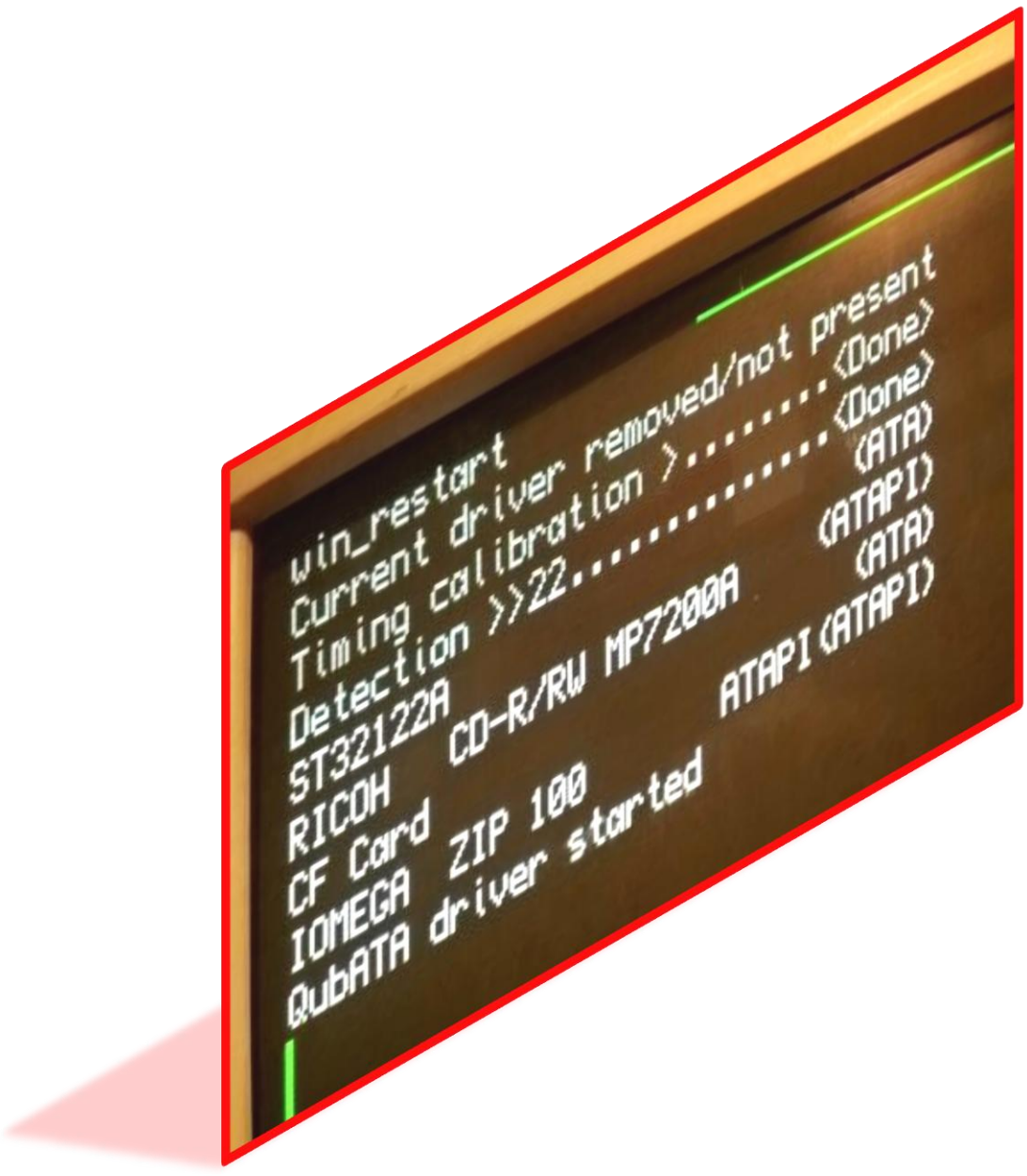
QUBATA DRIVER FOR TETROID

Manual (special edition) – Alain HAOUI



CONTENTS

INTRODUCTION 3

REQUIREMENTS AND COMPATIBILITY 4

INSTALLATION AND CONFIGURATION 4

STARTING 5

MANAGING DRIVER SECTION 6

WIN_REMOVE Remove driver 6

WIN_ RESTART Restart driver 6

BASIC COMMANDS SECTION 7

WIN_DRIVE Load partition 7

WIN_DRIVE Unload partition(s) 7

WIN_ CTRL Set drive options 8

WIN_ CTRL% Get current drive options 8

WIN_ VER$ Get driver version 9

WIN_ USE Change driver use name 9

WIN_CDEXT Extend CDROM commands 9

MAKE_ DIR Make directory 10

FMAKE_ DIR Make directory 10

PARTITIONS AND FORMAT SECTION 11

WIN_ FORMAT Set Format parameters 11

WIN_ DISK Manage partitions 13

Typical sequences 14

DIRECT RAW ACCESS & ALIEN SECTION 16

TRASHCAN MANAGEMENT SECTION 19

TECHNICAL NOTES AND LIMITS 20

TROUBLESHOOTING 22

FAT check summing 22

Other QubIDE utilities 22

CREDITS AND PERMISSIONS 23

MANUAL REVISION HISTORY 23

INTRODUCTION

This manual is written for the release of QubATA driver version 3.07 (T) for the Tetroid interface.

QubATA is a worth replacement driver of the original QubIDE ROM written by P.A.Borman.

This manual covers all software features (old and new) for driver and commands and comes in complement to any specific hardware supplied documentation.

This new driver was initially started to improve detecting and handling multiple devices on Expander hardware add-on but at final the almost entire driver was rewritten or optimized and some new features introduced. New code was arranged to get maintenance easier (at least possible) and enough slim to fit into the limited ROM space.

Major improvements with QubATA driver:

- Very fast and reliable devices detection with efficient time calibration on the target system
- Full Expander support for up to 16x2 devices (not very realistic but it is enough good for 4 devices, per example) – Need a hardware add-on with original QubIDE controller.
- Full compliance with ATA/ATAPI-4 protocol and removable medias support
- Fully functional Trashcan and associated utilities
- Enhanced direct raw access and “Alien” devices support
- Integrated commands for managing devices, partitions and format without any need to any extra tool utility
- CD Data support and Audio playing set commands (mimics of QPC CD-Audio module) on ATAPI supported hardware.
- Optimized and working slave blocks buffering with various sizes (up to 64 sectors per block) without side effects
- Full unsigned arithmetic permitting to handle big partitions/files up to 2GBytes (upper admissible limit for QDOS FS with this driver).

REQUIREMENTS AND COMPATIBILITY

This driver is suitable for the original QubIDE Card Controller on QL systems and qualified “QubIDE Clones” as the Tetroid Interface using CF Card memories as disk Media.

To work properly, hardware interface must be fitted with the last GALs V2 codes and this is mandatory.

This driver still compatible with QubIDE partition format from V2.xx (may be from V1.56) and may be used on existing QubIDE disks and partitions (and vice versa). However, it would be better to initialize disks and make new partitions with the QubATA driver after having ensured it is working on your system.

All original commands are supported with this new driver. Some parameters were extended but still compatible with originals. Some other commands are new. All commands (old and new) are described in this manual.

Some utilities supplied with original QubIDE shall still work but not all (see troubleshooting at the end of document).

INSTALLATION AND CONFIGURATION

This driver is supplied in two binary forms : one as ROM content replacement and the other as loadable RAM extension.

ROM version must be burned on the interface EPROM as required. It should not be executed from file with any pseudo ROM emulation command (E.g. SOFT_EPROM).

RAM version can be started safely over any existent ROM or other RAM version as per example:

LRESPR *flp1_QubATA_REXT* : REMark load and start new driver with TK2 present

***base*=RESPR(16128) : LBYTES *flp1_QubATA_REXT,base* : CALL *base :* REMark old way**

Before using the RAM version, the “Config Block” included in the REXT file must be overwritten to set the correct address of the interface according to jumpers configuration used when plugged in your system. This can be done with CONFIG or MENUCONFIG standard utilities.

Only one item [IDE base address (HEX)] has to be edited in the “Config Block”. No control is made to check address validity, so you must know what you are doing.

For Tetroid, this should be **0C000**.

On startup, if the adequate hardware can’t be found at the indicated address, driver will simply complain and not load at all. You have to correct address in the “Config Block” and try again.

STARTING

On startup, the driver tries first to locate correct hardware and checks GALs version before continuing if Ok.

When started from RAM version (REXT file), the driver Init routine will remove firstly any WIN driver currently loaded in the system. If there are files open on any WIN drive, driver will not load. As this uses a vectorized routine from the current WIN driver present in the system, the result may depend on the version of the already loaded driver (in the case it is an old QubIDE version or not QubATA driver).

Next, driver init routine proceeds to calibrate timing parameters according to your system characteristics and then starts detecting devices connected to the IDE bus controller.

The driver will report detected devices as following :

- (.) a dot indicates no devices detected
- (1) indicates Master device detected
- (2) indicates Master & Slave detected

Slaves alone without Masters are not supported and will not be detected.

After detection steps, driver will collect characteristics and report some information for each device found.

Devices haven't passed diagnostic correctly will be reported.

Finally, it tries to link in first partition from first master device as WIN1_ if it is QDOS valid format.

When started from RAM version, output defaults to channel #0.

Refer to the illustration on the cover page to have an idea.

MANAGING DRIVER SECTION

WIN_REMOVE — Remove driver

Syntax: **WIN_REMOVE** *Type:* **PROCedure**

Parameters:

- *none*

Description:

This command unloads all WINn provided they aren't in use and remove the current WIN driver from system if present. It frees all memories allocated by the WIN driver and unlinks all its service tasks (polling, scheduler…).

QubATA WIN driver can be restarted safely at any time without system reboot using the WIN_RESTART command (or after reset/reboot as usual from ROM).

After having removed the driver, all WIN commands, except WIN_RESTART, will report Not Found Error until the driver is restarted again.

Remarks:

This command will not return any error if no WIN driver is present or no WINn are linked in. It can be executed several times without risk.

WIN_ RESTART — Restart driver

Syntax: **WIN_RESTART** *Type:* **PROCedure**

Parameters:

- *none*

Description:

This command installs a new fresh WIN driver from start. It executes firstly an equivalent code to WIN_REMOVE command to make sure all traces relative to any previous driver are removed.

On restart, all steps realized by the driver on system boot are done (detection, calibration, diagnostic…) except that WIN1_ will not be linked in automatically. So, if master 1 or partition 1 has troubles, we can still use the driver commands and see what is happening after resetting all.

This command is the only WIN command still working after having removed driver with the previous WIN_REMOVE command.

This command executes same sequences as the ROM init routine at boot time and hence it produces same outputs and errors (on channel #0).

BASIC COMMANDS SECTION

WIN_DRIVE — Load partition

Syntax: **WIN_DRIVE *n,d,p***

Type: **PROCedure**

Parameters:

- ***n*** WIN number (1 to 8)
- ***d*** device number (1 to 32) - Masters (Odd), Slaves (Even)
- ***p*** partition number (1 to 32) or (0) for special partition Alien/Raw device

Description:

This command is used to link in a valid QDOS disk partition or a special RAW disk (Alien).

Examples:

WIN_DRIVE 1,1,1 : REMark link in WIN1_ from first partition on first master device
WIN_DRIVE 2,2,3 : REMark link in WIN2_ from third partition on second device (slave)
WIN_DRIVE 3,3,0 : REMark make WIN3_ as Alien partition for third device (master on slot 2)

Remarks:

This command will fail if ***n*** is already used for another present link or if the device/opartition doesn't exist or wasn't detected correctly by the driver. An “Alien” disk hasn’t obviously to be initialized or partitioned as it is linked in with a temporary “fake” FAT anyway.

WIN_DRIVE — Unload partition(s)

Syntax: **WIN_DRIVE *n***

Type: **PROCedure**

Parameters:

- ***n*** WIN number (1 to 8) or (-1) for all

Description:

This command is used to unlink a disk partition which was loaded with previous form of this command. If the ***n*** parameter is given as -1, unlink all WINn present provided they aren't in use.

Examples:

WIN_DRIVE 1 : REMark unlink WIN1_
WIN_DRIVE -1 : REMark unlink all WINn present

Remarks:

This command will fail if WINn_ is In Use (files open).

When used to unlink all WINn_ (with ***n***=-1), it starts from top (WIN8_ to WIN1_) and will stop at the first WIN that cannot be unlinked (In Use). If no WINn_ found/present, it doesn't return any error.

WIN_ CTRL — Set drive options

Syntax: **WIN_CTRL *n,flags%***

Type: **PROCedure**

Parameters:

- ***n*** WIN number (1 to 8)
- ***flags%*** *: (0 to 15)*
  - ✓ bit 0 turns ON checksum for FAT if set to one
  - ✓ bit 1 turns ON name case conversion from parent path when creating files
  - ✓ bit 2 allows moving files having same path when creating directory
  - ✓ bit 3 moves files into trashcan when deleted

Description:

This command turns ON/OFF options for WINn_

WINn_ should have been linked in before using this command. Options will apply only for this partition and not for whole disk.

Flags are saved on the FAT disk and still in place until they are modified again by another WIN_CTRL command.

Current settings may be obtained with the WIN_CTRL% command.

Examples:

WIN_CTRL 1,8 : REMark turn ON trashcan for WIN1_
WIN_CTRL 1,14 : REMark turn ON all options except FAT checksum
WIN_CTRL 1,0 : REMark turn OFF all options (default)

WIN_ CTRL% — Get current drive options

Syntax: ***flags%*=WIN_CTRL%*(n)***

Type: **FUNCtion**

Parameters:

- ***n*** WIN number (1 to 8)

Description:

This function returns current options setting for WINn_

See WIN_CTRL command for bits rules.

Examples:

***flags%*=WIN_CTRL%(*1*) : REMark get current flags for WIN1_**
WIN_CTRL *1*,(*flags%* ||*2*): REMark add name case conversion from parent path for WIN1_

WIN_ VER$ — Get driver version

Syntax: ***version$***=**WIN_VER$**

Type: **FUNCtion**

Parameters:

- *none*

Description:

This function will return the WIN driver version present in the system.

Returned value is 4 chars length string (as '3.06').

Last known QubIDE version was 2.02.

QubATA version starts from 3.xx

WIN_ USE — Change driver use name

Syntax: **WIN_USE *[name$]***

Type: **PROCedure**

Parameters:

- ***name$*** *string of 3 chars length (E.g. 'FLP')*

Description:

This command modifies the WIN name used by the driver or reset to default name if no name passed as parameter (default is 'WIN').

Any error reset default.

Examples:

WIN_USE '*FLP*' : REMark change WIN to FLP, so FLP1_ will refer to WIN1_
WIN_USE : REMark reset default name (WIN again)

WIN_CDEXT — Extend CDROM commands

Syntax: **WIN_CDEXT**

Type: **PROCedure**

Parameters:

- *none*

Description:

This command used to extend CDROM Audio keywords is not supported on Tetroid. It will return "Not Complete" error.

MAKE_ DIR — Make directory

Syntax: **MAKE_DIR *dirname$***

Type: **PROCedure**

FMAKE_ DIR — Make directory

Syntax: ***err*=FMAKE_DIR*(dirname$)***

Type: **FUNCtion**

Parameters:

- ***dirname$*** *name of directory to create*

Description:

These commands are implemented for completeness and are equivalent to those present in recent TK2 (from V2.24) or other extensions supporting level 2 directories.

These WIN commands shall work for creating directories for other drivers with level 2 support and vice versa. So, don't worry which ones are really used.

When creating a new directory, and depending on current options flags set for this WIN drive (see WIN_CTRL command bit #2), if there are existing files with the same path name, then these files can be moved automatically to the new created directory.

If this occurs, name case conversion from parent will be applied automatically to the moved files (see WIN_CTRL command bit #1).

In the case files moving option is not set for the drive and any of these files exist, then creating directory will fail with In Use Error.

Obviously, this option must have been set/unset according your wish before creating directory. Files moving can't be done automatically by the driver after directory has been created.

As usual, directory name may have up to 36 chars (without drive name WINn_) but it may be useful to leave some length for files name when added to directory.

Remarks :

You may have noticed that when using TK2 commands to create directory, if the file created can't be turned to directory type for any reason, a new file of "length 0" and type "data" will be left on the target device. QubATA commands don't.

If the ***dirname$*** ends with a trailing underscore, it will be removed.

PARTITIONS AND FORMAT SECTION

Normal BASIC FORMAT command has to be used to format any partition on any disk, provided partition exists and is healthy linked in with WIN_DRIVE command. BASIC FORMAT command will call automatically the Format trap routine supplied by QubATA driver.

FORMAT will only format disk which was previously initialized and partitioned. Each partition is formatted alone and becomes an independent WIN drive when linked in the system.

To initialize disks, create or modify partitions, refer to WIN_DISK command. See also TECHNICAL NOTES AND LIMITS section at the end of document.

FORMAT will be done according to the current formatting parameters set with the following command :

WIN_ FORMAT — Set Format parameters

Syntax: **WIN_FORMAT *[flags%[,bsf%]]*** *Type:* **PROCedure**

Parameters:

- ***flags%*** *: permission switches (0 to 15), default (0)*
  - ✓ bit 0 => Lock, if not set format refuses
  - ✓ bit 1 => Quick, if set format quickly
  - ✓ bit 2 => Quiet, if set don't verbose
  - ✓ bit 3 => Query, if set don't ask confirmation

  -1 => only rename partition when format (preserve all data)

- ***bsf%*** *:* Block Size Factor *(0 to 8), default (0)*
  - 0 => keep current factor
  - 1 => optimize FAT size
  - 2 => 1 sector per block
  - 3 => 2 sectors per block
  - 4 => 4 sectors per block
  - 5 => 8 sectors per block
  - 6 => 16 sectors per block
  - 7 => 32 sectors per block
  - 8 => 64 sectors per block

Description:

This command will set current parameters applicable to next WIN_DISK or FORMAT commands for WIN drives.

These parameters are global for all WIN drives and remain actives until they are modified again or the driver restarted/system reboot.

flags% is used to set permission switches, while ***bsf%*** is used to set/modify the number of Sectors per Block when creating and formatting partitions. See TECHNICAL NOTES AND LIMITS section at the end of document for some details about this.

The Sectors per Block value (derivated from ***bsf%*** parameter) is normally used before making partition with the WIN_DISK command. However, it can be used when formatting to change actual value on existing partition without need to modify partition (unless you need change the partition size).

FORMAT doesn't create or make any change to partition but may change only block size and therefore the FAT size for partition.

Before issuing a FORMAT command, at least the following setting must have be done to unlock write protection :

WIN_FORMAT 1 : REMark allow format for WIN drives

FORMAT command has to be confirmed by the user before commit unless the Quiet flag was turned On with the WIN_FORMAT settings.

Examples:

WIN_FORMAT 1 : REMark allow change or format partitions
WIN_DRIVE 2,2,1 : REMark link in WIN2_ from device 2, partition 1
FORMAT "WIN2_NameOfPart": REMark format partition and give it name
WIN_FORMAT 0 : REMark reset protection after format
…
WIN_FORMAT 3,4 : REMark unlock protection/format quickly, use 4 Sectors/Block
WIN_DRIVE 3,2,2 : REMark link in WIN3_ from device 2, partition 2
FORMAT "WIN3_Vroom": REMark format as quick as possible
…
WIN_FORMAT -1 : REMark allow name change only when format
WIN_DRIVE 2,2,1 : link in WIN2_
FORMAT "WIN2_NiceName" : REMark rename only WIN2_

QubATA driver will support existing partitions from QubIDE driver V2.xx (may be >V1.56) or created with an appropriate working version of **partition_exe** utility supplied on QubIDE support disks.

Please note that as **partition_exe** utility needs some information from the original QubIDE Driver Definition Block in memory, it can't be used if the QubATA driver is currently loaded.

However, there is no reason to use this utility as the QubATA driver comes with all necessary commands embedded in the driver itself.

Following is the main command to deal with partitions under QubATA driver :

WIN_ DISK — Manage partitions

Syntax: **WIN_DISK *[#chan,]op$[,dev[,part[,size]]]*** *Type:* **PROCedure**

Parameters:

- ***#chan*** *: output channel, default (#1)*
- ***op$*** *: operation (4 chars significant, case insensitive)*
  - "***SHOW***" => show all detected devices
  - "***DETAILS***" => display details for device given in ***dev***
  - "***INIT***" => initialize device given in ***dev*** (destroy all data structures)
  - "***SUMMARY***" => list all partitions on device given in ***dev***
  - "***REMOVE*** " => remove partition ***part*** on device ***dev***
  - "***CREATE***" => create new partition ***part*** on device ***dev*** with ***size*** MBytes
- ***dev*** *: device* number (1 Master, 2 Slave)
- ***part*** *:* partition number (1 to 32)
- ***size*** *:* size of partition in MBytes (1 to 2047=2GO)

Description:

This major command will permit all operations to deal with devices and partitions.

Refer to the previous WIN_FORMAT command for some important settings.

SHOW and ***DETAILS*** are used to display all already detected devices and their characteristics.

SUMMARY is used to display the list of existing/created partitions on any disk Media.

INIT must be used once for each disk to initialize partitions table before creating partitions. It scratches any existing partitions table and reset all data structures on the disk. It must be used only to initialize a new disk or to reset entirely/quickly an old disk.

REMOVE and ***CREATE*** are used to remove existing partition or to create a new one.

An existing partition can't be modified (shifted, moved, resized…) and has to be suppressed and recreated again and this implies data lose.

When creating a new partition, this command will use the Block Size Factor set with the WIN_FORMAT command. If the current value is not suitable for the requested partition size, then the command will shift to the nearest adequate value.

To permit any change on partitions, the write lock must be unlocked with the command WIN_FORMAT (E.g. WIN_FORMAT 1).

For obvious reasons, some changes on partitions require that all partitions on the same disk aren't in use or currently linked in the system. For simplicity, do "WIN_DRIVE -1" command to unlink all WIN drive before using WIN_DISK command.

Each modification has to be confirmed by the user before commit unless the Quiet flag was turned On with the WIN_FORMAT command.

Partitions are inserted in sequential order, so sufficient free place must be available when creating a partition in the middle of other partitions (E.G. create partition 3 when partitions 2 and 4 exist) but if this occurs, the command will report an error.

Examples:

WIN_DRIVE -1 : REMark unlink all WIN drives
WIN_FORMAT 1,4 : REMark allow change partitions and set Block Size to 4 Sectors
WIN_DISK "REMOVE",1,3 : REMark remove part 3 on device 1
WIN_DISK "CREATE",1,3,128 : REMark create a new part 3 on device 1 with 128MBytes of total size
WIN_DRIVE 3,1,3 : load the new partition as WIN3_
FORMAT "WIN3_NewName" : REMark format WIN3_ and give it this new name

Typical sequences

1) Allow partitioning and formatting Media
   WIN_FORMAT 1
2) Initialize the disk for a new Media
   WIN_DISK INIT,<dev>
3) Create a new partition on new or existing Media
   WIN_DISK CREATE,<dev>,<part>,<size>
4) Load the new created partition
   WIN_DRIVE <n>,<dev>,<part>
5) Format the new loaded partition
   FORMAT WIN<n>_Name
6) Reset protection
   WIN_FORMAT 0

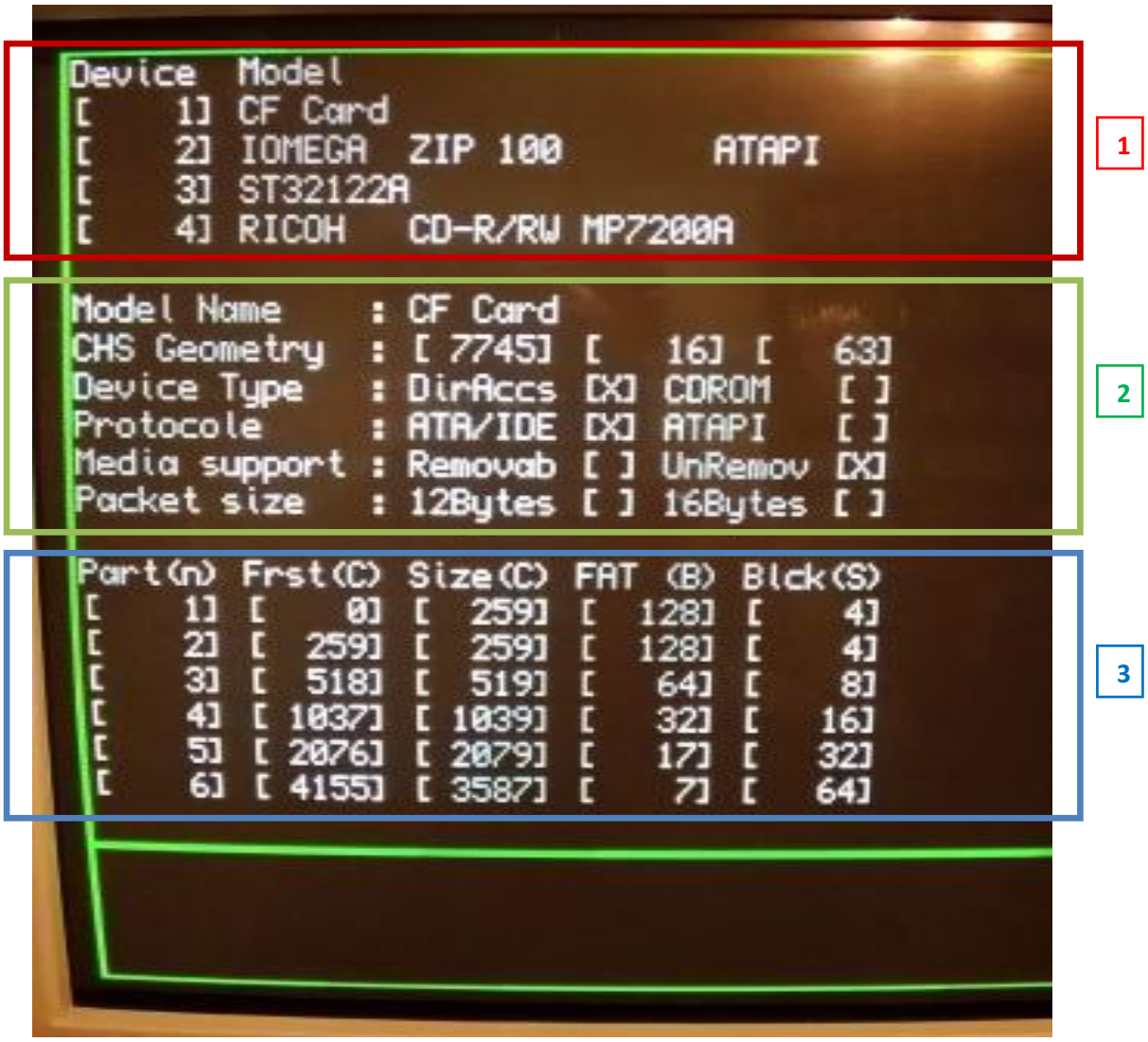


Illustration from some WIN_DISK command reports

First part is the output of **WIN_DISK *SHOW*** command. It reports devices detected and for each: the device number and the manufacturer model name.

Second part is the output of **WIN_DISK *DETAILS,1*** command and it reports details about device 1. CHS geometry gives the total number of cylinders on disk (here 7745), the total number of heads (here 16) and the number of sectors per track (here 63).
The capacity of this disk is : 7745 x 16 x 63 = 7,806,960 Sectors (of 512 bytes) = 3,7 GigaBytes
The cylinder size is : 16 x 63 = 1,008 Sectors = 504 MegaBytes

Last part shows the output of **WIN_DISK *SUMMARY,1*** command which displays a list of all partitions on device 1 :

- Column 1 is the partition number
- Column 2 is the start cylinder number for partition
- Column 3 is the number of cylinders for partition (partition size in cylinders)
- Column 4 is the number of blocks for FAT (FAT size in blocks)
- Column 5 is the number of sectors per block (block size in sectors)

DIRECT RAW ACCESS & ALIEN SECTION

QubATA driver provides an enhanced interface to access data on a disk using direct addressing whatever the format of the disk (QDOS or Not).

In direct access mode, data is read/written by whole Block at once from/to specified Position on the disk. Within QubATA direct access mode, Block is logical and its size is a multiple of sectors (of 512 bytes) which has to be specified when opening channel.

- ✓ Supported Block sizes are 512, 1024, 2048, 4096, ..., 32768 Bytes.
- ✓ Blocks are numbered from 0 to $3FFFFF (22 bits)
- ✓ Position is always at Block boundary (E.g; block_number x block_size)

To do direct access, a special file have to be open with a special name : "***Wnd***" (default is "***W2d***") :

- ***W** : Special direct access name - mandatory
- ***n*** : Block Size Factor (2=>512,3=>1024,4=>2048,...,8=>32768 bytes)
- ***d*** : Swap words key (d => no Swap, D => do word Swap)

When opening a disk for direct access from BASIC, the name should be enclosed in quotes, E.g.

OPEN #chan,"WIN2_*W2d"

Direct access is exclusive and can't be done if any file is open on the same WIN drive. While direct access is occurring, no other files can be open on the same WIN drive.

Before opening channel with the special file name, the WIN drive has to be linked in under 2 ways, depending if we access a valid QDOS partition or a whole "Alien" disk :

WIN_DRIVE 2,2,1 : REMark load WIN2_ from QDOS partition 1 on device number 2
WIN_DRIVE 2,2,0 : REMark load WIN2_ from device number 2 as Alien whole disk

At BASIC level, following commands can be used after having open channel ***#chan*** :

GET *#chan[\position],one_block_item$*
PUT *#chan[\position],one_block_item$*
***pos*=FPOS(*#chan[\position*])**

Take into account that the length of a string variable under BASIC can't exceed 32766 Bytes.

INPUT and **INKEY$** BASIC commands are not suitable for this usage. **PRINT** command may work if used properly. Read (**GET**) and Write (**PUT**) don't change current position when complete.

Writing a Block (**PUT**) will be flushed immediately to disk without any buffering delay.

Direct access should be finished with a normal CLOSE command as : **CLOSE *#chan***

At assembler level, following traps are implemented in a way to ensure compatibility with TK2.

IO.FSTRG (trap #3, D0=3) Read Block as String

This routine doesn't change current position pointer.
Compliant to TK2 implementation for GET Basic Keyword.

Call parameters :

a1	base of buffer to be filled (updated to end+1 on return)
d2.w	length of buffer (Unsigned)

Return :

d1.w	number of bytes fetched

Rules :

- d2= block size (n x sectors) => copy block to (a1)+ and return d1= block size, d0=OK
- d2= 2 => block size to (a1)+, copy nothing and return d1=2, d0=OK
- d2= 2+block size => length to (a1), copy block to 2(a1)+ and return d1=2+block size, d0=OK
- Anything else : return d0=Bad Parameter error

IO.SSTRG (trap #3, D0=7) Write Block as String

This routine doesn't change current position pointer.
Compliant to TK2 implementation for PUT Basic Keyword.

Call parameters :

a1	base of buffer to be send to disk (updated to end+1 on return)
d2.w	number of bytes to write (Unsigned)

Return :

d1.w	number of bytes sent

Rules :

- d2=block size (n x sectors) => copy block from (a1)+ and return d1=block size, d0=OK
- d2=2 => copy nothing (ignore) and return d1=0, d0=OK
- d2=2+block size => copy block from 2(a1)+ and return d1=2+block size, d0=OK
- Anything else : return d0=Bad Parameter error

IO.EXTOP (trap#3, d0=$9) Do externally operation

Call parameters :

a2	address of the user routine to be called
d1/d2/a1	parameters supplied to the user routine passed in a2

Return :

d0/d1/a1	as returned from user routine

When control is passed to the user routine, following registers are supplied as:

a0	pointer to the Channel Definition Block
a2	pointer to the FAT for the drive
a3	pointer to the Driver Definition Block
a4	pointer to the Physical Definition Block
a5	pointer to the base of the drive IDE registers (drive selected)
a6	base of system variables

User routine is called in Supervisor mode. All registers (other than d0/d1/a1) may be smashed.

FS.POSA (trap #3, d0=$42) Set absolute position

Call parameters :

d1.l	requested new absolute position (byte number - start from zero)

Return :

d1.l	new byte absolute position
	d1= block_number x block_size
	block_number (start from 0)
	block_size (512, 1024, 2048,... bytes)
d0=OK or EOF error	

Pointer is always set to the lower block boundary.

FS.POSRE (trap#3, d0=$43) Get current position

In direct access mode, this call is used to get the current position and no change is made to the current position. On entry, d1 is ignored for this call.

Return :

d1.l	current byte absolute position (current block_number x block_size)
	block_number (start from 0)
	block_size (512, 1024, 2048, ... bytes)

Pointer returned is always at block boundary.

TRASHCAN MANAGEMENT SECTION

QubATA driver comes with a fully functional trashcan started with QubIDE ROM from V2.0. Each partition can have its own trashcan. Trashcan is switched On or Off with the WIN_CTRL command.

As trashcan flag is set in FAT and flushed to disk, it will remain in place over load/unload partition, system reboot... until is changed again with a new setting.

With the trashcan active, deleted files are not really deleted but moved into trashcan hidden directory. When trashcan is switched back to Off, files already stored in the trashcan remain there but next files deleted are really suppressed definitely.

Files in trashcan are associated to their entry number and not to their name, so successive deleted files having the same name may reside together in trashcan as their entry numbers differ.

Directories are never moved to trashcan and obliviously can't be deleted if not empty.

Trashcan can't be accessed by the file system and has to be managed with a special utility program to show contents, recover deleted files or suppress definitely files for good. This can be made with the original ***trashcan_obj*** but another tool comes with QubATA driver to deal with this feature.

This tool, called ***TrashMan_task*** (Trashcan Manager) still in a basic state (no PE, no colors…) but does fully the job and it can be used to :

- List/Recover/Suppress files in trashcan
- Export files from trashcan to another device (leave files in trashcan)
- Empty trashcan (suppress all files)
- Compress really trashcan (suppress holes & shrink directory => reduce size)
- Turn On/off options for the WIN drive (all options individually)

This program is menu driven and self explanatory. It is Turbo compiled with ToolKit runtime embedded. It requires only a decent/recent TK2 to be present and executed as :

EX *WIN1_TOOLS_TrashMan_task*

Remarks :

Active Trashcan has to be delighted from time to time and its directory has to be compressed. Otherwise WIN drive will continue to grow even when unwanted files are regularly deleted.

TECHNICAL NOTES AND LIMITS

This driver shall work with any ATA or ATAPI device compliant to ATA/ATAPI-4 protocol. It is also removable Medias aware.

This includes hard disks, ZIP units, CDROM readers and CF Card memories with adapter. As mentioned, hardware must be fitted with the last GALs version 2 (mandatory).

Compatible devices must fire up in PIO modes 0, 1 or 2 even if they support higher mode 3 or 4. This is generally the case and there is no problem, but some existing (bad) devices (specially SD card adapters) fire up in mode 3 or 4 with using IOREADY line (data flux control on bus) which may not be supported on hardware. In this case, driver can do nothing and problems may occur in data transfer (probably device will get locked until is reset).

To use a CF Card memory on QubIDE native hardware, you need an “IDE to CF Card adapter” which can be easily found in the market at very reasonable price. Be aware to choose a model with Master/Slave select (by switch or jumper), otherwise a little modification has to be made on the QubIDE Card (connect CSel line to Ground).

To use SD cards memory, you need a “True IDE CF Card to SD adapter” over the “CF Card socket” but this may not work at all, depending on used adapters.

Please note that memory cards (CF, SD…) are not “Hot” removable Medias and should not be unplugged/plugged unless the system is really powered down to prevent device from electrical damages.

The following limits apply to this driver :

- 1 to 8 partitions can be loaded at the same time (WIN1_ to WIN8_) provided there is enough memory to load FATs.
- 1 to 32 physical devices (16 couples of Master/Slave) can be handled by driver provided your hardware is adequate (using Expander on QubIDE, per example). Please, ask for mod information (published information insufficient). Device number starts from 1 (Masters Odd, Slaves Even).
- 1 to 32 partitions can be used on each device or removable Media. Partition number starts from 1. Partition number 0 is a special case for use with “Alien” devices. The number of partitions doesn’t impact the system performance as they aren’t all loaded simultaneous.
- Each partition can be sized up to 2 GigaBytes depending on the number of blocks and the number of sectors per block used (see below).
- Each partition can handle up to 65275 different files in one FAT.
- Block size is independent for each partition and can be from 1 up to 64 sectors (512 bytes) per block.
- The biggest file size supported and working under QDOS File System may be a whole maximum partition size (minus FAT size) so approximately 2 GigaBytes.

The block size is set with FORMAT parameters using the WIN_FORMAT command. The block size can be changed after partition creation but before formatting.

The size of partition is defined when creating the partition with WIN_DISK CREATE command.

As the whole FAT of each partition is loaded into memory when the partition is linked in with WIN_DRIVE command, the size of the partition may have importance regarding the available memory on used system. Partition size and Block size (number of sectors/block) must be chosen carefully when creating and formatting partition. Any change for this implies a reset of partition and lose of all data on the concerned partition.

As a general rules, consider following :

Each block in the partition, whatever the block size used, takes 4 bytes as 2 index entries in the loaded FAT in memory. Each index in the FAT is an unsigned short integer (0 to 65535 or $FFFF).

The size of FAT doesn't depend on the size of partition but mainly depends on the number of blocks in partition.

The size of partition (in bytes) depends on the number of blocks and the number of sectors per block.

So the equation will be :

- Take the smallest block size possible (don't waste bytes on disk) – nearest 1,2 or 4 sectors/block,…
- Take the maximum number of blocks possible (to have the biggest partition size) – nearest 65535 blocks
- Produce a reasonable FAT size regarding available memory on the system and the number of FATs you have to load at one time – 128 or 256 KB per FAT is a good compromise.

Here is a little summary of different FAT size needs regarding block sizes used :

- 8 KBytes per 1 MBytes of storage with 1 Sec/Blk
- 4 KBytes per 1 MBytes of storage with 2 Sec/Blk
- 2 KBytes per 1 MBytes of storage with 4 Sec/Blk
- 1 KBytes per 1 MBytes of storage with 8 Sec/Blk

- 1 KBytes per 2 MBytes of storage with 16 Sec/Blk
- 1 KBytes per 4 MBytes of storage with 32 Sec/Blk
- 1 KBytes per 8 MBytes of storage with 64 Sec/Blk

A partition of 2 GBytes size with 64 Sectors/Block may be made with a 256 Kbytes FAT size only, but the smallest file with 1 Byte per example will take a whole block of 64 sectors (32768 Bytes) of storage.

TROUBLESHOOTING

FAT check summing

This option set with the WIN_CTRL command for each partition is CPU consummating and may have some impact on the system performance. It must be clear that this option doesn't prevent FAT from being corrupted (by other programs or commands, per example) but it will only detect that FAT is corrupted. If this happens, try to reboot the system immediately without making any change to any file. But as the FAT is flushed regularly from memory to disk, there is really a little chance that the FAT on the disk could be safe and not corrupted.

When the FAT present on the disk partition becomes corrupted, it is too late and data is lost (unless using a special tool). You probably have to format the partition and restore data from the last backup you have made last evening (haven't you ?).

In the worst case with the first partition on the disk being corrupted, all partitions may be lost as the partitions table resides here. This may technically be improved but needs change to QubIDE partition format and compatibility will be probably broken.

After an observation period and if you don't use bad behaving programs, the FAT checksum can be reasonably turned Off. Major FAT problems observed are caused by electrical interferences particularly when used with a QPlane backplane.

Note also that when modifying partitions on a disk with the WIN_DISK command, checksum option will be unset automatically for the first partition of the concerned disk to permit writing of the partitions table. You have to set this option again on partition 1 if wished. This is not a bug but may be improved in the future.

Other QubIDE utilities

Some of the software utilities supplied on QubIDE disks should not be used with QubATA driver specially those using information from Driver Definition Block in memory. As this has changed, compatibility can't be made in this way. This applies to **partition_exe** and **WinEditor_obj** by example.

Previous vectorized routines entries (link/unlink/rename) are maintained and still working with this driver.

Trashcan_obj (used to access Trashcan) can be used safely with no problem, however it is more recommended to use the new **TrashMan_task** utility supplied with this driver.

Backup utilities shall still work normally but not all tested.

Other utilities may be updated and supplied in the future if still having any interest.

Also, a separate extension toolkit will be soon supplied to access major (old and new) driver features with complete programming interface description.

CREDITS AND PERMISSIONS

First thanks and credits go obviously to Nasta for designing QubIDE, P.A. Borman for writing the QubIDE driver and QubbeSoft/QBranch for producing, distributing, supplying manuals and maintaining QubIDE hardware.

Thanks also to Dilwyn Jones, Rich Mellor and Marcel Kilgus for feedbacks and permissions.

Feedbacks, suggestions and comments are welcome via any channel (QL users list, forums…).

MANUAL REVISION HISTORY

Revision 1.00

First manual version. Need probably (surely) English review.

Revision 1.00 (T)

Specific manual version for Tetroid Interface.