the development of TANDOS 65. Its utility has so far exceeded expectations, however, that we are including it 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 remember). For this reason we have placed the instructions for using SECDMP in the 'Advanced Users Guide'. You have been warned!

SECDMP can only be used with the Microtan VDU screen as the output device. It loads to H4ØØ upwards.

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 carriage return.

X — Return to Tanbug — the screen is cleared on exit.

Un — Select disc unit n. n must be in the range of Ø–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 to 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. LØ will locate the cursor at the start of the buffer. LFF will locate it at the end as will LØ1FF (the Ø1 is ignored since SECDMP will only look at the last two characters – FF).

Eascii string — Enter. This command enables ASC11 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.

RAM USAGE

TANDOS 65 makes extensive use of RAM at B800 to BBFF. this 1K of RAM is physically located on the disc board and is mapped to overlap the top 1K of all TANRAM pages. The disc board itself ignores the TANRAM page structure. This area, referred to a the 'overlay' area is used both for storage of variables and as a 'safe' area to which TANDOS 65 programs (such as DIR, DSAVE etc) may be loaded without conflict with the user's memory.

You are cautioned to be very careful if you choose to use this RAM area for your programs. Unexpected and potentially catastrophic problems may result from altering values held in the overlay area.

The following table outlines the address, name and purpose of each byte of overlay RAM used by TANDOS 65.

Name	Hex Locations	
UNIT	B8ØØ	Disc drive no. for sector read/write
TRACK	B8Ø1	Disc drack no. for sector read/write
SECT	B8Ø2	Disc sector no. for sector read/write
DENS	B8Ø3	Disc density flag must be '1' (single density)
DMAA	B8Ø4–B8Ø5	Address of sector buffer B8Ø4 least significant.
PAGE	B8Ø6	Memory Page relating to DMAA.
ERROR	B8Ø8	Contains error code after disc command.
CURDRV	B8ØE	Contains no. of 'current drive'.
	B8Ø7,B8Ø9–B8ØD	
	B8ØF–B814	Used by Disc primitive operations.
DSCDEF	B815–B81C	Contains copy of DSCDEF from system sector.
PAGDEF	B81D–B824	Contains copy of PAGDEF from system sector.
SBUFØ	B825–B924	Sector buffer.

This section is used to hold a copy of parts of a disc system sector during file operations.

FSPSØ	B925	Free space pointer
FSPTØ	B926	
DRPSØ	B927	Directory pointer
DRPTØ	B928	
FSCLØ	B929	Free space count
FSCHØ	B92A	
USCLØ	B92B	Used space count
USCHØ	B92C	

File specification buffer.

UNITØ	B92D
FNAMØ	B92E–B933
EXTØ	B934–B936

Buffer used for directory operations.

FLENØ	B937–B398	File length
FSTRTØ	B939–B93A	Start pointer
FENDØ	B93B–B93C	End pointer
FATTØ	B93D	Attribute
PAGØ	B93E	Temporary store for a memory page number.
FDIRS	B93F	Used to hold pointer to a current directory file information
FDIRT	B94Ø	block – sector,
FDIRØ		track and offset from start of sector.
SAVSTK	B942	Used to save stack pointer.
	B943–B94E	Temporary variables.
ERREX	B94F–B95Ø	Error jump location.
	B96Ø–BBFB	'Overlay' area – used to load TANDOS 65 programs.
WBUF	BBFC	Used to store pointers to record I/O
WREC	BBFD	buffers.
RBUF	BBFE	
RREC	BBFF	

ROM USAGE

TANDOS 65 uses 4K of ROM located between HA8ØØ and HBBFF. This ROM is physically located on the disc board and is mapped to overlay 4K near the top of TANRAM. Since the disc board ignores the TANRAM pages, the ROM is available at all times.

The top 2K of ROM is concerned with the TANDOS 65 proper. Approximately one quarter of the area is taken up by routines which interface with the disc hardware. The remainder implements the DLOAD command, the autoload version of DLOAD and various common subroutines needed throughout the TANDOS 65.

The lower 2K of ROM implements the disc BASIC extensions and the general record input/output routines.

All the ROM routines to which the user may need access are 'vectored' i.e. the subroutines start with JMP SUBR and all the JMP instructions are grouped together. The location of the JMP instructions will not normally vary between different versions of TANDOS 65.

ROM Resident Routines

The following table lists all the routines accessible to the user and their jump vector locations.

Name	Vector Addresses (Hex)	Purpose
DLOAD	BØØØ	Entry point for DLOAD command.
AUTOLOAD	BØØ3	Entry point for autoload command.
WRITES	BØØ6	Write sector primitive
READS	BØØ9	Read sector primitive.
RESET	BØØC	Perform disc restore on driveno. in UNIT.
FDMAIN	BØØF	Main disc primitive.
FDSUB	BØ12	Low level disc primitive.
SETINT	BØ15	Primitive to start disc motor etc.
LDDFLT	B7AF	Set up default DLOAD parameters.
ESCAPE	B7B2	Restore Zpage & return to TANBUG.
ZSAVE	B7B5	Save Zpage (40.41) and save stack pointer.
LOADFL	B7B8	Load a file.
INIGDR	B7BB	Find first directory entry for specified file spec.
NFILE	B7BE	'No File' message.
PFILNM	B7C1	Print a file name (from dir. file info block).
GETLIN	B7C4	Get a command line.
ERRRET	B7C7	Jump to error handler.
QUERY	B7CA	Print '?' followed by CR.
WRTSCT	B7CD	Write a sector and check for errors.
RDSCT	B7DØ	Read a sector and check for errors.
GETSSC	B7D3	Read a system sector into sector buffer.
BADDEV	B7D6	Check UNIT for legal drive no. - error message if not.
OUTSTR	B7D9	General string output.
GD1	B7DC	Continue a directory search.
GDSCDT	B7DF	Get and save disc system sector information.
UPDSYS	B7E2	Update system sector from data in memory.
ENDIRB	B7E5	Enter a file information block into a dir.sector.
GFREE	B7E8	Get a free sector for file use.
GDFR	B7EB	Get a free sector for directory use.
GFRDIR	B7EE	Find a free directory entry.
GDIREC	B7F1	Find a particular directory entry.
GETSYS	B7F4	Get system info from system disc.
GETFIL	B7F7	Get a file name from a command line.
ALPNUM	B7FA	Check character is alphanumeric.
GEDIRB	B7FD	Save pointers to a directory entry.

Delete File Routine

This subroutine is vectored through a jump at HA 8Ø3 It will delete the file defined by the file Information Block pointed to by FDIRØ in the directory sector held in SBUFØ The routine uses both X and Y registers, and the accumulator.

If a disc error occurs, an error message is generated and a jump through ERREX (HB94F,HB95Ø) is executed. This will normally return to TANBUG after printing \CR\? \CR\ . The user may alter these locations to direct errors to a different handling routine.

The following code segment demonstrates how the delete file routine may be used with other ROM resident subroutines to perform the delete file function in its entirety.

Assume that filename is correctly loaded to UNITØ, FNAMØ, EXTØ (See OPENR for more details).

```
(subroutine or
 location address)

(B7F4)      JSR  GETSYS          ; Get system sector information
                                   from system disc.
(B92D)      LDA UNITØ
(B8ØØ).     STA UNIT             ; Transfer drive number ready
                                   for disc primitives.
(B7Ø6)      JSR BADDEV           ; Test unit number for legality.
(B7BB)      JSR INIGDR           ; Find relevant directory entry.
            CPX#Ø                ; Returns X=Ø if not found.
            BEQ ERROR            ; Deal with errors.
(SBUFØ)=
 B825)      LDA SBUFØ+15,X       ; Test the file attribute.
            BMI ERROR            ; Error because file has P attribute set.
(A8Ø3)      JSR DELFIL           ; Delete the file.
```

Auto—loading your own Programs

Using the auto-load facility, you may pass information to your program as a command line. This facility, used by the TANDOS 65 utilities, provides an efficient method of communicating with your programs.

On transfer of control of your program, the accumulator and X register contents are undefined. The Y register, used as an offset from the start of the command line, points to the first character following the name of the file just auto—loaded. This character can be retrieved using the statement:

```
LDA (ICURS),Y
```

where ICURS, the cursor index, is at address HA. TANBUG maintains this cursor index such that it always points to the start of the current line. To get the next character in the command, simply increment Y and repeat. It is obvious here that your command must all fit on one line.

The cursor is turned off by the auto—load process, but the cursor position can still be found as it is replaced by a space with M.S.B. set.

Disc Input/Output Routines

Seven routines are provided to support record structured disc I/O. All the routines are ROM resident but make use of some locations in the disc RAM area and up to 1K of RAM elsewhere (provided by the user).

Each routine is a subroutine and is vectored. Errors during execution all cause an indirect jump through ERREX (B94F, B95Ø). TANDOS 65 sets up this location to print '?' CR and an exit to TANBUG. The user may alter ERREX so that errors will be directed to the user's error handling routine.

The following describes the use of each of the I/O routines:

INIIO (HA8Ø3)

This routine must be called once only before using any of the other I/O routines. It may also be called safely only when all files are closed.

OPENR (HA8Ø6)

This opens a file to permit reading. The file specification must be given in UNITØ, FNAMØ and FEXTØ (See RAM usage).

UNITØ	(HB92D)	A single byte containing the drive number for the required file.
FNAMØ	(HB92E—HB933)	Six bytes containing the ASCII characters defining the file name. Names of less than 6 characters must be left justified and have unused bytes filled with space (H2Ø) characters.
FEXTØ	(HB934—HB936)	Three bytes containing the file extension held in a similar way to FNAMØ.

In addition to the file specification, the address of a sector buffer of 256 (H1ØØ) bytes must be defined. This is done by storing the most significant 8 bits of the address of the buffer in RBUF (HBBFE). The least significant byte of the address is assumed to be Ø. Thus for a buffer at location H1FØØ, RBUF must contain H1F.

OPENR checks that the file exists and sets up internal variables for subsequent RECIN operations.

OPENW (HA8Ø9)

This opens a file to permit writing. Its operation is very similar to OPENR and the file specification must be set up in the same way.

A sector buffer is also needed for write operations. The high byte of the address should be written to WBUF (HBBFC). The sector buffers used for read and write must be different if files are to be opened for read and write at

RECIN (HA812)

This reads a single record from the file currently open for read. The record is placed in the read record buffer.

The address of this 256 byte buffer must first be defined by writing the high byte of the address to RREC (HBBFF). On completion, the length of the record will be indicated by the contents of the first location of the record buffer (relative address Ø). This will be followed by that many bytes of data. The contents of the remainder of the buffer (if any) is undefined.

RECOUT (HA815)

This writes a single record to the file currently open for write. The record is copied from a write record buffer.

Again the address of this 256 byte buffer must first be defined by writing the high byte of its address to WREC (HBBFD). The format of the record to be output should be in the same format as records supplied on input i.e. the first byte of the buffer should contain the length of the record and the data should follow immediately.

The two record buffers (read and write) should be different if files are to be open both for read and write at the same time.

The addresses (in RREC and WREC) used for the buffers may be altered between each call to RECIN and RECOUT (unlike those used for the sector buffers held in RBUF and WBUF).

CLOSER (HA8ØC)

This closes the file previously opened for read.

No disc activity is forced by CLOSER.

CLOSEW (HA8ØF)

This closes the file previously opened for write.

Unlike CLOSER, CLOSEW provokes a considerable amount of disc activity. Firstly, any data as yet not written to disc but stored in WBUF (this is a function of the I/O routines and not something the user need be aware of) is written to the disc file. Secondly, the directory entry for the newly created file is generated and finally the system sector for the relevant disc is updated to reflect the current state.

Thus, so far as TANDOS 65 is concerned a file opened for write **does not exist** until CLOSEW is executed.

Appendix A

TANBUG 3 Command Quick Reference Chart

Command	Function
RESET	Initialise system and display TANBUG message
Mxxxx	Modify memory at hexadecimal address xxxx
lf	Step up through memory
esc	Step down through memory
space	Re–open currently displayed memory location
cr	Close currently displayed memory location
Lxxxx,y	List y lines of memory starting at address xxx
Gxxxx	Go, begin program execution at address xxxx
R	Display pseudo processor register locations
S	Set single step mode
N	Return to normal mode (Clear single step)
P	Proceed, past breakpoint or next instruction (singlestep)
Bxxxx,y	Set breakpoint number y at location xxxx
B	Clear breakpoints
Oxxxx,yyyy	Calculate offset between address xxxx and yyyy for branch
Cxxxx,yyyy,zzzz	copy block from xxxx to yyyy to location starting zzzz
Cntl P	Switch on/off parallel printer
Cntl V	Switch on/off serial printer
Cntl S	Switch on/off screen
Cntl L	Clear screen (in BASIC)
BAS	Enter BASIC
WAR	BASIC warm start
AUX	Jump to auxiliary command handling at SE800. May be used to add your own commands to Tanbug.
WFOR	Warm start for Tan–Forth. Generates jump to S4Ø4.