DivIDE Plus

Hard Disk Expansion Interface
For the Sinclair ZX Spectrum

© Jarek Adamski & Jurek Dudek

for RWAP Software

USER MANUAL

Manual Update 7.1 (January 2011)
© RWAP Software
www.rwapsoftware.co.uk

Table of CONTENTS

NOTICE....................6

IMPORTANT NOTE FOR SPECTRUM +2A, +2B AND +3 USERS....................6

COMPATIBLE ADD-ONS....................7

AVAILABLE FIRMWARE....................7

INSTALLATION OF DIVIDE PLUS....................9

Setting Up....................*10*

USING A DISK DRIVE....................10

Compatible Disk Drives....................*10*

Jumper Settings....................*10*

Use in 128K Mode....................*11*

Installing Firmware....................*11*

Loading the Disk Operating System....................*12*

GETTING SOFTWARE ONTO COMPACT FLASH OR A HARD DISK....................12

Running Problem Games....................*12*

FATWARE FIRMWARE....................13

INSTALLATION....................14

Installing for the first time....................*14*

Using FATWare....................*14*

RESIDOS FIRMWARE....................15

INSTALLATION....................16

Installing for the first time....................*16*

Upgrading to a new version of ResiDOS....................*16*

Installing Packages....................*16*

SETTING UP YOUR HARD DISK OR COMPACTFLASH....................18

IDEDOS or FAT?....................*18*

Setting up a FAT drive....................*18*

Setting up an IDEDOS drive....................*19*

SWITCHING DISKS OR COMPACTFLASH CARDS....................20

ENTERING RESIDOS COMMANDS....................21

Entering functions....................*21*

PARTITION MANAGEMENT....................22

Creating, deleting and listing partitions....................*22*

Automatic drive mappings....................*23*

Mapping drives to partitions....................*23*

DRIVES, DIRECTORIES, PATHS AND USER AREAS....................25

Setting the default drive, user area or path....................*25*

Creating and deleting directories....................*26*

Showing the current location....................*26*

USING FILES IN RESIDOS....................27

Filenames....................*27*

Wildcards....................*27*

LOADing and SAVEing files *28*
Changing BASIC auto-run status *28*
Cataloguing files *29*
Deleting files *29*
Copying files *29*
Renaming files *30*
Changing file attributes *30*
128K RAMDISK EMULATION 30
Setting up the emulated RAMdisk *30*
Using the emulated RAMdisk *31*
CUSTOMISING RESIDOS 32
Custom colours *32*
Autorun files *33*
Processor speed *33*
USING OPTIONAL PACKAGES, ALTERNATIVE BASICS AND INTERFACE 2 ROMS IN RESIDOS 34
Installing ROMs *34*
Installing ROMs from a file *35*
Installing ROMs already loaded into memory *35*
Managing alternative ROMs *36*
Using alternative ROMs *36*
SPECTRUM MODES WITH RESIDOS 38
128K Mode *38*
48K mode *38*
Disabling ResiDOS *39*
BREAK AND ERROR-TRAPPING 40
BREAK-trapping *40*
Error-trapping *40*
Warning! *41*
PACKAGES 42
THE RESIDOS TASK MANAGER 42
Overview *42*
Managing Tasks *43*
Other options *43*
THE FATFS PACKAGE 44
TAPEIO PACKAGE – LOAD / SAVE .TAP AND .TZX FILES 45
The TapeIO Package *45*
EXTENDED CHANNELS 49
Commands and functions for streams and channels *50*
CHANNELS PACKAGE 51
RESIDOS COMMAND REFERENCE 58
Disk management commands *58*
`%DRIVE unit, cyl, head, sec` *58*
Partition management commands *59*
Drive mapping commands *60*
Drive, directory and path commands *60*
File management commands *61*
Customisation commands *63*
ROM management commands *64*

Spectrum mode commands *65*
Error-trapping commands *65*
Error-trapping functions *66*
Channel commands *66*
Channel functions *66*
RESIDOS PROGRAMMING 67
Detecting ResiDOS *67*
Detecting ResiDOS from BASIC *68*
Package calls *68*
The ResiDOS Package *70*
The IDEDOS Package *89*
+3DOS Calls *89*
IDEDOS Calls *90*

+DIVIDE FIRMWARE 93

NOTICE 94
INTRODUCTION 94
COMPATIBILITY 94
HOW TO ACCESS IDE DEVICES 95
THE CATALOGUE (OR FILE DIRECTORY) 96
+DivIDE File-Types *97*
+DIVIDE DISK DRIVE COMMANDS 98
Saving a File and Verifying *98*
Loading a Program *99*
Breaking into a Program *99*
Renaming a File *99*
Merging a Program *100*
Erasing a File *100*
Creating an Autoload File *100*
SNAPSHOT FILES 102
Problems? *103*
ABBREVIATED SYNTAX 103
WILD-CARD FILES 104
FORMATTING NEW VIRTUAL DISKS 104
ERROR REPORTS 105
Nonsense in Basic *105*
Format Data Lost *105*
Sector Error *105*
Disk Write Protected *105*
File Not Found *105*
ADVANCED DISK OPERATIONS 106
Sector by Sector Copying - Two Drives *106*
Reading and Writing to a Sector *106*
Using Streams and Channels *106*
The Execute File *107*
QUICK REFERENCE GUIDE 109
BASIC *109*
READ DISK *109*
LOAD FILES *109*

SAVE FILES *109*
DELETE FILES *109*
FORMATTING DISKS *109*
STREAMS *110*
DIRECT SECTOR ACCESS *110*
WILDCARDS *110*

DEMFIR FIRMWARE 111

INTRODUCTION 112
STARTING UP DEMFIR 112
USING DEMFIR 112
The NMI Menu *112*
The File Browser *114*
The PokeManager *115*

MDOS3 FIRMWARE 117

What is MDOS3? *118*
Using MDOS3 .... *118*
NMI Menu *118*
MDOS3 Command Reference *122*
MDOS3 PROGRAMMING 125
The MDOS3 Service Hook Codes *125*
Table definitions: *135*

PROGRAMMING FOR DIVIDE PLUS 137

Implementation of the 128K BASIC Mode on DivIDE Plus *137*
Implementation of greater ROM/RAM space on DiviDE Plus *138*
How the DivIDE Modes Operate *139*

NOTICE

The manual has been written specifically for users of the DivIDE Plus hard disk interface for the Sinclair ZX Spectrum and contains various supplements for each type of firmware which can be used with that interface.

The manual is intended to provide the user with detailed information adequate for the efficient installation and operation of the equipment involved. However, while every effort has been taken to ensure accuracy, the manufacturer assumes no liability resulting from errors or omissions in this manual, or from the use of the information contained herein.

The manufacturer reserves the right both to change the specifications of the DivIDE Plus and the firmware and to revise this publication from time to time without obligation to notify any person of such revision or changes.

RWAP Software
April 2007

Important Note for Spectrum +2A, +2B and +3 Users

Please note that the interface as shipped, needs a jumper added to work properly with the Spectrum +2A, +2B and +3 computers. This jumper is located inside the interface, near the Spectrum edge connector and needs to be closed for compatibility with these machines. Please see Jumper Settings below.

In order to access the jumper, you will need to open the DivIDE Plus. The interface enclosure is glued on the corners – the best means of opening the enclosure, is to us ean isopropylic or ethylic alcohol, such as found in many video head cleaning sprays.

NOTE: This jumper should be left open (or missing) for use on the 16K and 48K original Sinclair ZX Spectrum, the Spectrum + (both 48K and 128K toastrack models) and the Spectrum +2.

Compatible Add-ons

The DivIDE Plus is unable to work with any Spectrum add-ons which have their own external memory, or use the same I/O ports such as:

- Sinclair ZX Interface 1
- Sinclair ZX Interface 2 with a ROM cartridge inserted
- Sinclair ZX Printer
- RAM Turbo Interface with a ROM cartridge inserted
- MB02+
- PlusD Disk Interface *(although we are working on this one)*
- DISCIPLE interface
- ZXMMC+
- ZXCF
- Mutliface 1, 3 and 128
- D40 / D80
- INTERFACE M/P

The following interfaces have all been tested successfully. To ensure maximum chance of success, plug these into the DivIDE Plus through connector, as not all pass the full range of signals to their own through connector:

- Kempston Joystick Interfaces (and compatibles) - although some may not work if they do not fully decode the port used to communicate with the ZX Spectrum
- RAM Turbo Interface (without a ROM cartridge inserted)
- Sinclair ZX Interface 2 (without a ROM cartridge inserted)
- Kempston Mouse Interface
- K-Mouse 2008 Inerface
- PS/2 PC Keyboard Interface
- UR/4
- Alphacom 32 Printer

Available Firmware

The functions provided by the DivIDE Plus depend on both the hardware and the firmware which is contained on the built in flash EPROM. There are various types of firmware available, all of which provide different functionality:

FATware 0.12

Written by Baze, this is one of the easiest to use firmwares, although it is read only at present. It natively supports up to 8 standard FAT-16 partitions (including long file names), which means you can format a CompactFlash card on your PC, insert it into the DivIDE Plus and access any files on it.

FATware supports TAP, SNA, Z80, SCR and interlaced SCR files. To access the files, press the Snapshot (NMI) button and navigate through the directory structure to find the file you wish to load.

The battery backed memory on the DivIDE Plus allows FATware to remember the last used file and also the position in the disk browser for when you turn on the Spectrum.

DEMFIR (DTP's Emulator Files Runner)

This is similar to FATware and handles the ISO 9660 file system found on CD-ROMs. You can use either CDs or unfragmented ISO images stored on a disk.

DEMFIR supports TAP, SNA, Z80, MFC and SCR files, but as with FATware is currently read only.

In order to use DEMFIR, you should put your CD into the CD-ROM drive and turn that on and allow it to start up BEFORE you switch on the Spectrum. Once the drive is up and running, turn on the Spectrum and then press the NMI button to start up DEMFIR. You will see the DEMFIR version message (D0.7b) appear in green in the top right hand corner. Press the ENTER key to browse the CD and find the files you need.

MDOS3

This is an enhanced version of the MDOS/MDOS2 systems which were used in D40/D80 disk units. MDOS3 is a full-featured OS which works with raw diskette images and supports up to 4 virtual drives.

One of its DivIDE-specific add-ons is a tape emulator.

With a huge D40/D80 software base available this is worth a try, especially if you're a former Didaktik user.

+DivIDE

Written by Rudy Biesma, this is an adaptation of the GDOS/G+DOS system found in the DISCiPLE/+D disk interfaces. It should work with all DISCiPLE and +D software which doesn't call routines in the G+DOS ROM directly.

+DivIDE uses virtual disks of 1600 sectors each on the ATA drive with some enhancements to original GDOS/G+DOS features. **NOTE:** +DivIDE uses raw LBA access to disk images so take care if you're hosting another file system on your disk.

ResiDOS

Written by Garry Lancaster, this uses the Spectrum +3e’s hard disk format (IDEDOS), although it is possible to partition a hard disk (or compact flash card) between IDEDOS and a windows FAT file system. Packages can also be installed into ResiDOS to provide you with direct access to FAT-16 formatted disks and even a ZX80 and ZX81 emulator!

ResiDOS appears to require a 48K or 128K ZX Spectrum in order to work - we could not get it to run on a 16K ZX Spectrum.

ResiDOS has a wide range of commands based on both +3DOS (the Spectrum +3's file system) and IDEDOS (the file system for the Spectrum +3e ROM). If you use the +3e ROM, you are now able to access both the floppy disk drive on the Spectrum +3 and the hard disk at the same time.

Installation of DivIDE Plus

The DivIDE Plus hard disk interface will work with all Sinclair ZX Spectrums and many clones. There is a jumper on the Board (near the connector) which is required to be set to allow you to use the interface with Spectrum +2A, +2B and +3 computers. DivIDE Plus will even work from within the ZX Spectrum's 128K BASIC mode, although this mode can only be enabled from within software – see programming for DivIDE Plus at the end of this manual.

A jumper on the board, near to the IDE / CF connector, is required to be set in order to allow you to program the EPROM chip with a different firmware.

Support is restricted to IDE/ATA devices which support LBA mode. This means that older hard disk drives are supported, together with Compact Flash memory (FlashIDE or CF cards). You can however, use two drives or FlashIDE cards (a master and a slave) on the same IDE cable.

SATA hard disk drives, CD and DVD drives are not supported.

In order to make the manual easier to understand, where a number is expected, you will see (n) or similar – this is to distinguish from where the letter itself should be used. Letters are not case sensitive, therefore for example:

```
          LOAD d(n)  can be LOAD D1 or LOAD d2
```

Where we refer to an IDE device this can be a hard disk, FlashIDE or Compact Flash memory card connected to the DivIDE interface using an IDE cable or the Compact Flash slot.

Setting Up

WARNING: Don't forget to turn off the power to your Spectrum before you connect or disconnect any peripherals, including the DivIDE Plus Interface.

With the power off, carefully connect your DivIDE Plus Interface to the computer and your IDE device to the DivIDE Plus. If you are using other peripherals, connect them to the computer in their normal places. If you are using a peripheral which plugs into the Spectrum's edge connector, you'll need to plug these into the through connector provided on top of the DivIDE Plus.

Turn the Spectrum power on and the normal Spectrum introductory screen will appear on your monitor/TV. The red indicator light on your DivIDE Plus will come on, indicating that the electrical connection between the DivIDE Plus and the Spectrum has been correctly made. Turn on the power to the disk drive and insert a blank disk.

If you are using the Spectrum 128K or + 2, go into the EDIT mode and then move the cursor to select SCREEN. Then press ENTER. You must be in this screen mode to avoid problems later.

USING A DISK DRIVE

Compatible Disk Drives

The DivIDE Plus interface will work with up to 2 devices at a time.

You can connect any IDE/ATA devices which support LBA mode and a Compact Flash memory card up to 2GB. This means that older hard disk drives are supported, together with Compact Flash memory (FlashIDE or CF cards). ResiDOS can support non-LBA mode devices, but only if you use the IDEDOS format.

You can use two drives or FlashIDE cards (a master and a slave) on the same IDE cable, or one drive on the IDE cable (which will be a master) and one compact flash memory card in the CF Slot (which will be a slave if there is an IDE connected).

We may be able to provide suitable external disk drives or Compact Flash memory cards if you do not have any.

Jumper Settings

The DivIDE Plus interface has three different jumpers which can be closed or open. A closed jumper has a linkage block between the two pins – remove this to have the jumper open.

There is one jumper inside the Interface case – this must be closed if you wish to use the interface with a ZX Spectrum +2A, +2B or +3 computer. Please beware that the interface unit is sealed with silicon – it can be prised open gently to break the seal.

The other two jumpers are found on the rear of the interface, near the IDE connector.

The jumper nearest the IDE connector must be closed if the Compact Flash slot is to be used on its own (this makes the Compact Flash the master drive). Remove the linkage block (set the jumper open) if you are using the IDE connector (the Compact Flash slot becomes the slave drive and the IDE port the master).

The jumper next to this (furthest from the IDE connector) when closed protects the onboard EPROM from being overwritten. This jumper must be open to install new firmware.

Use in 128K Mode

If you wish to use the DivIDE Plus Interface from 128K Mode on the Spectrum 128K, Spectrum +2, +2A, +2B or +3, then you will need to issue the following commands (remember to close the jumper inside if you are using the interface with a ZX Spectrum +2A, +2B or +3 computer):

```
OUT 32765,0

PRINT USR 0
```

You may need to reset the Spectrum after this command.

Installing Firmware

The DivIDE Plus comes shipped with all of the main firmwares installed.

To switch between each firmware at any time, you need to use the following BASIC commands on your Spectrum:

```
OUT 23,x

PRINT USR 0
```

where `x`:
0 - FATware (without the optional Firmware Selector installed)
2 - DEMFIR
4 - +divIDE
6 – MDOS3
28 – FATware (with the optional Firmware Selector installed)

For ResiDOS, you will currently need to use the following commands:

```
POKE 18455,237:POKE 18456,65:POKE 18457,199

PRINT USR 18455
```

Control of the various firmwares and the options available on DivIDE Plus is made much easier with the optional firmware selector software which provides a menu driven method of choosing the firmware and selecting whether you want 48K or 128K mode. Even better, the firmware selector will remember the settings for the next time you start up the Spectrum !

Updating firmware onto the DivIDE Plus could not be easier. Download the special flasher program plus the firmware from ourselves onto your PC and then use one of the various utilities available from WorldofSpectrum.org (such as playtzx or an emulator) to play this though the PC's speakers into your Spectrum. You will need to use `LOAD "″` on the ZX Spectrum and the program will then overwrite the installed firmware.

Loading the Disk Operating System

Connect the computer, the DivIDE Plus and the disk drive(s) and turn on the power. The normal Spectrum title screen will appear, and the indicator lights on the DivIDE will come on. The firmware will be loaded from the EPROM on the DivIDE interface with no further action required.

GETTING SOFTWARE ONTO COMPACT FLASH OR A HARD DISK

Many people will find that the easiest way for them to get programs onto the ZX Spectrum using the DivIDE Plus interface, is to use a PC to download .tap files from the internet and copy them onto the compact flash card.

For playing the majority of games, this is ideal, and you can use *FATware* to select and load the desired program. The only downside to this is if the program needs to write to the compact flash memory, as *FATware* is read only.

Another option is to load software from cassette and then use either *ResiDOS* or *+DivIDE* firmware to save it to the compact flash card / hard disk. A PC based management program has been written by Scott Falk-Heuhn to enable you to manage and copy *+DivIDE* images from compact flash onto your PC.

If you have software on +3 disk, if it is in BASIC, then you can easily copy software onto the compact flash card using *ResiDOS* simply by disabling ResiDOS whilst you load the software from disk, then re-enable *ResiDOS* to allow you to save it to the compact flash card/hard disk. However, if you want the ability to use both the disk drive and the compact flash card at the same time, you will need to upgrade the +3 to use +3e ROMs.

Running Problem Games

Some programs use a peculiar loading scheme which can cause them to fail to load from compact flash card. You can get around the majority of these programs by entering the command:

```
OUT 32765,48
```

before you use the `LOAD ""` command.

FATWare Firmware

for

ZX Spectrum DivIDE Plus Interface

© Baze

USER MANUAL

General DivIDE Version

Manual Update 1 (April 2007)

Installation

Installing for the first time

FATWare comes supplied on the DivIDE Plus – it is always there when you switch on power to the ZX Spectrum. As soon as you turn on you should see the *FATWare* boot screen appear!

FATWare will work with hard disks and also Compact Flash cards which contain at least one FAT16 partition (the standard format for Compact Flash cards).

Please check to see that *FATWare* has recognised your IDE or Compact Flash card – if not, you will need to press down the SPACE key whilst you turn on the power to your Spectrum. This makes it initialise itself and read the details of attached devices.

Using FATWare

FATWare is probably the easiest firmware to start with – as soon as you press the Snapshot (NMI) Button, you will see a disk navigator – this allows you to move back and forth through the directories on all attached devices.

FATWare allows you to load files in the following formats:
.tap (this is a file which simulates a cassette tape)
.z80 (snapshot of the Spectrum's memory)
.sna (snapshot of the Spectrum's memory)
.scr (screen display)

Simply move the cursor using the cursor keys – press <ENTER> to go into a sub-directory (or on .. to go back up a directory) until you find the file you want. Highlight the file, press <ENTER> and the file will be loaded into memory !

If the file is a .TAP file, the NMI menu will only allow you to select the file and you are then returned to BASIC. Simply enter `LOAD ""` to load the selected program.

We have also come across some instances where .z80 snapshot images (particularly those created with the Task Manager from *ResiDOS*) will not load correctly under *FATWare*. If this occurs, then the best way is to use the `%SNAPLOAD "filename"` command in *ResiDOS*.

That is all there is to it as currently *FATWare* is read only.

ResiDOS Firmware

for

ZX Spectrum DivIDE Plus Interface

USER MANUAL

Version released for RWAP Software DivIDE Plus Interface

Manual Update 1 (April 2007 – based on ResiDOS v2.25)

Installation

Installing for the first time

Installing ResiDOS is simple. First download the appropriate version of the installation software for the DivIDE Plus interface from the downloads page (http://www.worldofspectrum.org/residos/downloads.html) and load it into your Spectrum. The software is provided in the form of an emulator TAP or TZX file, so you will need to transfer it using software such as *playtzx* or an emulator. You'll find suitable software on the World Of Spectrum site. Once it's transferred, just LOAD it into your Spectrum and follow the simple on-screen instructions.

During installation, the program will confirm the size of your interface's RAM (usually 128K, 512K or 1024K).

Once you have changed the jumpers on your interface as instructed, the Spectrum will reset and you should see the *ResiDOS* boot screen appear!

Upgrading to a new version of ResiDOS

When a new version of *ResiDOS* becomes available, installation is even easier. Just load the installation program from tape as normal, and it will automatically detect your existing version. You can then choose to upgrade, and the system will be automatically updated, leaving all your installed ROMs and tasks still in the interface's memory.

If you choose to re-install, the normal installation procedure will occur. This will wipe all the packages, ROMs and tasks from your interface's memory.

If you have the FATfs package installed, then instead of loading from tape you can simply copy the installation files onto your FAT16 card, and `LOAD` them from there with
`LOAD % “residos.bas”.`

If you have the TapeIO package installed, you can copy the installer tape image (residos.tzx) onto your card and then load it with `%tapein "residos.tzx":LOAD ""`

The latest version of *ResiDOS* (and its manual) is available on the *ResiDOS* homepage: http://www.worldofspectrum.org/residos/

Installing Packages

One of the most useful functions of *ResiDOS* is the ability to install packages to add extra functionality to *ResiDOS*.

These packages can be downloaded directly from the *ResiDOS* homepage: http://www.worldofspectrum.org/residos/ and currently include both a ZX80 and a ZX81 emulator, plus the *ResiDOS Task Manager* and also a package to provide the ability to access FAT-16 formatted disks (in 8.3 format) directly using the *ResiDOS* commands.

Setting up your hard disk or CompactFlash

From v2.00, *ResiDOS* can work with cards and disks formatted to one of two different filesystems:

IDEDOS, including +3DOS
FAT

The base installation of *ResiDOS* only supports IDEDOS drives, so in order to allow use of FAT drives, you must install the optional FATfs package. In the future, other filesystems may also become available.

You can use any combination of IDEDOS and FAT drives. If you have two drives or cards, you can format one with IDEDOS and one with FAT. If you have only one card, you can choose either system. It is also possible to divide a single card between IDEDOS and FAT; see this page for details (although please note that it was written for the +3e, and so the syntax of the commands is slightly different).

IDEDOS or FAT?

FAT has the following advantages:

- All PCs and Macs can read and write to FAT drives (as can digital cameras, PDAs etc), so transferring files between your PC and Spectrum is extremely easy .
- Each partition can be up to 2GB in size (when FAT32 support is added, even this limitation will be removed); IDEDOS partitions are limited to 16MB (although there is no limit on the number of partitions).
- Files can be organized into directories

IDEDOS has the following advantages:

- IDEDOS drives can be directly used by the +3e
- IDEDOS drives can be used with the ZXVGS operating system
- CP/M, which may be ported to ResiDOS soon, needs IDEDOS drives
- Swap partitions, which could be used by games and other apps, are only currently available on IDEDOS drives

Setting up a FAT drive

ResiDOS cannot directly format FAT drives. Instead, format your card or disk with your PC. You need one or more FAT16 partitions. In Windows XP, this is done by selecting "FAT" as the format type; do not select "FAT32", as the FATfs package does not support it at the time of writing.

Then, after installing *ResiDOS* and the FATfs package, simply insert the card into your interface and reset the Spectrum. It should be automatically detected and mapped to one or more drive letters (shown on the startup screen).

Setting up an IDEDOS drive

Before you can use your hard disk or CompactFlash, it needs to be initialised with the IDEDOS system. *ResiDOS* can work with two different physical drives, normally known as the "master" and "slave" unit; in all the following commands, the first number given identifies the unit number, which is 0 for the master and 1 for the slave.

The first thing to do is make sure that ResiDOS understands the structure of your disk. Normally, it automatically detects this and displays the details on the boot screen. However, if for some reason this doesn't work, you can manually specify the parameters (shown on the drive itself, or to be found on the manufacturer's website) with the following command:

```
%DRIVE 0,977,10,17
```

(The above command assumes your hard disk is the master drive (unit 0) and has 977 cylinders, 10 heads and 17 sectors). Once you have entered this command, the details are permanently stored in your interface's RAM.

You can re-enable auto-detection for the master drive (unit 0) with the command:

```
%DRIVE 0
```

If you want to disable detection of a drive altogether, to save a bit of time on boot, you can do so with the following example command, which disables detection of the slave drive (unit 1):

```
%DRIVE 1,0,0,0
```

When you're sure that ResiDOS knows the correct details, you need to initialise the disk. Note that doing this erases everything currently stored on the disk. The command to use is:

```
%REFORMAT 0,15
```

This initialises the master drive (unit 0), allowing up to 15 partitions to be created. You can specify any number you like for the number of partitions, but note that the more you allow, the slower certain partition management commands will be. Each partition can be up to 16Mb in size; I'd recommend calculating the number you would need to completely fill the disk, and adding several more for a bit of elbow-room. So, a 1Gb disk could use about 1000/16=63 partitions, and you might wish to specify around 80.

From v1.92, it is possible to share a disk between ResiDOS and another filesystem (such as FAT). To do this, you need to limit ResiDOS to using a limited number of cylinders on the disk. The command to use is:

```
%REFORMAT 0,15,200
```

This formats the master drive as before, but only the first 200 cylinders will be used by ResiDOS, leaving the rest available for other filesystems. For more information on how this works, and how to create an effective shared disk, see the explanation on the +3e website about sharing disks - http://www.worldofspectrum.org/zxplus3e/sharingdisks.html

Note, however, that the syntax of +3e commands differs from the *ResiDOS* syntax described here.

Once the disk is initialised (you only ever need to do this once) you can use the partition management commands to add some partitions for storing your files.

Switching disks or CompactFlash cards

To switch to using a different hard disk or CompactFlash card, you can use the following command:

```
%DRIVES
```

This will unmap all existing drives and re-detect your disks/cards, automatically mapping any drives back again as it would do on boot. Your BASIC program and all other data will be unaffected by this process, so it's a very useful command for transferring files from one CompactFlash card to another.

Note that once your disk is initialised, *ResiDOS* stores its details (about cylinders, heads, sectors etc) on the disk itself, so when swapping disks or CompactFlash cards there is no need to set these details again.

Entering ResiDOS Commands

ResiDOS commands are easy to enter under any version of BASIC, from the standard keyword-driven 48K BASIC to the more recent versions such as *SE BASIC* and *Gosh Wonderful*, which allow commands to be entered letter-by-letter.

For *all* commands, the `%` characters shown on this website are completely optional. They're only required if the command would otherwise be accepted by BASIC (eg, in most BASICs, `LOAD "name"` would load a program from tape, so you need to use `LOAD %"name"` to load a program from disk or compactflash). For BASICs which use keyword entry (eg standard 48K BASIC), it's also usually convenient to start commands which don't have a keyword with `%`, simply to take the Spectrum out of `K` mode and into `L` mode. On single-character-entry BASICs like *SE BASIC* or *Gosh Wonderful* there's no need to bother with the leading `%`.

Additionally, commands which are not present as Spectrum keywords (such as `LOAD` and `SAVE`) may be entered as upper or lower-case (or a mixture), although they are always shown in capitals on this site. If you are using a single-character-entry BASIC, then they will usually accept keywords in lower-case as well.

Finally, commands can be abbreviated, to save you from having to type in the entire command name. This is done by adding a `"."` character to the partial command name. *ResiDOS* will make a guess at the command desired and fill it in for you.

As an example of the flexibility of the system, here are a number of ways to enter the same command (which lists the partitions on your disk):

```
          %PARTITIONS

PARTITIONS

partitions

%Partitions

%PAR.

p.
```

Entering functions

From v1.81, *ResiDOS* contains its own built-in functions, that can be used as parts of expressions in the same way as other functions. They are entered as if they were user-defined functions, as the `FN` keyword followed by the function name and a list of parameters in brackets. Optionally, you can use a `%` sign between `FN` and the function name. A couple of examples of using *ResiDOS* functions in expressions are:

```
          GOTO FN ERL()+1

          PRINT 100*FN %ERR()
```

Partition Management

Note that some of this section is concerned with creating and managing partitions on IDEDOS drives, so if you are using FAT drives (with the optional FATfs package), then much of it does not apply. However, it is still possible to list partitions on FAT drives with the `%PARTITIONS` command. You should also read the section on mapping partitions to drives, at the end of this page.

The IDEDOS system used by *ResiDOS* allows you to divide your hard disk up into many separate partitions, which can each be assigned to a drive letter and used for storing files. You can have as many partitions as you like on your disk, but due to memory constraints it may not be possible to map them all at one time.

Partitions can be one of two types: "data", for holding files; and "swap", for use as temporary workspace by programs or the operating system. Other types may be added in the future. Note that no programs currently require swap partitions, but there is no harm in creating one for future use.

Commands exist to allow you to create and delete partitions and assign them to drive letters. Unlike some inferior operating systems, this can all be done from BASIC, and there is no need to reset your Speccy at any point!

Creating, deleting and listing partitions

You can create a partition using one of the following commands, which create data and swap partitions respectively:

```
%PARTITION "name",size

%PARTITION "name",size, 2
```

To identify which unit a partition is to be created on, the number can be included at the start of the name, followed by the unit number (eg "0>Documents" or "1>Games"). If you don't specify the unit, then unit 0 is assumed. Names can be up to 16 characters long (this doesn't include the unit identifier) and are not case-sensitive.

The size for a partition can be given in kilobytes (K) or megabytes (MB), with a maximum size of 16 megabytes allowed. A number smaller than 256 is assumed to be a size in megabytes; a number larger or equal to 256 is assumed to be a size in kilobytes.

You can delete a partition, together with any files that are stored on it, using the command:

```
%ERASE "unit>name"
```

In this command, you must specify the unit number, as otherwise it will be assumed you are trying to erase a file. You can use `ERA` or `DEL` instead of `ERASE` if you prefer. Note that a partition can only be deleted if there are currently no drives mapped to it.

You may also rename a partition, using the following command:

```
%REN "unit>name","newname"
```

Again, you must include the unit identifier for this command, otherwise it will assume you want to rename a file, not a partition.

Finally, you may display a list of partitions, in a normal or expanded form, using the following commands:

```
%PARTITIONS

%PARTITIONS+
```

This will show all partitions on all units. Note that if you have other filesystem packages installed (such as FATfs) then some cards/disks will show up more than once, with different unit numbers.

Automatic drive mappings

Normally, when the Spectrum starts up, *ResiDOS* automatically maps all the data partitions it finds to all the available drive letters. If you don't like this behaviour, you can manually specify your own drive mappings. You can also turn off the automatic mapping feature with the command:

```
%AUTOMAP-
```

The feature can be re-enabled at a later time with the command:

```
%AUTOMAP+
```

Mapping drives to partitions

It is also possible to manually map drives to partitions, so that you can choose which drive letters a particular partition uses.

When mapping a drive to a partition, you can make the assignment permanent, which means that *ResiDOS* will automatically make the same assignment every time the Spectrum starts up. This is done by adding the + character to the end of the mapping command. Such permanent mappings happen before the automatic mapping takes place, so you can permanently map your favourite drives and let automatic mapping take care of the remainder.

The commands to map drives to partitions are:

```
%MAP "drive:","name"

%MAP+ "drive:","name"
```

Similar commands are available to remove mappings from drives. Using the + character here means that any permanent assignment associated with the drive will also be removed:

```
%UNMAP "drive:"

%UNMAP+ "drive:"
```

Finally, you can obtain a listing of the current drive mappings with the following command, in which you can optionally specify a stream number for the output (ie stream 3 is often used for the printer):

```
%MAP

%MAP #3
```

Drives, directories, paths and user areas

The location of a file is determined by one or more of the following pieces of information:

- drive (a letter from A-P)
- user area (a number from 0-15)
- path (a list of directories, separated with the / or \ character)

Drives are available for all filesystems. User areas are only supported by IDEDOS drives. Directories and paths are only supported by FAT drives.

You can specify any, all or none of the pieces of information above when naming a file. Any missing information is provided by the current default drive, user area or path, which can be changed using the `%CD` command (see below). When specifying locations, the user area must come first, followed by the drive, followed by a colon (:), followed by the path. For example, the following are all valid locations:
A:
2:
\
0e:
../
docs/letters/
f:\zcode\infocom\zork

Locations can be included with filenames in any command that uses them (eg `SAVE`, `LOAD`, `%CP`; see the following section on using files in ResiDOS for more information).

Setting the default drive, user area or path

You can set the drive which is used by default to any you like; if you don't use this facility, then *ResiDOS* will use A: as the default drive, and 0 as the default user area. To set the default drive, user area or path, use the commands:

```
%CD "c:"

%CD "3:"

%CD "5d:"

%CD "games/"

%CD "e:\data\new"
```

If you wish, you can make your preference permanent by adding the + character to the end of the command. For example, to make E: the default drive, and have *ResiDOS* remember this preference every time you switch on, use:

```
%CD+ "E:"
```

To make a path permanent, you must specify the drive to which it applies. Each drive has its own default path, so you can set permanent paths for multiple different drives. For example:

```
%CD+ "A:games/"

%CD+ "e:\data\new"
```

Creating and deleting directories

You can create directories with the `%MD` (or `%MKDIR`) command:

```
%MD "advents"

%MKDIR "c:\advents\level9"
```

You can delete directories with the `%RD` (or `%RMDIR`) command. They must be empty before they can be removed:

```
%RD "games"

%RMDIR "/data/test"
```

Showing the current location

One final command that is available is `%PWD`, which simply prints the current working directory (ie the current user area, drive and path):

```
%PWD
```

Using files in ResiDOS

Filenames

Filenames consist of an 8-character name, optionally followed by a full stop (`.`) and an extension of up to 3 characters. The characters allowed are:

- letters `A` to `Z` (upper- or lower-case; there is no distinction)
- numbers `0` to `9`
- the following characters: `" # $ ' @ ^ _ { } ~ ``

Although extensions are not needed by BASIC or *ResiDOS*, you may find it helpful to name files in a consistent way, so that all BASIC programs have an extension of `.BAS`, for example.

As well as the filename itself, you can specify a drive letter followed by a colon (`:`) to identify which drive the file is stored on (A: to P:); if this is not done, then the default drive will be assumed (this is drive `A:` unless you change it).

Finally, you can also specify a *user area* number just before the drive (if you don't, then the default user area - usually 0 - is assumed). The user area can be between 0 and 15. This facility allows you a sort of primitive directory system on each drive. See the section on setting the default drive, user area and path above.

Some example filenames are:

```
prog
TEST.BAS
c:puzzle
5e:screen.scr
test21.z__
```

Wildcards

Some commands can have wildcards provided in their filenames. These are used to match multiple files. The following wildcard characters are allowed:

`*` matches any sequence of characters

`?` matches any single character

Wildcard characters can be used in either the name or extension part of the filename. Some examples are:

```
*.*
*.bas
c:test.b??
```

LOADing and SAVEing files

You can load and save files to and from the hard disk or CompactFlash card using the standard `LOAD` and `SAVE` syntax, but adding a `%` sign after the command name. For example:

```
SAVE %"hello.bas" LINE 10

LOAD %"piccy" SCREEN$
```

Note that `DATA` files are not currently supported, although BASIC programs, `CODE` files and `SCREEN$` are.

If you attempt to `SAVE` a file when one already exists with the same name, the existing file will be renamed by changing its extension to .BAK, before the new file is saved. If there was an existing .BAK file of the same name, it will be automatically deleted.

You can also `LOAD` any type of file that was not created by the Spectrum (standard PC text files, for example), using `LOAD %"name" CODE`. If you do not specify a start address, then 32768 will be used as the default. Such files can alternatively be `LOAD`ed using the `SCREEN$` keyword, although this obviously only really makes sense if they are Spectrum screen dumps.

Additionally, snapshot files in the standard .SNA or .Z80 file formats (including those saved by the Task Manager, which are in .Z80 format) can be reloaded using the command:

```
%SNAPLOAD "snapname"
```

This command can load 48K and 128K snapshots, although you will get an "out of memory" error if you try to load a 128K snapshot when in 48K mode. Note also that the command decides which format the snapshot is in from the file extension, so be sure to name your snapshots with the appropriate extension (.Z80 or .SNA).

To aid compatiblity with certain snapshots, there are two additional options with this command. Use "`+`" to make the snapshot loader use a small amount of screen memory (normally it uses the stack area, which is okay in most cases). Use "`-`" to run the snapshot using the Spectrum's built-in ROM rather than the patched one used with ResiDOS. Either or both of these options may be used, eg:

```
%SNAPLOAD+ "snapname"

%SNAPLOAD- "snapname"

%SNAPLOAD+- "snapname"
```

Changing BASIC auto-run status

It is not possible to MERGE files in *ResiDOS*, but you can change whether or not a BASIC program auto-runs, using the `LINE` command:

```
LINE %"filename",line
```

If you specify a line number of 0, then the BASIC program will no longer auto-run.

Cataloguing files

To show a list of all files on the current drive in the current user area, use one of the following:

```
%CAT

%DIR
```

You can also specify a string containing a drive, path and/or user area, to see files in that particular location (otherwise the defaults are assumed). The string can also contain a filename or wildcards, and will display only those files which match the string.

Additionally, you can add the + character at the end of the command to show all files (including system files, which are normally hidden) together with the attributes for each file. Some additional information is also given for BASIC and CODE files saved by *ResiDOS*.

Finally, you can specify a stream number, so that you can (for example) print the catalog listing. (This also applies to all other commands in *ResiDOS* which just produce a listing of information).

Some examples are:

```
%CAT "d:games\"

%DIR "15C:"

%CAT "*.bas"

%CAT+ "e:test.*"

%DIR+ #3;"c:"
```

Deleting files

To erase files from the hard disk, use one of the following commands:

```
%ERASE "filename"

%ERA "filename"

%DEL "filename"
```

You can use wildcards in order to erase many files at the same time, but be careful if using this facility!

Copying files

You can make a copy of file to a new file with a different name, or copy one or more files (using wildcards) to a different drive, user area or directory. If you specify "" as the destination, then the current location is used. Some examples are:

```
%CP "letter.doc","letter2.doc"

%CP "*.rom","b:"

%CP "0c:*.*","3c:"
```

```
        %CP "manic.sna","e:"

        %CP "/snaps/sinclair/c/*.z80",""
```

Renaming files

To rename a file, use:

```
        %REN "filename","newname"
```

This must be a single file (you cannot use wildcards). It is possible to use this command to change the user area for a file (but not the drive), simply by specifying the same name with a different user area.

Changing file attributes

You can add or remove various file attributes from a file (or group of files, using wildcards). The possible attributes are:

- `P` (protected, or read-only)
- `S` (system, or hidden)
- `A` (archive - doesn't have any real use)

Protected files cannot be erased or changed, and system files cannot be copied and are not normally listed in catalogs (only in expanded catalogs). You can add or remove a single attribute at a time, by using a "+" or "-" character, as in the following examples:

```
%ATTRIB "*.*","+p"

%ATTRIB "*.sys","+s"

%ATTRIB "letter.doc","-p"
```

128K RAMdisk emulation

128K Spectrum BASIC includes a number of commands for saving files to the extra memory as a RAMdisk. *ResiDOS* includes emulation of these RAMdisk commands, which work in both 128K and 48K modes.

Setting up the emulated RAMdisk

ResiDOS always uses user area 15 of drive M: to hold RAMdisk files (it uses the hard disk or compactflash card rather than memory, and so can store a lot more files than the 62K or so that the 128K Spectrum can manage). In order to use the RAMdisk commands, therefore, you must first map drive M: to a suitable partition. Here's one suggested way to ensure you always have a 500K RAMdisk available whenever you start the Speccy:

```
        %PARTITION "RAMdisk",500

        %MAP+ "M:","RAMdisk"
```

Alternatively, if you want to share a partition that you already use (this is quite safe, as the RAMdisk always uses user area 15, whereas the normal default user area is 0), remap it to drive M: as follows. If this is the drive you want to use by default for normal file operations (on user area 0, for example), you can also do that as shown.

```
%MAP+ "M:","name"

%CD+ "0M:"
```

If you are map drive M: to a partition on a FAT disk (using the optional FATfs package), then you should be slightly more careful. Since FAT filesystems don't support user areas, using the RAMdisk will affect all files in the current directory of the drive; there is no user area protection, as with IDEDOS partitions. So be careful if you decide to do something drastic like `ERASE !"*.*"`.

Using the emulated RAMdisk

Once it has been set up, you can use all the standard 128K BASIC commands that access the RAMdisk. The only limitations are that you cannot currently load or save DATA files, or perform `MERGE` operations, as these facilities are not yet available in *ResiDOS*.

Some example commands are:

```
CAT !

LOAD !"name"

SAVE !"name" LINE 10

LOAD !"name" SCREEN$
```

Customising ResiDOS

As well as being able to configure *ResiDOS* to automatically set up drive mappings and default drives when you boot up, you can customise it in other ways.

All "permanent" settings are held in a special system file called residos.cfg which is saved on the first available partition of your system. This is a text-based configuration file, so if you wish you can edit it on your PC/Mac, as well as using the normal *ResiDOS* commands.

If you switch cards, then a new config file will be created on the new card. This allows you to keep different settings for different cards (for example, the AUTORUN file will probably need to be different). If you prefer to keep the same settings, you can simply copy the residos.cfg file across from one card to the other.

You can stop the permanent settings (and automapping) from happening if you hold down the CAPS SHIFT key whilst booting the Spectrum. You can also delete all permanent settings in one go by deleting the residos.cfg file.

Custom colours

You can choose custom colours which will be set every time the Spectrum starts up, rather than the usual boring old black on white. To do this, use the normal colour commands, but followed by "%". By using the percentage sign you specify that the current colour scheme (including the new colour you are just setting) will be used whenever you boot. Additionally, you can use the new `ATTR` command to specify all colours together (this is calculated as ink+(paper*8)+(bright*64)+(flash*128)). The commands are:

```
INK % n

PAPER % n

BRIGHT % n

FLASH % n

ATTR % n
```

Note that because the % sets all the current scheme as default, the following examples have the same effect, and set a default colour scheme of yellow ink on blue paper:

```
INK % 6:PAPER % 1

INK 6:PAPER % 1
```

Autorun files

You can also set an "autorun" file. This must be a BASIC program (which can do whatever you like) which will run automatically on boot, or just whenever you enter the special `%AUTORUN` command.

To set an autorun file, use one of the following commands:

```
%AUTORUN "filename"

%AUTORUN+ "filename"
```

You may specify a drive and/or user area with the filename (otherwise the defaults will be assumed). These two commands differ, because with the second case, the autorun file will automatically be loaded every time you reset (or `NEW`) your Spectrum. The first command stores the filename, but will not load it unless you enter the following command:

```
%AUTORUN
```

Note that you can hold the CAPS SHIFT key down when booting to stop drives from being automatically mapped, so any autorun file will not be able to be loaded.

You can disable the autorun file with the following command:

```
%AUTORUN-
```

Processor speed

Some systems (only ZX-Badaloc, at the time of writing) allow the Z80 processor speed to be increased above the standard 3.54MHz. The following commands can be used to select the system speed:

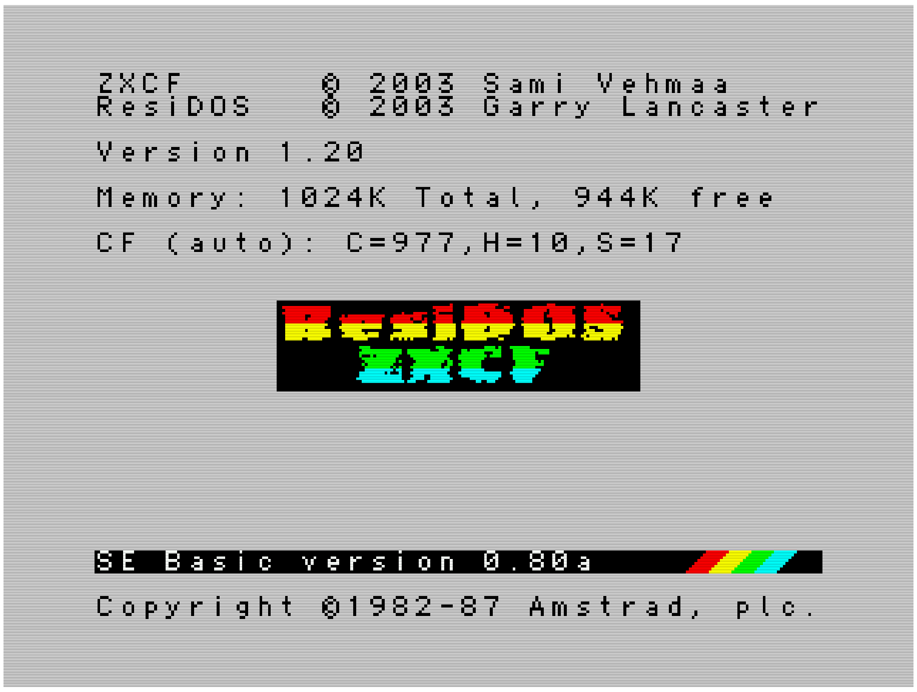
```
%SPEED n

%SPEED+ n

%SPEED- n
```

Each command sets the current speed to n (discussed below). If the "+" option is used, then the speed is made permanent and takes effect every time the Spectrum starts up. If the "-" option is used, then any permanent speed setting is removed, and the Spectrum reverts to the default system speed every time it starts up.

The value of n depends on the system. For ZX-Badaloc it is as follows:
0 - 3.54MHz
1 - 7MHz
2 - 14MHz
3 - 21MHz
4 - 7.08 MHz, 100Hz interrupts, 100Hz VGA
5 - 14.16 MHz, 100Hz interrupts, 100Hz VGA
6 - 28.32 MHz, 100Hz interrupts, 100Hz VGA
7 - 42.50 MHz, 100Hz interrupts, 100Hz VGA (not recommended at time of writing)

This command may be executed on all *ResiDOS* systems, although it will have no effect on anything except ZX-Badaloc.

Using optional packages, alternative BASICs and Interface 2 ROMs in ResiDOS



```
ZXCF      © 2003 Sami Vehmaa
ResiDOS   © 2003 Garry Lancaster

Version 1.20

Memory: 1024K Total, 944K free

CF (auto): C=977,H=10,S=17
```

```
SE Basic version 0.80a

Copyright ©1982-87 Amstrad, plc.
```

The RAM in your interface can be used to store alternative versions of BASIC, or Interface 2 ROM images. Once they have been added, you can start them up at any time using a simple command.

Additionally, *ResiDOS* can be enhanced by loading optional packages which provide additional features such as new commands and access to other types of filesystems, such as the common FAT-16 type used on PCs. The latest packages can be downloaded directly from the *ResiDOS* homepage: http://www.worldofspectrum.org/residos/

Installing ROMs

You can install ROMs in one of two ways:
- directly from a file stored on your hard disk or card, or
- by first loading the ROM into memory (from tape or hard disk)

Suitable ROMs for installation include *ResiDOS* specific packages, alternative BASICs (such as Gosh Wonderful, SE BASIC or Sea Change) or Interface 2 ROMs (such as those originally released by Sinclair, or unreleased prototypes from Parker which have recently come to light).

Installing ROMs from a file

If you have copied a ROM to disk or card, then install it with the following commands:

```
CLEAR 32767

%INSTALL "filename",n
```

The first command reserves space for the installation process, and the second performs the installation. The name of the ROM kept by ResiDOS will be the filename (excluding any extension after a "."). The number, n, should be any of the following values, added together as appropriate:

0: for a package, or an Interface 2 ROM (or any other ROM that does not require ResiDOS support)
1: for a BASIC ROM
16: to install the ROM into the writeable "RAM" part of the interface memory (the non-writeable "ROM" part is used by default)
32: to disable the NMI button for this ROM (only affects BASIC ROMs)

If you don't specify a value, then the default of 0 will be used. If you are installing a package, then any value except 0 will be ignored, since packages include their own information telling ResiDOS how they should be installed.

Note that it is generally advisable to only store BASIC ROMs into the non-writeable "ROM" part of the memory, since unless they have been specifically patched they tend to overwrite themselves. However, if you are running short of ROM space on a ZXATASP or DivIDE Plus interface, you may specify RAM instead. This is not a concern on other interfaces, since the memory can be configured automatically by ResiDOS to be ROM or RAM as required.

A couple of examples are:

```
CLEAR 32767

%INSTALL "SEBASIC.ROM",1

%INSTALL "c:jetpac.rom",0

%INSTALL "FATfs.pkg"
```

Installing ROMs already loaded into memory

An alternative installation method is to load the ROM image into memory, and then install it directly from there. Since this allows you to load from tape, it is handy for installing packages like FATfs when you don't have access to the disk.

To use this method, first `CLEAR` to a suitable address, and then load in the ROM to above this address. For example:

```
CLEAR 39999

LOAD "FATfs" CODE 40000
```

Then the ROM can be installed by using the following form of the `%INSTALL` command:

```
%INSTALL "name"@address,n
```

As before, the "n" value is optional.

To install the package just loaded in the previous example, use:

```
%INSTALL "FATfs"@40000
```

Managing alternative ROMs

You can see a list of all the installed ROMs in your interface by using the following commands (the second version includes hidden ROMs, used by the system but not selectable by the user):

```
%ROMS

%ROMS+
```

The names of the ROMs (except built-in system ones) will be the same as the filename used when loading them, except that any extension is ignored. For example, the ROMs loaded in the previous section would be named "SEBASIC" and "jetpac".

You can uninstall any ROM using the following command (this only removes it from the interface memory, not from the hard disk):

```
%UNINSTALL "name"
```

Using alternative ROMs

Once you have installed some formats as described in the previous section, they are available for instant use whenever you switch on the Spectrum. To activate a particular format, use the command:

```
%ROM "name"
```

For example, having installed the ROMs in the previous section, we could use either of them like this:

```
%ROM "SEBASIC"

%ROM "jetpac"
```

If you specify an Interface II ROM, it will immediately start. Pressing the reset button will reset the ROM. To return to *ResiDOS* you will need to either reset your interface and the Spectrum together, or switch off the power completely and then switch the system back on.

If you are activating a BASIC ROM, the Spectrum will restart using that BASIC. You can then use *ResiDOS* commands as normal, and the additional features of the particular BASIC you have chosen.

The new BASIC will remain as the default one even if you switch off the Spectrum and turn back on again. To return to the original BASIC, you can simply select the "Standard" ROM (which is automatically created as part of the installation process), for example:

```
%ROM "Standard"
```

Alternatively, hold down the CAPS SHIFT key while the Speccy is booting up, and the standard ROM will be automatically re-selected.

One final feature is the ability to switch off the interface memory altogether and use your Spectrum's built-in ROM, without having to unplug the interface or set the upload or disable jumper (this is useful for 128K Spectrums and +3s). To do this, use the command:

```
%ZX
```

Spectrum Modes with ResiDOS

ResiDOS can operate in a number of different modes, depending upon the configuration of your Spectrum. The two basic modes of operation are:
128K mode
48K mode

When *ResiDOS* starts up, it displays the current mode on the title screen. If you have a 48K Spectrum, then 48K mode is the only possibility. For 128K Spectrums, ResiDOS will usually start up in 128K mode.

128K Mode

In *ResiDOS*, 128K mode behaves in the same way as what is often referred to as USR 0 mode; this mode is usually required by demos, and modern programs from Russia and other countries. (Normally, 128K Spectrums enter this mode by typing `USR 0` in 128K BASIC).

This mode gives access to all the special hardware of the 128K machines (the sound and extra memory), but without access to the 128K BASIC editor and commands. Under *ResiDOS*, the 128K RAMdisk commands and SPECTRUM command are also available (in fact, they also work in 48K mode).

This mode is recommended for loading 128K programs. Some 128K programs will not load in this mode (since they check to see if 128K BASIC is active). In such cases, you must disable *ResiDOS* and restart into 128K BASIC with the `%ZX` command.

If you wish to use the single key 128K BASIC editor, you will need to install either *SE BASIC* or *Gosh Wonderful BASIC* from within *ResiDOS* - see the section on installing ROMs above.

IMPORTANT NOTE: The current version of the Task Manager does not understand that some tasks might run in 128K mode and use the whole 128K of memory, and will not save 128K snapshots. It is possible to have a single task loaded which uses the extra memory of the 128K Spectrum, and switch between that and other tasks. However, if two 128K programs are loaded, there will be problems. This will be addressed in a future version.

48K mode

Some games will not work correctly in 128K mode. For these you can use the `SPECTRUM` command to force the computer into 48K mode until the next reset.

If you always want to work in this mode, the command `SPECTRUM+` will make 48K mode active every time the computer is switched on. To disable this mode, and re-enter 128K mode on the next reset, use the `SPECTRUM-` command.

Disabling ResiDOS

It is possible to temporarily disable *ResiDOS* and then re-enable it, whilst leaving any program in memory unchanged. This can be useful if you have some piece of hardware which is incompatible with *ResiDOS*, such as Microdrives, or a Wafadrive etc. It is also useful for switching from the %ZX mode's 128K BASIC back into *ResiDOS* without having to switch off.

On the Sinclair ZX Spectrum +3 machine, this can be very useful in order to allow you to disable *ResiDOS* temporarily, to load a program from a +3 disk - once it loads, you can re-enable *ResiDOS* and then save the program to compact flash card.

The commands to do this are:

DivIDE Plus

Disable ResiDOS: `OUT 23,0` or use: `%ZX`

Re-enable ResiDOS: `OUT 23,104`

BREAK and error-trapping

ResiDOS contains facilities to detect BREAK and error conditions, and have them handled by your own BASIC program, instead of simply having an error message produced. This allows you to write more professional-looking BASIC programs, which can't be broken in to, and which can cope with expected errors (such as being unable to find a file requested by the user, for example).

You are advised to `SAVE` your work before running any program with error-trapping facilities, as if you have made a mistake somewhere, much trouble can ensue!

BREAK-trapping

You can prevent the user from BREAKing into your program with the `%ONBREAK` command. This has two forms; to stop BREAK from being effective, use:

```
%ONBREAK THEN CONTINUE
```

The `THEN` keyword is optional, and used to make entering the `CONTINUE` keyword easier on normal keyword-driven BASICs. On single-letter-entry BASICs, there is no need to use it.

To restore the normal operation of BREAK, use:

```
%ONBREAK OFF
```

(You could include the `THEN` keyword again, but it would be more trouble than it's worth to enter it).

Note that BREAK-trapping prevents the user from stopping the program in any of the following ways:

- pressing CAPS SHIFT + BREAK
- pressing SPACE or "n" at a "scroll?" prompt
- entering `STOP` in an `INPUT` statement

It does *not* prevent the user from BREAK-ing in by pressing CAPS SHIFT + 6 in an `INPUT LINE` statement, so you are advised to use a straightforward `INPUT` instead.

Error-trapping

All other errors (except for "OK", "Program finished" and "STOP statement") can be trapped using the `%ONERR` command. This takes any of the following forms, allowing you to transfer control to an error routine, or just call a subroutine before continuing with the program after the point where the error occurred:

```
%ONERR THEN GOTO line

%ONERR THEN GOSUB line

%ONERR OFF
```

Your error-handling routine can use any of the following new functions to determine what the error was. They provide the error number, line and statement respectively:

```
FN ERR()
```

```
FN ERL()
```

```
FN ERS()
```

Additionally, the following command makes the last trapped error occur as normal (as if it had not been trapped):

```
%REPORT
```

As an example, here is a program that turns on error trapping, causes an error and lets the error routine output details of the error:

```
  10 %ONERR GOTO 9000
  20 PRINT "Hello"
  30 PRINT: PRINT f
  40 PRINT "Never get here"
8999 STOP
9000 PRINT "Error code was ";FN ERR()
9010 PRINT "Error line was ";FN ERL()
9020 PRINT "Error statement was ";FN ERS()
9030 PRINT "Now causing error..."
9040 %REPORT
```

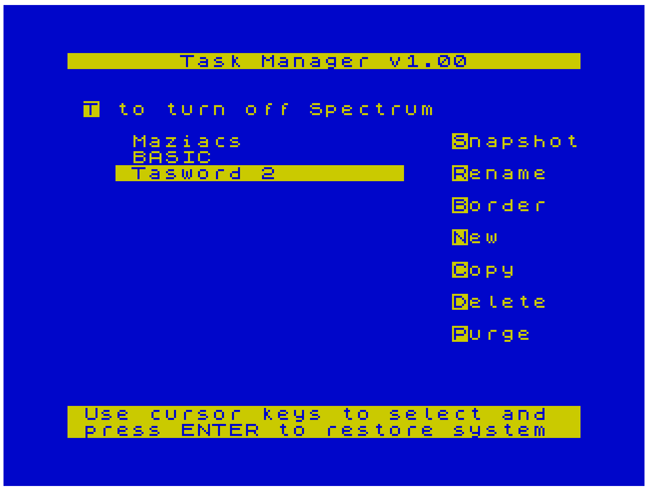
Warning!

There is a known flaw in *ResiDOS* with error-trapping. The error-trapping variables are shared between tasks, rather than being saved with each task individually. This means that changing error-trapping in one task will affect all the others. I hope to resolve this in a new release of the task manager in the near future.

Packages

There is a host of various packages available from the ResiDOS home page - http://www.worldofspectrum.org/residos which can be downloaded and installed as part of the ResiDOS firmware. The most useful packages are detailed below (the Task Manager, the FATfs package, the TapeIO package, and the channels package).

The ResiDOS Task Manager



```
         Task Manager v1.00

 T to turn off Spectrum

     Maziacs               Snapshot
     BASIC
     Tasword 2             Rename
                           Border
                           New
                           Copy
                           Delete
                           Purge

 Use cursor keys to select and
 press ENTER to restore system
```

Overview

The Task Manager is an optional package that you can install on ResiDOS. Note that the current version (v1.00) only operates with 48K tasks, so you cannot save snapshots of 128K games. You should also only attempt to run one 128K game as a task at a time, otherwise the extra Spectrum memory will be overwritten by the 2nd 128K game. This limitation will be removed in a future release.

When you press the NMI button on your interface, the Spectrum immediately stops whatever it was doing (whether you were in BASIC, a word processor, a commercial game or whatever) and enters the Task Manager.

The Task Manager screen shows a list of the current tasks running on your Spectrum, together with the menu options that are available to you. When you first enter the task manager, the task list will consist of just a single task, named "Task 0", which is whatever you were doing at the time. To return to it, simply press ENTER.

Using the RAM in your interface, you can store several tasks at the same time, and switch between them instantly from the task manager screen. The number of tasks you can have running at once depends on the amount of RAM in your interface. Since ZXATASP interfaces have their memory strictly divided between ROM and RAM, they can run fewer tasks than a different interface with an equivalent amount of memory. The maximum number of tasks allowable is 16 (this is only possible on a 1Mb ZXCF interface).

The menu options described below allow you to manage your tasks quickly and easily.

Managing Tasks

The options available are shown on the task manager screen. Simply press the highlighted letter to select the option.

Cursor keys (up and down)
: Use these to move the highlight up and down the task list.

`ENTER`
: Re-enters the highlighted task.

`N`ew
: Creates a new task, activates it and resets the Spectrum into BASIC. (Don't worry, all the other tasks are still there!)

`C`opy
: Makes a copy of the highlighted task.

`D`elete
: Deletes the highlighted task.

`P`urge
: Deletes all tasks *except* the highlighted task.

`R`ename
: Renames the highlighted task. Task names are up to 16 characters long.

`B`order
: Sets the border colour for the highlighted task. Since it isn't possible for *ResiDOS* to know the border colour, it makes a guess which is sometimes wrong. You can change that here before saving a snapshot.

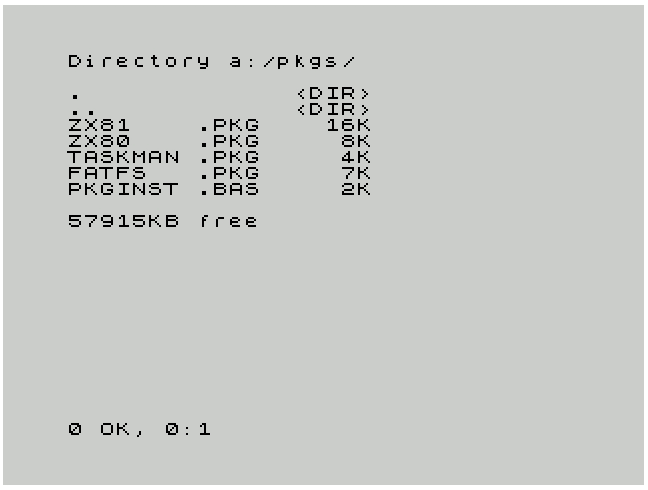
`S`napshot
: Saves a snapshot of the highlighted task onto the hard disk in the standard .Z80 file format. This can be reloaded later using the `%SNAPLOAD "filename"` command (which can load any 48K or 128K snapshot in .Z80 or .SNA format).

: **NOTE:** When prompted to enter the name of the snapshot to create, please remember to add the suffix .z80 to the filename, otherwise you will not be able to reload your snapshot.

Other options

`T`urn off Spectrum
: This takes you to a special "shutdown" screen, from which you can turn off the Spectrum in safety. When you switch back on, the *Task Manager* will start up again, and you will be able to continue exactly where you were, with all your tasks still running.

The FATfs Package



```
Directory a:/pkgs/

.               <DIR>
..              <DIR>
ZX81     .PKG     16K
ZX80     .PKG      8K
TASKMAN  .PKG      4K
FATFS    .PKG      7K
PKGINST  .BAS      2K

57915KB free




0 OK, 0:1
```

Although an optional package, FATfs is installed as part of the DivIDE Plus ResiDOS implementation.

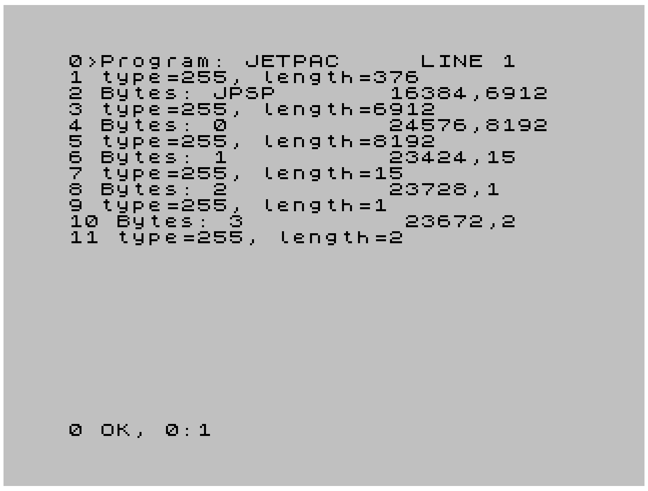
This package adds support for reading and writing to FAT16-formatted disks to your ResiDOS system. This is incredibly useful, since all PCs and Macs (and even some PDAs and digital cameras etc) can use this format of disk. This means copying files (including other packages, upgrades to ResiDOS, snapshots and Infocom games for ZXZVM) between your PC and Speccy becomes extremely easy.

Once installed, any FAT16-formatted disks can be accessed using `LOAD`, `SAVE` and all the other ResiDOS commands.

The current version has the following limitations, which will be addressed in a future release:

- no support for FAT32 or FAT12 disks
- no support for long filenames
- partitions cannot be created, renamed or deleted by ResiDOS

TapeIO Package – Load / Save .tap and .tzx Files



```
0>Program: JETPAC      LINE 1
1 type=255, length=376
2 Bytes: JPSP       16384,6912
3 type=255, length=6912
4 Bytes: 0          24576,8192
5 type=255, length=8192
6 Bytes: 1          23424,15
7 type=255, length=15
8 Bytes: 2          23728,1
9 type=255, length=1
10 Bytes: 3          23672,2
11 type=255, length=2





0 OK, 0:1
```

The TapeIO Package

The TapeIO package is an optional package that you can install on ResiDOS which gives you the ability to load and save directly to .TAP and .TZX files on your flash card or hard disk as if it was a real tape. This means that you can take advantage of the thousands of .TAP and .TZX emulator files available and use them with your Spectrum.

The most important commands to use are `%TAPEIN` and `%TAPEOUT`. Each of these takes the name of a .TAP or .TZX file. Thereafter, and standard LOAD commands that would normally access the tape will be redirected to whichever .TAP/.TZX file you specified in the `%TAPEIN` command. Similarly, any standard `SAVE` commands that would normally access the tape will be redirected to the .TAP/.TZX file specified in the `%TAPEOUT` command. In order to redirect access to tape, simply use the `%TAPEIN` or `%TAPEOUT` command without a filename.

For example, to load Jetpac from a jetpac.tap file, use the following commands:

```
%TAPEIN "jetpac.tap"
LOAD ""
```

To create a .tzx file containing your latest masterpiece, so that it can be loaded by an emulator, use commands such as:

```
%TAPEOUT "myprog.tzx"
SAVE "My Program" LINE 10
%TAPEOUT
```

Controlling the tapepointer

Some more useful commands that are available are `%TAPELIST` and `%TAPEWIND`.
The first of these lists out the contents of the current .TAP/.TZX file being used for input (ie the last one specified in a `%TAPEIN` command). It also shows which is the next block that will be `LOAD`ed, with a cursor (similar to the cursor in a BASIC program listing).

With the `%TAPEWIND` command, you can "wind" the tape pointer forwards (using a positive number) or backwards (using a negative number) by a number of blocks, or to a specific block number (using the "=" option). If you do not specify any number, then the input tapefile will be rewound to the start.

For example, use the following commands to wind the tape in different ways and see the result on the tapelisting:

```
%TAPEWIND 3

%TAPELIST

%TAPEWIND -2

%TAPELIST

%TAPEWIND=6

%TAPELIST

%TAPEWIND
```

Finally, you can also change the behaviour when the last file is loaded from the input .TAP/.TZX file. Normally, when this happens the tapefile is "ejected" and subsequent `LOAD` commands use the real tape again, until another `%TAPEIN` command is used. However, if you would prefer the tapefile to be "rewound" to the start, then use the following command:

```
%TAPEMODE 1
```

To revert to the default behaviour (ejecting the tapefile), use:

```
%TAPEMODE 0
```

Transferring files between disk and .TAP/.TZX files

It's also easy to transfer files from within .TAP and .TZX files to separate files on your flash card or hard disk. There are commands available to transfer files with headers (`%TAPEGET` and `%TAPEPUT`) and also headerless files (`%TAPEBGET` and `%TAPEBPUT`). For details, see the full command reference below.

Troubleshooting

All .TAP files should work correctly with *ResiDOS*, as well as some .TZX files which use the standard Spectrum loading system. However, any .TZX files using speedloading or other custom loaders will not work. If a game appears to stop loading from a .TZX file, then it is likely to have a custom loader. Using the `%TAPELIST` command on such a file should show which blocks *ResiDOS* can "see" and help to show if it is a non-standard loader program.
If a CODE file which might overwrite the ROM area or the stack is encountered, then TapeIO will load this byte-by-byte in "slow" mode, to ensure that ResiDOS data is not corrupted. In such cases, the border will flash multicoloured during loading.

If you have a 128K Spectrum and find a game that fails to load, please try again after selecting 48K mode (with the `%SPECTRUM` command). Many games will only load in 48K mode by design (and would have included instructions on the cassette inlay to switch to 48K mode before loading).

TapeIO Command Reference

Here is a complete list of all the commands supported by the *TapeIO* package, including several options and variants that aren't described above. Please read carefully to get the most out of this package.

`%TAPEIN "`*`filename`*`"`

Redirect all tape `LOAD` commands to use the .TAP or .TZX file specified.

`%TAPEIN`

Redirect all tape `LOAD` commands back to physical tape.

`%TAPEOUT "`*`filename`*`"`

Create a new .TAP or .TZX file, and redirect all tape `SAVE` commands to it.

`%TAPEOUT+ "`*`filename`*`"`

Open an existing .TAP or .TZX file, and redirect all tape `SAVE` commands to it (appending to the end of the file).

`%TAPEOUT`

Redirect all tape `SAVE` commands back to physical tape.

`%TAPELIST`

List the contents of the current input tapefile.

`%TAPELIST #`*`n`*

List the contents of the current input tapefile to stream "n".

`%TAPEWIND `*`n`*

Wind the tapepointer by "n" blocks (positive is forwards, negative is backwards).

`%TAPEWIND=`*`n`*

Wind the tapepointer to block number "n".

`%TAPEWIND`

Rewind the tapepointer to the start of the tapefile.

`%TAPEGET "`*`diskname`*`"`

Copy the next file with a header from the input tapefile onto disk, using the specified name for the disk file.

`%TAPEGET`

Copy all files with a header from the input tapefile onto disk, translating names to fit in the 8.3 name format.

`%TAPEBGET "`*`diskname`*`"`

Copy the next block (could be a header or data block) from the input tapefile onto disk, using the specified name for the disk file.

`%TAPEBGET+ "`*`diskname`*`"`

Copy the next block (could be a header or data block) from the input tapefile onto disk, using the specified name for the disk file. Copy the entire block, including the block type and parity byte (which are normally stripped out).

`%TAPEPUT "`*`diskname`*`"`

Copy a disk file into the current output tapefile, complete with an appropriate header.

`%TAPEBPUT "`*`diskname`*`"`

Copy a disk file into the current output tapefile as a headerless block (of type 255).

`%TAPEBPUT "`*`diskname`*`", `*`n`*

Copy a disk file into the current output tapefile as a headerless block (of type "n").

`%TAPEBPUT- "`*`diskname`*`"`

Copy a disk file into the current output tapefile as a headerless block, but don't add a type byte or parity byte.

`%TAPEMODE `*`n`*

Set the behaviour for what happens when the end of the input tapefile is reached (0=eject, 1=rewind).

`%TAPEMODE+ `*`n`*

Set the behaviour for what happens when the end of the input tapefile is reached (0=eject, 1=rewind). Make this behaviour permanent.

`%TAPEMODE- `*`n`*

Set the behaviour for what happens when the end of the input tapefile is reached (0=eject, 1=rewind). Remove any permanent behaviour.

Extended Channels

ResiDOS provides built-in support for accessing new extended channels in addition to the three standard ("K", "S" and "P") channels. Streams can be opened and closed using the normal `OPEN #` and `CLOSE #` commands (and ResiDOS also fixes the bug with the Spectrum's `CLOSE #` command so that it is safe to use). Machine-code programmers also have comprehensive access to these extended channels (see the programming section for details).

Once a stream is open to a channel, you can send output to it and receive input from it using any standard Spectrum BASIC command that allows a stream number to be specified. Additionally, most ResiDOS commands that display output may be redirected to a stream, allowing you to capture and manipulate such information. For example, any of these commands are allowed:

```
PRINT #5;"Some text"

INPUT #7;a$

LIST #4;9000

LPRINT #4;x$

LLIST #10

CAT #6;"*.bas"

%roms+ #15

%partitions #8

%pwd #2

%map #4

%tapelist #9
```

Additionally, ResiDOS provides two additional functions and a command to read/write the current position in the stream, and to read the size of the stream (not all channels support these operations):

```
FN ptr#(stream)

FN ext#(stream)

POINT #stream, position
```

ResiDOS does not actually provide any extended channel types itself; these may be provided by various optional packages. The most important of these is the channels package, which you are encouraged to download and install.

Commands and functions for streams and channels

ResiDOS uses the following commands for manipulating streams and channels:

```
OPEN #stream, c$
```

This command opens a stream to a channel. The value of stream can be any number from 0 to 15; if one of the standard streams (0-3) is specified, then its original assignment is replaced with the new channel specified. The value of `c$` is either a single character representing one of the Spectrum's standard channels ("K", "S" or "P"), or a string specifying a new extended channel.

The exact form of the channel specifier is documented separately for each new channel provided, but generally it consists of a letter followed by a ">" character; optionally, there may be further parameters inside the string (usually separated by commas). If you want to calculate a value for a parameter, this is perfectly possible - just remember to use the `STR$` function to build up the string.
For example, to open stream 5 to a region of memory specified by the variables `addr` and `len`, use the following command:

```
OPEN #5, "m>" + STR$ addr + "," + STR$ len
```

```
CLOSE #stream
```

This command closes a stream, flushing any output that might be buffered. If the stream was one of the standard streams (0-3), then it is re-assigned to its default channel (ie "K" for streams 0 & 1, "S" for stream 2 and "P" for stream 3).
For example, to close stream 5:

```
CLOSE #5
```

```
FN ptr#(stream)
```

This function returns the current position of the the pointer in the stream specified. Not all channels support this function; see the documentation for individual channels.
For example, to find the position of the pointer in stream 5:

```
LET pos=FN ptr#(5)
```

```
FN ext#(stream)
```

This function returns the extent (size) of the stream specified. Not all channels support this function; see the documentation for individual channels.
For example, to find the size of stream 5:

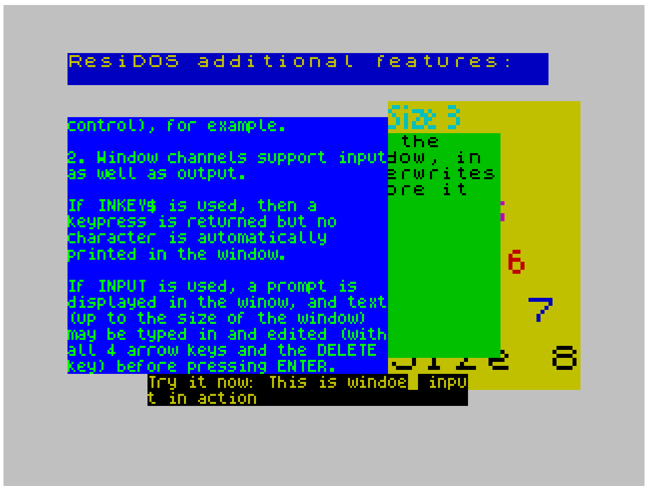
```
LET size=FN ext#(5)
```

```
POINT #stream, pos
```

This command sets the current position of the pointer for the stream specified. Not all channels support this function; see the documentation for individual channels.
For example, to set the pointer to the start of stream 5:

```
POINT #5, 0
```

Channels Package



```
ResiDOS additional features:

control), for example.

2. Window channels support input
as well as output.

If INKEY$ is used, then a
keypress is returned but no
character is automatically
printed in the window.

If INPUT is used, a prompt is
displayed in the winow, and text
(up to the size of the window)
may be typed in and edited (with
all 4 arrow keys and the DELETE
key) before pressing ENTER.
Try it now: This is windoe  inpu
t in action
```

The channels package provides a number of new channels for use under ResiDOS. Most of these are compatible with the extended channels of the ZX Spectrum +3e, although there are also several important additions.

After installing the package, in order to make use of the new channels, you will need to read the information on using extended channels in ResiDOS.

The channels provided are:

- File access ("I>", "O>" and "U>")
- String variables ("V>")
- Memory regions ("M>")
- Text windows ("W>")
- Null ("Z>")

- File channels

  Streams may be opened to any file in one of three different access modes: input (using the "I>" channel), output (using the "O>" channel) and update (using the "U>" channel).

  In input and update modes, the file must already exist, or an error will occur. In output mode, the file is deleted if it already exists, and then a new file is created. Once the file is open, it can be input from (if opened in input or update modes) or output to (if opened in output or update modes). Because of file buffering, it is essential to `CLOSE #` any file channel before finishing, otherwise data may be lost.

  The channel specifier for all file channel types requires the filename of the file to follow the ">" character.

  File channels support all the pointer operations (`FN ptr#()`, `FN ext#()` and `POINT #`).

  It should be noted that files on IDEDOS disks are always stored as a number of 128-byte "records" and so you may read rubbish at the end of a file if it is not an exact multiple of 128 bytes in length. To avoid this problem, you should write some code or "signature" at the end of your file which you can detect when reading it back. Alternatively, you could write the number of bytes or entries in your file at the start.

This problem does not exist for files on FAT disks, but it would be good practice to do this anyway, since it will then allow your program to be used with files on both FAT and IDEDOS disks.

Examples

```
OPEN #4, "o>test.txt"
```

Creates a file named "test.txt" on the default drive for output through stream 4.

```
OPEN #5, "i>a:test2.txt"
```

Opens a file named "test2.txt" on drive A: for input through stream 5.

```
OPEN #6, "u>c:/data/list.txt"
```

Opens a file named "list.txt" in the "data" directory of drive C: for update through stream 6.

- Variable channels

  The "V>" channel can be used to direct output to or input from a string variable, which can be easily manipulated within a BASIC program. This would allow you to (for example) examine disk catalogues in your BASIC program, or make an auto-running game demo (by inputting from a string containing set keystrokes).

  The channel specifier for the variable channel type requires the name of the string variable to follow the ">" character.

  Variable channels support all the pointer (`FN ptr#()`, `FN ext#()` and `POINT #`).

  The string specified must be a character array with a single dimension, large enough to hold the maximum amount of data you expect to have to deal with.

  Example

  Here is an example program that outputs a list of installed ROMs to a string.

```
10 DIM a$(1000)
20 OPEN #8, "V>a$"
30 %roms+ #8
40 LET l=FN ptr#(8)
50 CLOSE #8
60 PRINT "Assignment length is ";l
70 PRINT "List is:"
80 PRINT a$( TO l)
```

- Memory channels

  The "M>" channel can be used in a very similar way to the variable channels. However, as it is a fixed memory region, it is more suitable for use by machine-code programs.

  The channel specifier for the memory channel type requires the address and length of the memory region (separated by a comma) to follow the ">" character.

  Memory channels support all the pointer operations (`FN ptr#()`, `FN ext#()` and `POINT #`).

Example

Here is an example program that outputs a disk catalogue to memory and then runs a machine-code routine to process it:

```
10 CLEAR 29999
20 OPEN #7, "M>30000,1000"
30 CAT #7
40 LET x=USR myroutine
50 CLOSE #7
```

- Window channels

The "W>" window channels are the most complex of the extended channels currently available on ResiDOS. Two different types of window, with very different behaviours, are available:

+3e-compatible windows

These windows are the default type, and provide the same facilities as those provided by the ZX Spectrum +3e (except that, under ResiDOS, input is also supported). These windows support Spectrum-style control codes, so that you can use normal `PRINT` items such as `INK`, `PAPER` and `AT` (amongst others) in your `PRINT #` commands, they also support Spectrum tokens and graphics characters (so that you can use `LIST #` directly, for example).

VT52-compatible windows

These windows are created when you specify a character set address of zero (see below for details). These windows broadly emulate a subset of the VT52 terminal, and so use completely different control codes to the +3e-compatible windows. They are intended for applications such as CP/M and internet application console windows. They support only the standard ASCII character set, not Spectrum tokens or graphics characters. Additionally, they do not support line input; only (non-echoed) character input is provided.

The channel specifier for the window channel type requires the following values (separated by commas) to follow the ">" character: top line (0-23), leftmost column (0-31), height (1-24), width (1-32), and optionally character size (3-8) and character set address. If no character size is specified, the default is 8. If a character set address is given, then this is used instead of the built-in fonts; this allows you to use nice fonts such as those provided with art programs and adventure games. If the character set address is given as zero (0), then a VT52-compatible window is created instead of a +3e-compatible window.

Note that the character size has no bearing on the way the window is defined, but it does affect the number of "actual" columns you have available. For example, the following defines a window the size of the entire screen; but because a character size of 5 is specified, the number of characters that can be printed in the window at any time is 24x51:

```
OPEN #5,"w>0,0,24,32,5"
```

Window channels do not support the pointer operations. However, the "set position" operation (`POINT #`) can be used to change the region of memory to which window output is sent. Normally, this is the standard Spectrum screen address (16384), but it can be set to any 8K boundary. This is mostly useful for machine-code programmers, as it allows them to perform window output directly to the 128K Spectrum's alternate screen at 49152 in page 7, for example.

+3e-compatible windows

When `PRINT`ing to +3e-compatible windows, you can use many of the same control functions as you can with the normal screen. For example: ' (apostrophe; start a new line), , (comma; start a new column), `TAB`, `AT`, `INK` , `PAPER`, `FLASH`, `BRIGHT` , `INVERSE`, `OVER`. Of these, only `AT` behaves slightly differently; it takes y to be a pixel line and x to be an actual character column.

When first defined, windows are in non-justified mode, but they can be set to be left-, full- or centre-justified. Note that in justified mode, some features and control codes cannot be accessed, so you may need to switch back to non-justified mode to use them.

Input is supported in +3e-compatible windows. If you use `INKEY$ #` , then the keyboard is simply scanned and a character returned without anything being output to the window. If you use `INPUT #`, then a cursor is added to the window at the current position. The user can then input any text desired, using the left and right arrows to move along the text input so far, or the up and down arrows to move to the start or end of the text. The `DELETE` key deletes the character to the left of the cursor, and the `ENTER` key completes the input. Depending upon memory available, the entire size of the window can be used in the input, although care is taken to ensure that no input character is ever scrolled off the top of the window. An absolute maximum of 255 characters is allowed in the input line.

Control Codes

A complete list of control codes follows; these codes can be sent to a window by `PRINT`ing them using the `CHR$` function. If a code is preceded by (j) then it will be ignored if issued in justified mode (however, their settings will still be taken into account; for example, you can justify double-width text, but you must set it before entering justified mode). If a code is preceded by (e) then it can only be used in justified mode if the "embedded codes" feature has been set (which requires more memory to be allocated for the channel).

0	Turn justification off
1	Turn justification on
2	Save current window contents
3	Restore saved window contents
4	Home cursor to top left
5	Home cursor to bottom left
(j) 6	Tab to left or centre of window (`PRINT` comma)
7	Scroll window
(j) 8	Move cursor left
(j) 9	Move cursor right
10	Move cursor down
11	Move cursor up
(j) 12	Delete character to left of cursor
13	Start new line (`PRINT` apostrophe)
14	Clear window to current attributes
15	Wash window with current attributes

(e) 16, n	Set `INK n` (0-7)
(e) 17, n	Set `PAPER n` (0-7)
(e) 18, n	Set `FLASH n` (0-1)
(e) 19, n	Set `BRIGHT n` (0-1)
(e) 20, n	Set `INVERSE n` (0-1)
(e) 21, n	Set `OVER n` (0-1)
(j) 22, y, x	Set cursor to pixel line y, character size column x
(j) 23, n	`TAB` to character size column n
(e) 24, n	Set attributes n (0-255)
(j) 25, n	If n=1, then characters 165-255 will be printed as UDGs, with data located at the end of the standard UDGs. If n=0, then characters 165-255 will be printed as BASIC keyword tokens (the default).
(e) 26, n	Fill window with byte n. Attributes are affected, but not cursor position.
(e) 27, n	Fill window with character n. Attributes and cursor position are affected.
(j) 28, n	Set double width (n=1) or normal width (n=0).
(e) 29, n	Set height n (0=normal, 1=double, 2=reduced, 3=double reduced)
30, n	Set justification mode (0=left, 1=centred, 2=full)
(j) 31, n	Allow embedded codes in justified mode if n=1.

VT52-compatible windows

VT52-compatible windows behave very differently from +3e-compatible windows, and are most useful for applications such as CP/M and internet console applications. They do not support line input (`INKEY$#` and `INPUT #` are allowed, but are implemented as non-echoed character input) or non-ASCII characters (such as the Spectrum tokens and graphics characters). By default, these windows operate in a "non-wrapping" mode (when the end of a line is reached, text does not wrap automatically to the next line unless CR and LF codes are issued) and with a visible cursor, although these modes can be changed.

Control Codes

The full list of control codes supported is as follows (other control codes are ignored):

7 (BEL)	Sounds a beep.
8 (BS)	Moves the cursor 1 character to the left and replaces the character under the cursor with a space. If wrapping is enabled and the cursor is already at the left, it moves to the rightmost position of the previous line (unless it is already on the top line).
10 (LF)	Moves the cursor down 1 line, scrolling the window up if necessary.
13 (CR)	Moves the cursor to the leftmost column of the current line.
27 (ESC)	Introduces an escape sequence or a literal character. The following escape sequences are interpreted. If an ESC character is followed by any character other than those in this list, then that character is output as a literal (without any further interpretation) directly to the window, and the escape sequence is terminated.

ESC A	Moves the cursor up. If the cursor is already on the top row, it has no effect.
ESC B	Moves the cursor down. If the cursor is already on the bottom row, it has no effect.
ESC C	Moves the cursor right. If the cursor is already in the rightmost column, it has no effect.

ESC D	Moves the cursor left. If the cursor is already in the leftmost column, it has no effect.
ESC E	Clears the window. The cursor position is not affected.
ESC H	Homes the cursor (ie moves the cursor to row 0, column 0).
ESC I	(Reverse index). Moves the cursor up 1 line, scrolling the window down if necessary.
ESC J	Erases all characters from the cursor (including the character under the cursor) to the end of the window. The cursor position is not affected.
ESC K	Erases all characters from the cursor (including the character under the cursor) to the end of the line. The cursor position is not affected.
ESC L	Inserts a line. The row with the cursor on it and all rows below are scrolled down 1 line. The cursor row is cleared. The cursor position is not affected.
ESC M	Deletes the row with the cursor on it. All rows below are scrolled up 1 line. The bottom row is cleared. The cursor position is not affected.
ESC N	Deletes the character under the cursor. All characters to the right of the cursor are moved 1 column to the left. The character at the end of the row is cleared. The cursor position is not affected.
ESC Y r c	Moves the cursor to the given position. r is the row number plus 32, and c is the column number plus 32. If the position is beyond the edge of the window, the cursor is moved to the edge of the window.
ESC b c	Sets the foreground (ink) colour. c can be either a Spectrum colour (0 to 7) or a colour value compatible with Locomotive's +3 CP/M implementation, as follows: 32, black 34 or 35, blue 40 or 44, red 42 or 47, magenta 64 or 80, green 66 or 83, cyan 72 or 92, yellow 74 or 95, white
ESC c c	Sets the background (paper) colour. c can be either a Spectrum colour (0 to 7) or a colour value compatible with Locomotive's +3 CP/M implementation, as follows: 32, black 34 or 35, blue 40 or 44, red 42 or 47, magenta 64 or 80, green 66 or 83, cyan 72 or 92, yellow 74 or 95, white

ESC d	Erases all characters from the start of the window up to the cursor (including the character under the cursor). The cursor position is not affected.
ESC e	Enables the cursor blob.
ESC f	Disables the cursor blob.
ESC j	Saves the cursor position.
ESC k	Restores the cursor position (as saved by ESC j).
ESC l	Erases all characters on the current line. The cursor position is not affected.
ESC o	Erases all characters from the start of the line up to the cursor (including the character under the cursor). The cursor position is not affected.
ESC p	Enables reverse video mode.
ESC q	Disables reverse video mode.
ESC r	Enables underline mode.
ESC u	Disables underline mode.
ESC v	Enables wrapping at the end of a line.
ESC w	Disables wrapping at the end of a line.

- Null channels

  The "Z>" channel simply discards all output that is sent to it. It always returns a CR character (ASCII 13) when it is read from (so the `INPUT #` command always returns an empty string). This can be handy if you wish to execute a command which produces output that you do not want.

  The channel specifier for the null channel type is simply "Z>".

  Null channels support all the pointer operations (`FN ptr#()`, `FN ext#()` and `POINT #`).

  Example

  Here is an example program that does nothing much at all!

```
    10 OPEN #4, "Z>"
    30 CAT #4
    40 INPUT #4;x$
```

ResiDOS Command Reference

The following is a complete list of commands currently provided by *ResiDOS*. "Hard disk" refers to hard disks, CompactFlash cards or SD/MMC cards (depending upon your interface).

Note that this reference *does not* include commands provided by optional packages, such as TapeIO and the ZX80/ZX81 emulators. Please see the package instructions for information on the commands that they provide.

For *all* commands, the `%` characters are optional. They're only required if the command would otherwise be accepted by BASIC (eg, in most BASICs, `LOAD "name"` would load a program from tape, so you need to use `LOAD %"name"` to load a program from disk or compactflash). For BASICs which use keyword entry (eg standard Spectrum BASIC), it's usually convenient to start commands which don't have a keyword with `%`, simply to take the Spectrum out of `K` mode and into `L` mode. On single-character-entry BASICs like *SE BASIC* or *Gosh Wonderful* there's no need to bother with the leading `%`.

For commands which simply display some information (ie `CAT`, `DIR`, `ROMS`, `PARTITIONS`, `MAP`) you can redirect the display to any stream by specifying a stream number after the command name (but before any parameters). For example:

```
%ROMS+ #3

%DIR #4;"*.bas"
```

Disk management commands

`%DRIVE unit, cyl, head, sec`

Specifies the parameters (number of cylinders, heads and sectors) of your hard disk. "Unit" is 0 for the master drive, or 1 for the slave drive. Only needs to be done if the auto-detection fails to work.

`%DRIVE unit, 0, 0, 0`

Disables any detection of "unit" altogether. This might help improve boot speed slightly if, for example, you never have a slave drive attached.

`%DRIVE unit`

Changes "unit" to use auto-detection again, rather than manually set parameters.

`%DRIVES`

Unmaps all current drives and re-initialises IDEDOS (causing disks to be re-detected, and drives mapped back again), without losing any of the program or data currently in memory. This is useful to allow swapping of compactflash cards while the Spectrum is still on.

`%REFORMAT unit, maxpartition`

Completely erases the hard disk (unit=0 for master, 1 for slave) and initialises it with the IDEDOS structure. Following this, you will be able to use the partition management commands to add partitions for storing data on the disk. The maxpartition number is approximately the total number of partitions you will be able to create; don't make this too large (hundreds of partitions), or other disk management routines will be too slow. Don't make it too small, or you may not be able to create all the partitions you want (some partitions are used to keep track of free space, etc). Suggested values are between 15-63.

`%REFORMAT unit,maxpartition,cyls`

Same as the previous command, but limits IDEDOS to using the first cyls cylinders on the disk, allowing it to be shared with other filesystems (such as FAT, used by PCs and the optional FATfs package).

Partition management commands

Note that some commands in this section only work with an IDEDOS drive (formatted with the `%REFORMAT` command. For other filesystems (eg with the optional FATfs package) it may not be possible to create, delete or rename partitions; see the documentation for the filesystem package for details of what it supports.

`%PARTITIONS`

Displays a list of all the partitions on your disk.

`%PARTITIONS+`

Displays an expanded listing of all the partitions on your disk, including their physical positions on the disk.

`%PARTITION "name",size`

Creates a new data partition on your disk, for storing files. Partition names are not case-sensitive, and can be up to 16 characters long, and include a unit identifier ("0>" or "1>"). The maximum partition size is 16Mb. Sizes can be specified in kilobytes (numbers larger or equal to 256), or megabytes (numbers up to 255).

`%PARTITION "name",size,2`

Creates a swap partition on your disk for temporary use by the Spectrum. These are not currently used by ResiDOS itself, but may be used by applications in the future. The maximum partition size is 16Mb. Sizes can be specified in kilobytes (numbers larger or equal to 256), or megabytes (numbers up to 255). (NB: In fact, you can specify any partition type in this command, but only 2 (swap) and 3 (data) are useful types).

`ERASE % "unit>name"`

`%ERASE "unit>name"`

`%ERA "unit>name"`

`%DEL "unit>name"`

Deletes a partition, together with all the files it contains. This can only be done if no drive is currently "mapped" to it. The unit identifier must be present in this command.

`%REN "unit>name","newname″`

Renames a partition. The unit identifier must be present in the original name in this command.

Drive mapping commands

`%AUTOMAP+`

Turns automatic drive mappings on.

`%AUTOMAP-`

Turns automatic drive mappings off.

`%MAP`

Displays a list of the current drive mappings.

`%MAP "drive","name"`

Maps a drive ("A:" to "P:") to the partition specified. A maximum of 6 drives can be mapped at any time.

`%MAP+ "drive","name"`

Maps a drive, and makes the assignment permanent, so it will automatically be restored each time you switch on.

`%UNMAP "drive"`

Unmaps the drive ("A:" to "P:") specified.

`%UNMAP+ "drive"`

Unmaps a drive. Additionally, removes any permanent assignment for the drive.

Drive, directory and path commands

`%CD "location:path"`

Makes the specified location and path the default. The location can be a user area (0-15) or a drive (A-P), or both. The path can be any list of directories (seperated with / or \ characters) from the current directory. If you don't specify all of these, the current default for the others will be retained. Any filenames that don't include a drive letter, user area or path will now refer to these defaults.

`%CD+ "location:path"`

Makes the specified location and path the default, and makes this permanent so that it will also be the default on startup. Each drive can have its own default path. If you don't set any defaults, then the root directory and user area 0 on drive A: will be used when you switch on.

`%MD "directory"`

`%MKDIR "directory"`

Creates a directory.

`%RD "directory"`

`%RMDIR "directory"`

Removes a directory (it must be empty).

`%PWD`

Prints the current working drive, user area and directory.

File management commands

`CAT %`

`%CAT`

`%DIR`

Shows a list of all files on the default drive in the default user area.

`CAT+ %`

`%CAT+`

`%DIR+`

Shows an expanded listing of all files on the default drive in the default user area.

`CAT % "filespec"`

`%CAT "filespec"`

`%DIR "filespec"`

Shows a list of files that match the file specification provided. This can include a user area, drive, filename, extension and wildcard characters (* and ?).

`CAT+ % "filespec"`

`%CAT+ "filespec"`

`%DIR+ "filespec"`

Shows an expanded listing of files that match the file specification provided.

`LOAD %"filename"`

Loads a BASIC program from hard disk.

`LOAD %"filename" CODE`

Loads a CODE file from hard disk. If the file came from another system (ie was not SAVEd by ResiDOS or a +3/+3e), then the default load address will be 32768.

`LOAD %"filename" CODE a`

Loads a CODE file from hard disk at a specific address.

`LOAD %"filename" CODE a, l`

Loads a CODE file from hard disk at a specific address, and with a specific length.

`LOAD %"filename" SCREEN$`

Loads a screen from hard disk.

`SAVE %"filename"`

Saves a BASIC program to hard disk.

`SAVE %"filename" LINE n`

Saves an auto-starting BASIC program to hard disk.

`SAVE %"filename" CODE a, l`

Saves a CODE file to hard disk.

`SAVE %"filename" SCREEN$`

Saves a screen to hard disk.

`%SNAPLOAD "filename"`

Loads a standard .SNA or .Z80 snapshot from hard disk.

`%SNAPLOAD+ "filename"`

Loads a standard .SNA or .Z80 snapshot from hard disk. Overwrites a small amount of screen (instead of stack memory), which aids compatibility with some snapshots.

`%SNAPLOAD- "filename"`

Loads a standard .SNA or .Z80 snapshot from hard disk. Starts the snapshot with the built-in ROM, which aids compatibility with some snapshots.

`%SNAPLOAD+- "filename"`

Loads a standard .SNA or .Z80 snapshot from hard disk. Uses the screen and the built-in ROM, which aids compatibility with some snapshots.

`LINE %"filename", line`

Changes the auto-run line number for a BASIC program. If the line provided is 0, auto-running is disabled for the program.

`ERASE % "filespec"`

`%ERASE "filespec"`

`%ERA "filespec"`

`%DEL "filespec"`

Erases a file from the hard disk. The filespec can include wildcards to erase multiple files at once.

`%REN "filename","newname"`

Renames a file (wildcards are not allowed). This command can be used to change the user area a file is located in (but not the drive).

`%ATTRIB "filespec","+attrib"`

Adds the attribute (P=protected, S=system or A=archive) to the file (or group of files) specified.

`%ATTRIB "filespec","-attrib"`

Removes the attribute (P=protected, S=system or A=archive) from the file (or group of files) specified.

`%CP "filename","filename2"`

Copies a single file to a new file, which may be in the same location, or a different one.

`%CP "filespec","location:"`

Copies one or more files (if using wildcards) to another location.

Customisation commands

`INK % n`

Sets ink colour to n and saves the current colour scheme as the default.

`PAPER % n`

Sets paper colour to n and saves the current colour scheme as the default.

`BRIGHT % n`

Sets bright to n and saves the current colour scheme as the default.

`FLASH % n`

Sets flash to n and saves the current colour scheme as the default.

`ATTR % n`

Sets attributes to n and saves them as the default colour scheme.

`%AUTORUN "filename"`

Sets the autorun file, which will only be executed with `%AUTORUN`.

`%AUTORUN+ "filename"`

Sets the autorun file, which will be executed automatically on boot, or with `%AUTORUN`.

`%AUTORUN-`

Removes any autorun file setting.

`%AUTORUN`

Loads and runs the autorun file.

`%SPEED n`

Sets the processor speed to n. Supported only on ZX-Badaloc, where n is 0 (3.54MHz), 1 (7MHz), 2 (14MHz) or 3 (21MHz). Add 4 to enable the clock/screen/interrupt doubler (gives speed up to 42MHz).

`%SPEED+ n`

Sets the processor speed to n and makes this the default speed when starting up.

`%SPEED- n`

Sets the processor speed to n and removes any default speed, so the standard system speed is used when starting up.

ROM management commands

`%INSTALL "filename"`

Installs a file from the hard disk to the interface memory as a package or an Interface II-type ROM. NB: you must first `CLEAR 32767` or lower.

`%INSTALL "filename",n`

Installs a file image from the hard disk to the interface as a type "n" ROM. NB: you must first `CLEAR 32767` or lower. Values for "n" are made as the sum of the following, and are ignored for packages:
0: for an Interface II ROM (or any other ROM that does not require ResiDOS support)
1: for a BASIC ROM
16: to install the ROM into the writeable "RAM" part of the interface memory
32: to disable the NMI button for this ROM (only affects BASIC ROMs)

`%INSTALL "name"@addr`

`%INSTALL "name"@addr,n`

As the previous commands, but indicates that the ROM image is already loaded into memory at the specified address and does not need to be loaded from a file. The name refers only to the name that will be given to the ROM when it is installed. NB: you should `CLEAR` to an address lower than addr before loading the image to memory and then executing the `%INSTALL` command.

`%ROMS`

Shows a list of all installed ROMs.

`%ROMS+`

Shows a list of all installed ROMs, including hidden system ones.

`%UNINSTALL"name"`

Uninstalls a ROM.

`%ROM "name"`

Restarts computer using selected BASIC or Interface 2 ROM.

`%ZX`

Restarts computer using internal ROM, effectively disabling the interface.

Spectrum mode commands

`%SPECTRUM`

Switch 128K Spectrum into 48K mode until the next time the computer is reset.

`%SPECTRUM+`

Always run in 48K mode, even after a reset.

`%SPECTRUM-`

Turns off permanent 48K mode (you'll need to reset to return to 128K mode).

ZXATASP: `OUT 671,192:OUT 671,128`
ZXCF: `OUT 4287,128`
ZXMMC+: `OUT 127,0`
ZX-Badaloc: `OUT 21727,0`
divIDE Plus: `OUT 23,0`

Disables ResiDOS, whilst keeping the current program in memory.

ZXATASP: `OUT 671,64:OUT 671,0`
ZXCF: `OUT 4287,64`
ZXMMC+: `OUT 127,192`
ZX-Badaloc: `OUT 21727,200`
divIDE Plus: `OUT 23,104`

Re-enables ResiDOS, whilst keeping the current program in memory.

Error-trapping commands

`%ONBREAK THEN CONTINUE`

Turns on BREAK-trapping. The `THEN` is optional.

`%ONBREAK OFF`

Turns off BREAK-trapping.

`%ONERR THEN GOTO line`

Turns on error-trapping. If an error occurs, control is transferred to the line indicated. The `THEN` is optional.

`%ONERR THEN GOSUB line`

Turns on error-trapping. If an error occurs, control is transferred to the line indicated. When a `RETURN` is encountered, execution continues at the statement after the error occurred. The `THEN` is optional.

`%ONERR OFF`

Turns off error-trapping.

`%REPORT`

Reports the last error that was trapped.

Error-trapping functions

`FN %ERR()`

Gives the error number of the last error that was trapped.

`FN %ERL()`

Gives the line number of the last error that was trapped.

`FN %ERS()`

Gives the statement number of the last error that was trapped.

Channel commands

`OPEN #` *`stream, c$`*

Opens a stream (0..15) to the channel specified by c$. This may be a standard channel ("K", "S" or "P") or a new channel provided by a package.

`CLOSE #` *`stream`*

Closes a stream (0..15). If the stream is in the range 0..3, the original default channel assignment is restored.

`POINT #` *`stream, pos`*

Sets the current pointer position of the stream to *pos*.

Channel functions

`FN %PTR#(`*`stream`*`)`

Returns the current pointer position of the stream specified.

`FN %EXT#(`*`stream`*`)`

Returns the extent (size) of the stream specified.

ResiDOS Programming

From v1.40, it is possible to access some of the features of the operating system from machine-code programs, so that utility programs which access files can be written.

Calls are available in *ResiDOS* using *hook codes*, which are invoked using the RST 8 instruction followed by a hook code. There are two main hook codes available to programmers: the version information hook code, which detects which version of *ResiDOS* is available (if any), and the package call hook code, which is used to access all other facilities.

All the following pieces of code are provided in the format used by Interlogic's free *z80asm* assembler which is available for many platforms at Z88 Forever! (www.worldofspectrum.org/z88forever). Binaries are provided for several platforms, and most others (including *MacOS X*) can compile the source package. Additionally, instead of using "magic numbers" for the various call IDs etc, these are defined as meaningful words. You should download the resitools file from the ResiDOS Home page (www.worldofspectrum.org/residos), which contains the necessary include files, together with an example utility program (assembly file and pre-built TAP/TZX).

Detecting ResiDOS

Any program which is going to use *ResiDOS* facilities must first check that it is installed, and that a suitable version number is available. You should check for a version which is high enough to support all the calls you are going to use. *ResiDOS* version 1.40 was the first to support package calls; each call introduced subsequently has the first version for which it was available listed.

Some suitable code for testing the version follows. It can be called from BASIC, and returns the version number in BC (or 0 if not installed).

```
.residetect
         ld       hl,(ERR_SP)
         push     hl                          ; save the existing ERR_SP
         ld       hl,detect_error
         push     hl                          ; stack error-handler return address
         ld       hl,0
         add      hl,sp
         ld       (ERR_SP),hl                 ; set the error-handler SP
         rst      RST_HOOK           ; invoke the version info hook code
         defb     HOOK_VERSION
         pop      hl                          ; ResiDOS doesn't return, so if we get
         jr       noresidos          ; here, some other hardware is present
.detect_error
         pop      hl
         ld       (ERR_SP),hl                 ; restore the old ERR_SP
         ld       a,(ERR_NR)
         inc      a                           ; is the error code now "OK"?
         jr       nz,noresidos                ; if not, ResiDOS was not detected
         ex       de,hl                       ; get HL=ResiDOS version
         push     hl                          ; save the version
         ld       de,$0140           ; DE=minimum version to run with
         and      a
         sbc      hl,de
         pop      bc                          ; restore the version to BC
         ret      nc                          ; and return with it if at least v1.40
.noresidos
         ld       bc,0                        ; no ResiDOS
         ld       a,$ff
         ld       (ERR_NR),a                  ; clear error
         ret
```

Detecting ResiDOS from BASIC

You may wish to detect whether ResiDOS is installed from BASIC (this may be done before attempting to use a ResiDOS BASIC command to load a machine code file from disk, for example).

The suggested way to do this (as used by the ResiDOS installer program) is to check whether the ResiDOS initialisation hook is present in the BASIC ROM. This is done as follows:

```
    IF PEEK 4768=207 AND PEEK 4769=254 THEN PRINT "ResiDOS is installed"
```

Note that there is no way to determine the version of ResiDOS which is installed without using machine-code.

Package calls

The facilities of *ResiDOS* are divided up into *packages*, each of which provides its own set of calls. Two packages are currently provided: the *ResiDOS* package itself, and the *IDEDOS* package. The first of these provides system functions not related to disk access; the other provides all the useful calls for accessing partitions and files.

Making a package call involves the following steps:

- setting up registers required by the call
- switching to the alternate set, and loading B with the package ID and HL with the call ID
- invoking the package call hook code
- checking for errors if the carry flag is reset (Fc=0)
- examining the registers returned by the call

Non-standard Packages

From ResiDOS v1.40, two packages have always been available: ResiDOS and IDEDOS. These packages work in a slightly different way to all subsequently-released packages.
For these packages, the alternate registers are set up with B=package ID (PKG_RESIDOS or PKG_IDEDOS) and HL=call ID (as listed in the call references).
Here is some generic code to make a ResiDOS or IDEDOS package call. Note that if you are intending to return to BASIC at any point, you should always preserve the alternate HL register pair (H'L') as this is required by BASIC:

```
        ; set up any entry parameters here
        exx
        ld      b,package_id
        ld      hl,call_id
        rst     RST_HOOK           ; invoke the package call hook code
        defb    HOOK_PACKAGE
        jr      nc,process_error   ; handle error if Fc=0
        ; handle any return values here
```

Standard Packages

From ResiDOS v2.00, a new standardised system was introduced to make calls to all subsequently-released packages.

For these packages, the alternate registers are set up with BC=call ID (as listed in the call references). In fact, this call ID is made up of two 8-bit values: B=package ID and C=call number, but these are usually referred to as a single 16-bit call ID value.

Note that any call to a standard package may fail with Fc=0 and A=rc_resi_package_not_found if the package is not installed, or if the version of the package installed is not high enough to support the call.

Code to invoke a call provided by a standard package would be:

```
; set up any entry parameters here
exx
ld       bc,call_id
rst      RST_HOOK           ; invoke the package call hook code
defb     HOOK_PACKAGE
jr       nc,process_error   ; handle error if Fc=0
; handle any return values here
```

Standard Calls

All standard packages provide a set of 8 standard calls (where B=package ID and C=$00..$07). These are mostly of use only to ResiDOS itself, to manage the packages. However, one of these standard calls can be used to determine the version of a package that is installed, and this can be used by application programs to ensure that a recent-enough version of the package is available to them.

This is the PKG_STDCALL_INFO ($02) call, and can be used by setting A=info_version (0) and making the call. If the package is present, the call will be successful (Fc=1) and BC contains the binary-coded-decimal version number of the installed package. If the call fails (Fc=0), an error of A=rc_resi_package_not_found will be returned.

For example, to check what version (if any) of the TapeIO package is installed, use the following code:

```
ld      a,info_version
exx
ld       bc,TAPEIOPKG_ID*$100+PKG_STDCALL_INFO
rst      RST_HOOK           ; invoke the package call hook code
defb     HOOK_PACKAGE
jr       nc,process_error   ; handle error if Fc=0
; BC holds version number (in BCD form)
```

The ResiDOS Package

This package provides calls to access *ResiDOS* facilities and hardware features (such as extra built-in memory). The calls provided can be used without knowing whether your application is running on a DivIDE Plus, ZXCF, ZXATASP or any future interface which may be supported by *ResiDOS*.

The ResiDOS Package provides the following calls:

- RESI_REPORT
- RESI_BASIC
- RESI_GETPAGER
- RESI_FINDPKG
- RESI_FINDBASIC
- RESI_ALLOC
- RESI_DEALLOC
- RESI_LOGO
- RESI_MESSAGE
- RESI_NMISVC
- RESI_BROADCASTEXP
- RESI_CONFIG
- RESI_CHANNELSVC
- RESI_STREAM_OPEN
- RESI_STREAM_CLOSE
- RESI_STREAM_IN
- RESI_STREAM_OUT
- RESI_STREAM_PTR
- RESI_STREAM_SELECT
- RESI_COMMAND
- RESI_IM1SVC
- RESI_TEMPSVC
- RESI_STREAM_LINEIN

RESI_REPORT ($0310)

Generates report message for specified ResiDOS/IDEDOS error code.

Available from: v1.40

IN: A=error code returned by ResiDOS/IDEDOS
B=package ID (only needed if A=rc_pkg_*)

OUT: -

Register status on return:
(call does not return)

Example:

```
ld      a,rc_noswap                  ; "no swap partition"
exx                                  ; switch to alternate set
ld      b,PKG_RESIDOS                ; ResiDOS package
ld      hl,RESI_REPORT               ; ResiDOS call ID
rst     RST_HOOK          ; invoke the package call hook code
defb    HOOK_PACKAGE
```

RESI_BASIC ($0313)

Calls a routine in the current BASIC ROM. This isn't terribly useful for normal programs, but is provided to allow packages to call standard routines in the BASIC ROM. Note that the "current" BASIC ROM may be an alternate such as Gosh Wonderful or SE BASIC and so only well-known entry points should be used. The IY register pair is set to ERRNR ($5c3a) by ResiDOS before the BASIC ROM routine is called.

Available from: v1.84

IN:	IY=address of routine in BASIC ROM
	Other parameters as required by routine
OUT:	Outputs from routine in BASIC ROM

Register status on return:
As defined by BASIC ROM routine

Example:

```
        ld       ix,$4000          ; parameters needed by
        ld       de,$1b00          ; BASIC ROM's "SA-BYTES"
        ld       a,$ff                     ; routine to save a screen to tape
        ld       iy,$04c2          ; SA-BYTES routine address
        exx                                ; switch to alternate set
        ld       b,PKG_RESIDOS             ; ResiDOS package
        ld       hl,RESI_BASIC             ; ResiDOS call ID
       call    PACKAGE_CALL_PKG        ; make the call from this package
```

RESI_GETPAGER ($031c)

Installs a routine to the specified location which can be used to switch built-in interface memory to the bottom 16K ($0000-$3fff), where the BASIC ROM is normally located. The routine can be located anywhere you like within the Spectrum's standard memory map (you should allow a space of 32 bytes to hold the routine). If running on a 128K Spectrum, you can place it within any of the additional pages (although it is your responsibility to ensure the correct page is switched in).

Once you have obtained this routine, you can call it at any time with a "bank ID" in A which has previously been provided to you by the calls RESI_FINDPKG, RESI_FINDBASIC or RESI_ALLOC. Do not use any other values! After calling the routine, the 16K memory bank indicated by the "bank ID" will be located at $0000-$3fff. If you are switching in a bank obtained by RESI_ALLOC, don't forget to disable interrupts first, otherwise you may crash. It is safe to leave interrupts enabled when paging in a package or BASIC, provided the interrupt mode is 1.

Available from: v1.84
IN: DE=address of 32-byte area to hold paging routine
OUT: Fc=1, success

Register status on return:
......../IXIY same
AFBCDEHL/.... different

The paging routine is called as follows:
IN: A=bank ID
OUT: -

Register status on return:
AFBCDEHL/IXIY same
......../.... different

An example of using the call, and the paging routine is as follows:
```
          ld        de,mypager                    ; location for paging routine
          exx
          ld        b,PKG_RESIDOS
          ld        hl,RESI_GETPAGER    ; get paging routine
          rst       RST_HOOK
          defb      HOOK_PACKAGE
          jr        nc,failed           ; call failed if Fc=0
          ....                                    ; more program
          di                                      ; disable interrupts
          ld        a,(mybank)                    ; bank obtained by RESI_ALLOC
          call      mypager                       ; switch it in to $0000-$3fff
          ....                                    ; more program
          ld        a,(basicbank)                 ; bank obtained by RESI_FINDBASIC
          call      mypager                       ; switch BASIC ROM back in
          ei                                      ; re-enable interrupts
          ....                                    ; more program

.mypager
          defs      32                            ; 32 bytes to hold paging routine
```

RESI_FINDPKG ($031f)

Obtains the bank ID for a package, so that the programmer can page in the package directly using a pager routine obtained with RESI_GETPAGER.

Available from: v1.84
IN: A=package ID
OUT(s): Fc=1, A=bank ID
OUT(f): Fc=0, A=error code

Register status on return:
....DEHL/IXIY same
AFBC..../.... different

Example:
```
          ld        a,PKG_IDEDOS                  ; request bank for IDEDOS package
          exx
          ld        b,PKG_RESIDOS
          ld        hl,RESI_FINDPKG               ; get bank
          rst       RST_HOOK
          defb      HOOK_PACKAGE
          jr        nc,failed           ; call failed if Fc=0
          ld        (idedos_bank),a               ; save for later
```

RESI_FINDBASIC ($0322)

Obtains the bank ID for the version of BASIC currently being used, so that the programmer can page it back in directly using a pager routine obtained with RESI_GETPAGER.

Available from: v1.84
IN: -
OUT(s): Fc=1, A=bank ID
OUT(f): Fc=0, A=error code

Register status on return:
..BCDEHL/IXIY same
AF....../.... different

Example:
```
        exx
        ld      b,PKG_RESIDOS
        ld      hl,RESI_FINDBASIC  ; get bank
        rst     RST_HOOK
        defb    HOOK_PACKAGE
        jr      nc,failed          ; call failed if Fc=0
        ld      (basic_bank),a              ; save for later
```

RESI_ALLOC ($0325)

Requests a 16K bank of RAM for the program's use. If successful, the bank can be switched in to $0000-$3fff by disabling interrupts and using a pager routine obtained with RESI_GETPAGER. A program or package can request as many banks as it likes, and switch between them using the pager routine.

Available from: v1.84
IN: -
OUT(s): Fc=1, A=bank ID
OUT(f): Fc=0, A=error code

Register status on return:
......../IXIY same
AFBCDEHL/.... different

Example:
```
        exx
        ld      b,PKG_RESIDOS
        ld      hl,RESI_ALLOC               ; get free bank
        rst     RST_HOOK
        defb    HOOK_PACKAGE
        jr      nc,failed          ; call failed if Fc=0
        ld      (mybank),a                  ; save for later
```

RESI_DEALLOC ($0328)

Frees a 16K bank of RAM previously obtained with RESI_ALLOC. After using this call, the program or package may not access the bank again. If a program fails to de-allocate its banks, they will be reclaimed by the system the next time that ResiDOS starts up. If a package fails to de-allocate its banks when it is uninstalled, they will not be freed (this would be considered to be a bug in the package).

Available from: v1.84
    IN: A=bank ID
OUT(s): Fc=1
OUT(f): Fc=0, A=error code

Register status on return:
......../IXIY same
AFBCDEHL/.... different

Example:
```
        ld      a,(mybank)
        exx
        ld      b,PKG_RESIDOS
        ld      hl,RESI_DEALLOC            ; release bank
        rst     RST_HOOK
        defb    HOOK_PACKAGE
```

RESI_LOGO ($032b)

Displays any interface-specific graphical logo at the requested position on screen. If the interface used by this version of ResiDOS does not have a logo (currently only ZXCF interfaces use a logo), then this call returns an error of rc_notimp.

Available from: v2.00
    IN: C=screen line
         B=screen column
OUT(s): Fc=1
OUT(f): Fc=0, A=error code

Register status on return:
......../IXIY same
AFBCDEHL/.... different

Example:
```
        ld      bc,$0800               ; line 0, column 8
        exx
        ld      b,PKG_RESIDOS
        ld      hl,RESI_LOGO           ; display logo
        rst     RST_HOOK
        defb    HOOK_PACKAGE
```

RESI_MESSAGE ($032e)

This call allows you to obtain the text description of any ResiDOS/IDEDOS/ package error obtained from a previous call. The text returned is the same as that which would be generated by the RESI_REPORT call. The buffer you provide at DE must be at least 23 bytes long. If the error code in A is a package-specific error (rc_pkg_error_0...rc_pkg_error_31) then you should also supply the package ID for the call that produced the error in B. Messages obtained are terminated with bit 7 set in the last character.

Available from: v2.00
    IN: A=error code
        B=package ID (only needed if A=rc_pkg_*)
        DE=address of 23-byte buffer
  OUT(s): Fc=1
  OUT(f): Fc=0, A=error code

Register status on return:
........./IXIY same
AFBCDEHL/.... different

Example:

```
        ld      a,rc_notimp             ; get text for "rc_notimp" error
        ld      b,PKG_RESIDOS           ; generated by ResiDOS package
                                       ; (not actually needed in this case)
        ld      de,mybuffer
        exx
        ld        b,PKG_RESIDOS
        ld        hl,RESI_MESSAGE         ; get message to DE
        rst       RST_HOOK
        defb      HOOK_PACKAGE
```

RESI_NMISVC ($0331)

This call is only for the use of packages which implement an NMI handler (ie have PKGCAPS_NMI set in their package capabilities) and should only be called during execution of their package_nmi call.

It allows any NMI handler to obtain the set of register values that were in force at the time that the NMI button was pressed, and to change these before returning from the package_nmi call. This allows the state of the Spectrum to be changed by the NMI handler (Task Manager does this when switching tasks, for example).

The NMI register record obtained is held in the same format as the 30-byte header of v1.45 Z80 snapshot files (described here).

Available from: v2.00
    IN: B=reason code:
            nmisvc_getregs (0) to obtain NMI register record
            nmisvc_putregs (1) to update NMI register record
        HL=address of NMI register record
  OUT(s): Fc=1
  OUT(f): Fc=0, A=error code

Register status on return:
......../.... same
AFBCDEHL/IXIY different

Example:
```
        ld      b,nmisvc_getregs        ; obtain NMI register record
        ld      hl,z80hdrbuffer         ; address of 30-byte buffer
        exx
        ld       b,PKG_RESIDOS
        ld       hl,RESI_NMISVC          ; get record to HL
        call    PACKAGE_CALL_PKG        ; make the call from this package
```

RESI_BROADCASTEXP ($0334)

This call broadcasts an expansion call to all currently-installed packages (using their package_exp call). Any parameters required can be passed in BC/DE/HL/IX and will be preserved; return results from each package called are simply ignored.

Any reason code may be provided to this function; only packages which support the reason code will be affected. Reason codes that are currently supported by one or more packages are:

exp_boot	Initialise package after power-cycle (no parameters).
exp_fs_setdrive	Filesystem packages: set the current drive letter. B holds the upper-case drive letter.
exp_fs_setuser	Filesystem packages: set the current user area. B holds the user area number (0-15).
exp_fs_snapdata	Filesystem packages: get snapshot data. (Details to be documented later.)

Note that none of the above reason codes should be broadcast by applications or other packages - they are for internal ResiDOS use only. If you wish to add a new expansion call reason code, please discuss it with me first.

Available from: v2.00
IN: A=reason code
BC/DE/HL/IX=parameters, dependent on reason code
OUT(s): Fc=1

Register status on return:
..BCDEHL/IXIY same
AF....../.... different

Example:
```
        ld      a,exp_boot              ; initialise all packages
                                      ; DO NOT USE THIS REASON!
        exx
        ld       b,PKG_RESIDOS
        ld       hl,RESI_BROADCASTEXP    ; broadcast to all packages
        rst      RST_HOOK
        defb     HOOK_PACKAGE
```

RESI_CONFIG ($0337)

This call accesses the ResiDOS configuration file, residos.cfg on the boot drive. It can be used to read, write or delete any named entry.

NOTE: It is important that the buffers pointed to be DE and HL do not lie in the top 2K of RAM. They must lie between $4000 and $f800.

Available from: v2.00

```
      IN:  A=reason code:
             rc_config_getvalue (0), to read a numeric value
             rc_config_gettext (1), to read a text value
             rc_config_setvalue (2), to write a numeric value
             rc_config_settext (3), to write a text value
             rc_config_delete (4), to delete a value
           DE=address of setting name (lower-case, $ff-terminated)
           HL=numeric value, or address of text buffer
    OUT(s):  Fc=1
           HL=numeric value (for rc_config_getvalue only)
    OUT(f):  Fc=0, A=error code

    Register status on return:
    ......../IXIY same
    AFBCDEHL/.... different
```

Example:

```
        ld      b,rc_config_setvalue    ; set a numeric value
        ld      hl,3                    ; number to set
        ld    de,configname       ; address of setting name (lower-case, $ff-terminated)
        exx
        ld       b,PKG_RESIDOS
        ld       hl,RESI_CONFIG
        rst      RST_HOOK
        defb     HOOK_PACKAGE
```

RESI_CHANNELSVC ($033a)

This call provides services for packages that implement extended channels. Only packages can provide channels (since the input/output handlers are always package calls) so application programs should generally not use this call, although they may wish to use the channelsvc_parseint* reasons for non-channel-related purposes.

The channelsvc_parseint* reasons are intended to allow a package to easily parse parameters present in a channel specifier string. A default of zero is used if no integer is found. Therefore, a string "A>,76" would provide values of 0 and 76 when parsed with a channelsvc_parseint followed by a channelsvc_parseintcomma.

The channelsvc_create reason creates a channel in the CHANS area, updating all system variables as appropriate. The layout of all ResiDOS-compatible channels is as follows (as defined in packages.def):

```
defw $0008      ; "output" routine (switch to ResiDOS)
defw $0008      ; "input" routine (switch to ResiDOS)
defb 'A'        ; channel name (single uppercase letter)
defw pkgout     ; package call for output routine
defw pkgin      ; package call for input routine
defw length     ; total length of channel (11+size of user data)
...optional user data follows...
```

The channelsvc_delete reason deletes a channel, updating all system variables as appropriate.

The channelsvc_grow and channelsvc_shrink reasons increase or decrease the size of a channel, by adding or removing data from the end of the user data area. Channels may not be shrunk to less than 11 bytes. Again, all system variables are kept up-to-date.

Available from: v2.12

IN: A=reason code:

channelsvc_parseint (0), to parse an integer from a string

channelsvc_parseintmax (1), to parse an integer up to a maximum value from a string

channelsvc_parseintcomma (2), to parse an integer preceded by a comma from a string

channelsvc_parseintcommamax (3), to parse an integer up to a maximum value, preceded by a comma, from a string

channelsvc_create (4), to create a channel

channelsvc_delete (5), to delete a channel

channelsvc_grow (6), to increase the size of a channel

channelsvc_shrink (7), to decrease the size of a channel

For the channelsvc_parseint* calls:

HL=address of string to parse

BC=length of string to parse

DE=maximum value allowable (*max versions only)

For channelsvc_create:

D=channel name (uppercase letter)

Bit 7 may be used as a flag, and is ignored by ResiDOS

E=package ID which handles this channel

H=package call number for output to this channel

L=package call number for input from this channel

BC=total length of channel data (min 11 bytes if no user data)

For channelsvc_delete:

HL=address of channel

For channelsvc_grow, channelsvc_shrink:
HL=address of channel
BC=number of bytes to grow or shrink channel by

OUT(s): Fc=1

For the channelsvc_parseint* calls:
HL=address of string after integer
BC=length of string after integer
DE=value (defaults to zero if no integer found)

For channelsvc_create:
HL=address of channel
DE=address of user data (ie DE=HL+chandata_userdata)

OUT(f): Fc=0, A=error code

Register status on return:
......../IXIY same
AFBCDEHL/.... different

Example:
```
        ld      a,channelsvc_create     ; create a channel
        ld      d,'X'                   ; channel name
        ld      e,MYPKG_ID
        ld      h,MYOUTCALL
        ld      l,MYINCALL
        ld      bc,11+4                 ; "X" channel has 4 bytes of user data
        exx
        ld      b,PKG_RESIDOS
        ld      hl,RESI_CHANNELSVC
        call    PACKAGE_CALL_PKG        ; make the call from this package
```

RESI_STREAM_OPEN ($033d)

This call allows you to open a stream to any channel descriptor. This can be one of the standard (S, P, or K) channels, or a new channel provided by ResiDOS or a package. This call is directly equivalent to using the BASIC command `OPEN #`.

Note that, in common with the other RESI_STREAM_* calls, errors generated can be ResiDOS/IDEDOS/package errors (as normally returned by other calls) or they may be a BASIC error code (-1). You should check the zero flag to determine which type of error has been returned. Typically, the only BASIC errors generated are likely to be "O Invalid Stream" (Fz=1, A=$17) or "J Invalid I/O Device" (Fz=1, A=$12).

Available from: v2.14
IN: A=stream (0..15)
DE=address of string specifying channel to open
BC=length of string specifying channel to open
OUT(s): Fc=1
OUT(f): Fc=0,
Fz=0, A=ResiDOS/IDEDOS/package error code
B=package id (for package error codes)

Fz=1, A=BASIC error code - 1

Register status on return:
......../..IY same
AFBCDEHL/IX.. different

Example:
```
        ld      a,4                     ; stream 4
        ld      de,msg_chan_start
        ld      bc,msg_chan_end-msg_chan_start
        exx
        ld        b,PKG_RESIDOS
        ld        hl,RESI_STREAM_OPEN     ; open stream to channel
        rst       RST_HOOK
        defb      HOOK_PACKAGE

       ....

.msg_chan_start
       defm    "i>a:data.txt"
.msg_chan_end
```

RESI_STREAM_CLOSE ($0340)

This call closes a stream. If the stream is in the range 0..3, then it is re-attached to its default channel. This call is directly equivalent to using the BASIC command `CLOSE #`.

Note that, in common with the other RESI_STREAM_* calls, errors generated can be ResiDOS/IDEDOS/package errors (as normally returned by other calls) or they may be a BASIC error code (-1). You should check the zero flag to determine which type of error has been returned. Typically, the only BASIC errors generated are likely to be "O Invalid Stream" (Fz=1, A=$17) or "J Invalid I/O Device" (Fz=1, A=$12).

Available from: v2.14
IN: A=stream (0..15)
OUT(s): Fc=1
OUT(f): Fc=0,
Fz=0, A=ResiDOS/IDEDOS/package error code
B=package id (for package error codes)
Fz=1, A=BASIC error code - 1

Register status on return:
......../..IY same
AFBCDEHL/IX.. different

Example:
```
        ld      a,4                     ; stream 4
        exx
        ld        b,PKG_RESIDOS
        ld        hl,RESI_STREAM_CLOSE    ; close stream
        rst       RST_HOOK
        defb      HOOK_PACKAGE
```

RESI_STREAM_IN ($0343)

This call reads a single character from the currently active stream, if one is available. If Fc=1 and Fz=1, a character was not available, but you can loop back and wait until Fc=1 and Fz=0. If the stream is permanently unable to provide a character (an end-of-file condition, for example) then an error will be returned.

NOTE: If the current stream is open to one of the standard channels (K, S or P) then if an error occurs, a BASIC error report may be generated, and no return made to the caller.

Note that, in common with the other RESI_STREAM_* calls, errors generated can be ResiDOS/IDEDOS/package errors (as normally returned by other calls) or they may be a BASIC error code (-1). You should check the zero flag to determine which type of error has been returned. Typically, the only BASIC errors generated are likely to be "O Invalid Stream" (Fz=1, A=$17) or "J Invalid I/O Device" (Fz=1, A=$12).

Available from: v2.14

```
   IN:  -
OUT(s):  Fc=1,
         Fz=0, A=character
         Fz=1, character not currently available
             (this is not an EOF condition)
OUT(f):  Fc=0,
         Fz=0, A=ResiDOS/IDEDOS/package error code
             B=package id (for package error codes)
         Fz=1, A=BASIC error code - 1

Register status on return:
......../..IY same
AFBCDEHL/IX.. different
```

Example:

```
        exx
        ld      b,PKG_RESIDOS
        ld      hl,RESI_STREAM_IN       ; read from stream
        rst     RST_HOOK
        defb    HOOK_PACKAGE
```

RESI_STREAM_OUT ($0346)

This call writes a single character to the currently active stream.

NOTE: If the current stream is open to one of the standard channels (K, S or P) then if an error occurs, a BASIC error report may be generated, and no return made to the caller.

Note that, in common with the other RESI_STREAM_* calls, errors generated can be ResiDOS/IDEDOS/package errors (as normally returned by other calls) or they may be a BASIC error code (-1). You should check the zero flag to determine which type of error has been returned. Typically, the only BASIC errors generated are likely to be "O Invalid Stream" (Fz=1, A=$17) or "J Invalid I/O Device" (Fz=1, A=$12).

Available from: v2.14
    IN: A=character
  OUT(s): Fc=1
  OUT(f): Fc=0,
        Fz=0, A=ResiDOS/IDEDOS/package error code
          B=package id (for package error codes)
        Fz=1, A=BASIC error code - 1

Register status on return:
......../..IY same
AFBCDEHL/IX.. different

Example:
```
        ld      a,'*'
        exx
        ld      b,PKG_RESIDOS
        ld      hl,RESI_STREAM_OUT      ; write '*' to stream
        rst     RST_HOOK
        defb    HOOK_PACKAGE
```

RESI_STREAM_PTR ($0349)

This call gets or sets the read/write pointer or size of the currently active stream. If the current stream does not support the requested operation, then the BASIC error code "J Invalid I/O Device" (Fc=0, Fz=1, A=$12) will be returned.

Note that, in common with the other RESI_STREAM_* calls, errors generated can be ResiDOS/IDEDOS/package errors (as normally returned by other calls) or they may be a BASIC error code (-1). You should check the zero flag to determine which type of error has been returned. Typically, the only BASIC errors generated are likely to be "O Invalid Stream" (Fz=1, A=$17) or "J Invalid I/O Device" (Fz=1, A=$12).

Available from: v2.14
    IN: A=reason code:
        streamptr_getptr (0), to obtain the current stream pointer
        streamptr_setptr (1), to change the current stream pointer
        streamptr_extent (2), to obtain the current stream size

      For streamptr_setptr:
        DEBC=pointer

  OUT(s): Fc=1

      For streamptr_getptr:
        DEBC=pointer

      For streamptr_extent:
        DEBC=size

OUT(f): Fc=0,
Fz=0, A=ResiDOS/IDEDOS/package error code
B=package id (for package error codes)
Fz=1, A=BASIC error code - 1

Register status on return:
......../..IY same
AFBCDEHL/IX.. different

Example:
```
        ld      a,streamptr_getptr
        exx
        ld      b,PKG_RESIDOS
        ld      hl,RESI_STREAM_PTR      ; get the current stream pointer
        rst     RST_HOOK
        defb    HOOK_PACKAGE
```

RESI_STREAM_SELECT ($034c)

This call sets the current stream. This is the stream that all input and output (via BASIC commands and RST $10 as well as through the other RESI_STREAM_* calls) will be sent through.

Note that, in common with the other RESI_STREAM_* calls, errors generated can be ResiDOS/IDEDOS/package errors (as normally returned by other calls) or they may be a BASIC error code (-1). You should check the zero flag to determine which type of error has been returned. Typically, the only BASIC errors generated are likely to be "O Invalid Stream" (Fz=1, A=$17) or "J Invalid I/O Device" (Fz=1, A=$12).

Available from: v2.14
IN: A=stream (0..15)
OUT(s): Fc=1
OUT(f): Fc=0,
Fz=0, A=ResiDOS/IDEDOS/package error code
B=package id (for package error codes)
Fz=1, A=BASIC error code - 1

Register status on return:
......../IXIY same
AFBCDEHL/.... different

Example:
```
        ld      a,4
        exx
        ld      b,PKG_RESIDOS
        ld      hl,RESI_STREAM_SELECT   ; set the current stream to #4
        rst     RST_HOOK
        defb    HOOK_PACKAGE
```

RESI_COMMAND ($034f)

This powerful call allows the machine-code programmer to perform any ResiDOS BASIC command or function (including any supplied by installed packages). On entry, HL contains a command/function handler ID for the required BASIC command or function. These IDs are documented separately for ResiDOS and (optionally) for each package supplying commands and functions.

For functions, the BASIC floating-point stack should contain all the input values required by the function (with the final function parameter being topmost on the floating-point stack). When the function completes, the topmost value on the floating-point stack is the result of the function (if any); all inputs to the function will have been consumed.

For commands, IX should point to a 32-byte buffer containing input parameters for the command (as documented separately for each command). IY should point to the system variable ERR_NR ($5c3a), and this should contain an 8-bit field of command options (as documented separately for each command). Finally, some commands may also require parameters to be loaded on the BASIC floating-point stack (although this is not usual - none of the built-in ResiDOS commands do this). Again, this requirement is documented separately for each command.

If the command or function fails for any reason, a BASIC error report is not generated, but the error status is returned to the caller. If desired, the caller can then generate the BASIC error using the RESI_REPORT call.

Note that this call is not intended to replace the finer control that machine-code programs can obtain by calling lower-level +3DOS, IDEDOS, ResiDOS or package calls, but it can be useful to access functionality that isn't readily exposed except through BASIC commands.

NOTE: Packages may execute commands or functions provided by ResiDOS or other packages, but not from within a command or function handler.

```
Available from: v2.19
      IN:  HL=command/function handler ID
           floating-point stack: as required by command/function
           IX=address of 32-byte parameter buffer (commands only)
           IY=ERR_NR ($5c3a)
           (IY+0)=command option flags (commands only)
    OUT(s):  Fc=1
           floating-point stack: contains result (functions only)
    OUT(f):  Fc=0,
               Fz=0, A=ResiDOS/IDEDOS/package error code
                   B=package id (for package error codes)
               Fz=1, A=BASIC error code - 1

    Register status on return:
    ......../..IY same
    AFBCDEHL/IX.. different
```

Example:
```
         ld       ix,parambuffer
                                             ; don't need to set string address,
                                             ; as length is zero
         ld       (ix+2),0          ; string length 0 (low byte)
         ld       (ix+3),0          ; string length 0 (high byte)
         ld       (ix+4),2          ; stream 2
         ld       (iy+0),$80                 ; option 7 ("+", expanded catalog)
         ld       hl,HANDLERID_CAT  ; CAT command handler
         exx
         ld       b,PKG_RESIDOS
         ld       hl,RESI_COMMAND            ; execute command: CAT+ #2;""
                                             ; (ie same as just CAT+)
         rst      RST_HOOK
         defb     HOOK_PACKAGE

...
         ld       (iy+0),$ff                 ; remember to reset ERR_NR before
...                                          ; returning to BASIC (also restore H'L')

.parambuffer
         defs     32
```

RESI_IM1SVC ($0352)

This call is only for the use of packages. By calling this function, a package can register itself to be called whenever an interrupt occurs. When this happens, the package will continue to receive a call to their standard _EXP call with a reason of A=exp_im1. At this point they can perform any operations they like, although it is desirable to keep processing to a minimum. The package must preserve the BCDEHLIXIY registers.

Packages may register or de-register a handler at any time. The interrupt chain is preserved across resets and power cycles. Therefore, packages which enable/ disable interrupt handlers in response to commands or other events may wish to ensure that they deregister their interrupt handler at boot (when EXP_BOOT is supplied to their _EXP call), in case a power cycle/reset occurred when one was still active. It is also important to remember to de-register your package's interrupt handler before allowing it to be uninstalled (ie when your _BYE call is invoked).

Available from: v2.20

IN: A=reason code:
im1svc_reg (0) to register the IM1 handler for this package
im1svc_dereg (1) to de-register the IM1 handler for this package
OUT(s): Fc=1
OUT(f): Fc=0, A=error code

Register status on return:
......../IXIY same
AFBCDEHL/.... different

Example:
```
         ld      a,im1svc_reg           ; register interrupt handler
         exx
         ld        b,PKG_RESIDOS
         ld        hl,RESI_IM1SVC
         call    PACKAGE_CALL_PKG       ; make the call from this package
```

RESI_TEMPSVC ($0355)

ResiDOS contains an internal 2K temporary buffer, which it uses to hold data from main memory when performing a memory-intensive operation. The buffer is used during the %CP and CAT BASIC commands, and also during configuration file processing operations. At other times, the buffer is available for the use of a package to save and restore data to.

This call is only for the use of packages, and may not be used during an NMI or IM1 interrupt. Filesystem packages should also not use this call at times when they are likely to be called by the ResiDOS config processing or %CP or CAT commands.

Typically, a package can use this buffer during processing of a BASIC command or other call that it is implementing. On return to the caller, it should always release the temp buffer so that it can be used for other purposes again.

To use the buffer, a package must first lock it using the tempsvc_lock reason. This may fail (with rc_resi_no_room) if the temp buffer is locked by another package. Once locked, the package may issue any number of calls with tempsvc_save and tempsvc_restore to save and restore data in the temp buffer. It is up to the package to manage the 2K area as desired. Finally, the package must unlock the buffer with the tempsvc_unlock reason.

```
Available from: v2.21
      IN:  A=reason code:
             tempsvc_lock (0) to lock the temp buffer for this package
             tempsvc_unlock (1) to unlock the temp buffer
             tempsvc_save (2) to save data to the temp buffer
             tempsvc_restore (3) to restore data from the temp buffer

          For tempsvc_save & tempsvc_restore:
             HL=offset into 2K temporary buffer (0...2047)
             DE=memory address to save/restore from
             BC=size to save/restore

    OUT(s):  Fc=1
    OUT(f):  Fc=0, A=error code

    Register status on return:
    ......../IXIY same
    AFBCDEHL/.... different
```

Example:

```
        ld      a,tempsvc_lock          ; lock the temp buffer
        exx
        ld        b,PKG_RESIDOS
        ld        hl,RESI_TEMPSVC
        call    PACKAGE_CALL_PKG        ; make the call from this package
        jr      nc,handle_errors

      ....

        ld      a,tempsvc_save          ; save data to temp buffer
        ld      hl,0                    ; use offset 0 (start of buffer)
        ld      de,$c000                ; save from $c000
        ld      bc,1024                 ; 1K to save
        exx
        ld        b,PKG_RESIDOS
        ld        hl,RESI_TEMPSVC
        call    PACKAGE_CALL_PKG        ; make the call from this package
        jr      nc,handle_errors

        ....

        ld      a,tempsvc_restore       ; restore data from temp buffer
        ld      hl,0                    ; use offset 0 (start of buffer)
        ld      de,$c000                ; restore to $c000
        ld      bc,1024                 ; 1K to restore
        exx
        ld        b,PKG_RESIDOS
        ld        hl,RESI_TEMPSVC
        call    PACKAGE_CALL_PKG        ; make the call from this package
        jr      nc,handle_errors

      ....

        ld      a,tempsvc_unlock        ; unlock the temp buffer
        exx
        ld        b,PKG_RESIDOS
        ld        hl,RESI_TEMPSVC
        call    PACKAGE_CALL_PKG        ; make the call from this package
        jr      nc,handle_errors
```

RESI_STREAM_LINEIN ($0358)

This call reads characters from the currently active stream into a buffer until a CR (ASCII 13) character is received. The CR is not added to the buffer (nor is any other terminator). The return values provide the size of the input data returned and the address following the last data byte (so that the user can easily append a terminator if required).

If the buffer is filled before a CR is received, this call will continue to poll the stream and discard all further characters until a CR is obtained.

This call differs from manually calling RESI_STREAM_IN until a CR is received. Some channels provide a special routine for line input, and this will only be invoked by RESI_STREAM_LINEIN, not by RESI_STREAM_IN. For example, window channels will provide an on-screen user prompt where a line can be typed and edited before ENTER is pressed and the result returned. Using RESI_STREAM_IN would only use the character input routine, which provides no on-screen feedback for window channels.

New channels which provide only character input can still be accessed using this routine, which gives a convenient way of obtaining an entire line of input at once. However, the standard channels (of which only "K" allows input) may not be accessed with this routine - use RESI_STREAM_IN instead for this case.

Note that, in common with the other RESI_STREAM_* calls, errors generated can be ResiDOS/IDEDOS/package errors (as normally returned by other calls) or they may be a BASIC error code (-1). You should check the zero flag to determine which type of error has been returned. Typically, the only BASIC errors generated are likely to be "O Invalid Stream" (Fz=1, A=$17) or "J Invalid I/O Device" (Fz=1, A=$12).

Available from: v2.23

```
      IN:  HL=buffer address
           BC=buffer size
  OUT(s):  Fc=1,
           HL=address in buffer following input data
           BC=number of characters received
  OUT(f):  Fc=0,
           Fz=0, A=ResiDOS/IDEDOS/package error code
                 B=package id (for package error codes)
           Fz=1, A=BASIC error code - 1
```

Register status on return:
......../..IY same
AFBCDEHL/IX.. different

Example:

```
        ld      hl,buffer
        ld      bc,bufsize
        exx
        ld      b,PKG_RESIDOS
        ld      hl,RESI_STREAM_LINEIN   ; read line from stream
        rst     RST_HOOK
```

The IDEDOS Package

This package provides many of the calls available in +3DOS (used on the +3 and +3e) and IDEDOS (used on the +3e). The calls all use the same input and output parameters, but there are some "gotchas":

- 128K memory configurations are not supported. You should always specify 0 for the page number in calls which expect it.
- Some calls return addresses "in page 7" (such as DOS_REF_HEAD). Data at these addresses cannot be accessed directly, since it is not in main memory. Instead, you must use the IDE_ACCESS_DATA call, detailed later.

+3DOS Calls

The following +3DOS calls may be used. Full details are in the +3 manual, which can be obtained from The World Of Spectrum. Calls marked with an asterisk probably won't work as expected, or be very useful.

- DOS_INITIALISE
- DOS_VERSION
- DOS_OPEN
- DOS_CLOSE
- DOS_ABANDON
- DOS_REF_HEAD
- DOS_READ
- DOS_WRITE
- DOS_BYTE_READ
- DOS_BYTE_WRITE
- DOS_CATALOG
- DOS_FREE_SPACE
- DOS_DELETE
- DOS_RENAME
- DOS_BOOT *
- DOS_SET_DRIVE
- DOS_SET_USER
- DOS_GET_POSITION
- DOS_SET_POSITION
- DOS_GET_EOF
- DOS_GET_1346 *
- DOS_SET_1346 *
- DOS_FLUSH
- DOS_SET_ACCESS
- DOS_SET_ATTRIBUTES
- DOS_OPEN_DRIVE
- DOS_SET_MESSAGE
- DOS_REF_XDPB
- DOS_MAP_B *
- DD_SET_RETRY
- DD_SEL_FORMAT *

- DD_ENCODE
- DD_L_XDPB *
- DD_L_DPB *

IDEDOS Calls

The following IDEDOS calls are provided. Documentation is provided at the +3e website.

- IDE_VERSION
- IDE_INTERFACE
- IDE_INIT
- IDE_DRIVE
- IDE_SECTOR_READ
- IDE_SECTOR_WRITE
- IDE_FORMAT
- IDE_PARTITION_FIND
- IDE_PARTITION_NEW
- IDE_PARTITION_INIT
- IDE_PARTITION_ERASE
- IDE_PARTITION_RENAME
- IDE_PARTITION_READ
- IDE_PARTITION_WRITE
- IDE_PARTITION_WINFO
- IDE_PARTITION_OPEN
- IDE_PARTITION_CLOSE
- IDE_PARTITION_GETINFO
- IDE_PARTITION_SETINFO
- IDE_SWAP_OPEN
- IDE_SWAP_CLOSE
- IDE_SWAP_OUT
- IDE_SWAP_IN
- IDE_SWAP_EX
- IDE_SWAP_POS
- IDE_SWAP_MOVE
- IDE_SWAP_RESIZE
- IDE_DOS_MAP
- IDE_DOS_UNMAP
- IDE_DOS_MAPPING
- IDE_DOS_UNPERMANENT
- IDE_SNAPLOAD
- IDE_ACCESS_DATA
- IDE_IDENTIFY
- IDE_PARTITIONS

One slight difference is that for the IDE_DOS_MAP and IDE_DOS_UNMAP calls, the FLAGS3 variable is not used (it doesn't exist on all Spectrums). Instead, you can provide a drive letter as lower-case to make the mapping permanent.

A special call, IDE_ACCESS_DATA, is provided so that data which exists in the *ResiDOS* memory can be read or written by programs. Such addresses are referred to in the +3DOS documentation as being "in page 7". Instead, use this call, with the provided address as the source or destination, and your own buffer as the other address.

+DivIDE Firmware

for

ZX Spectrum DivIDE Interface

USER MANUAL

Version released for RWAP Software DivIDE Plus Interface

Manual Update 1 (February 2007)

NOTICE

The manual for +DivIDE is based on the manual for the PlusD disk interface made available to the public following the cessation of Miles Gordon Technology.

The manual is intended to provide the user with detailed information adequate for the efficient installation and operation of the equipment involved. However, while every effort has been taken to ensure accuracy, the manufacturer assumes no liability resulting from errors or omissions in this manual, or from the use of the information contained herein.

Introduction

The +DivIDE firmware produced for the DivIDE interface is based on the G+DOS firmware first devised by Miles Gordon Technology for their ZX Spectrum Disciple and Plus D floppy disk interfaces and contains various improvements to remove bugs, as well as implement the firmware for use with hard disks and compact flash memory.

In order to make the manual easier to understand, where a number is expected, you will see (n) or similar – this is to distinguish from where the letter itself should be used. Letters are not case sensitive, therefore for example:

```
            LOAD d(n) can be LOAD D1 or LOAD d2
```

Where we refer to an IDE device this can be a hard disk, FlashIDE or Compact Flash memory card connected to the DivIDE Plus interface using an IDE cable or Compact Flash adaptor.

Compatibility

+DivIDE should work with all DISCiPLE and +D software which doesn't call routines in G+DOS directly. At the moment it is not 128K mode compatible - it does work in 48K mode on an original 128K.

SATA hard disk drives, CD and DVD drives are not supported. Maximum drive size is 50GB.

How to Access IDE Devices

In order to retain compatibility with G+DOS, +DivIDE uses virtual disks of 1600 sectors each on the IDE device.

You have to identify which virtual disk is to be used by the G+DOS disk access commands (such as `LOAD D1 "test"` or `CAT 1`). In order to do this, you need to use the `GOTO *` command.

```
GOTO *(d);(m),(n)
```

Where:

`d` is the disk access number (1 or 2 at present),
`m` is the IDE device number (0 for Master, 1 for a slave drive)
`n` is the number of the virtual disk on that IDE device (0-65535)

With a fixed 1600 sectors per virtual disk, this means that +DivIDE can handle a maximum of 1600x65535 sectors on a hard disk. With each sector being 512 bytes, this gives a maximum disk size of 50GB that can be accessed.

Each virtual disk can be a maximum of 800K as per the original G+DOS allocation scheme, with the first 10K set aside for the virtual disk catalogue. The catalogue (or directory) of a virtual disk can be viewed using the command:

```
CAT (d)
```

This also means that there is still a maximum of 80 files on each virtual disk.

The Catalogue (or file directory)

Type `CAT 1` (or `CAT 2` if you wish to access drive 2) - `ENTER`. Please note that these are the virtual drives identified previously with the `GOTO *` command.

In about two seconds the screen will display:
- The heading +DivIDE Disk 1 (or 2) Catalogue.
- A list of the files on the disk with various comments (The maximum number of files per disk is 80).
- The amount of free space on the disk in kilobytes.

A typical catalogue screen looks like this:

```
 1 +SYSTEM 14 CDE 8192,5656
 2 CONFIG 31 BAS 9100
 3 CONFIG1_C 4 CDE 40300,1750
 4 CONFIG2_C 14 CDE 42240,6656
 5 CONFIG3_C 1 CDE 49000,60
 6 snap F 97 SNP 48K
 7 screen 14 SCREEN$
 8 listing 34 M/DRIVE
 9 data 8 SPECIAL
 10 numeric 4 D.ARRAY
 11 character 5 $.ARRAY

 Number of free K-bytes = 667
```

Most of the information on this catalogue screen is automatically displayed by +DivIDE. The only information which you have to supply is the file-name, which is shown in the second column. These file-names can be up to ten characters long, in upper case or lower case characters, or both.

The first column on the left shows the program number. Whenever you save a file, +DivIDE will give it the first available program number. So, if you have the catalogue shown in the example above, the next file you save will automatically become program number 12. This program number will stay the same until the file is erased (we do not list files in alphabetical order). But if you erased, say, program number 8, then the next file you save to the disk will become the new program number 8 – taking the first available program number.

The third column shows the number of disk sectors used. Each sector holds 512 bytes (= ½ kilobyte), so to find out the number of kilobytes used for each file, divide the number in the third column by 2.

The fourth column is for the type of file, which +DivIDE automatically selects. The different types are shown in the table at the top of page 7. You probably won't fully understand the significance of all these different types of file yet, but most of them are covered in this Introductory Manual. Those marked with an asterisk are likely to be used by more experienced programmers and demand a fuller explanation than we can give in this Manual.

BAS	= Basic
CDE	= Code
SNP 48K	= 48K Snapshot File
SNP128K	= 128K Snapshot File
SCREEN$	= Screen File
M/DRIVE	= Microdrive File *
SPECIAL	= Special File *
D.ARRAY	= Data Array
$.ARRAY	= Character Array
OPENTYP	= File created by OPEN # statement *
EXECUTE	= Execute File *

+DivIDE File-Types

The right-hand column in the catalogue tells you the starting address of the file, and, after the comma, the number of bytes used if it is a code file. If it is a file in BASIC, the number will show the line at which the program starts.

The amount of free space left on the disk is displayed in kilobytes at the bottom of the screen.

Remember that +DivIDE displays all this information automatically. All you'll have to do is to select a name for each file - and we'll explain how to do this later.

If your catalogue fills the screen, and you wish to scroll down to the next screen, press ENTER.

You can also display an abbreviated catalogue by typing CAT 1! (or CAT 2!). This would show the same catalogue in this format:

```
+SYSTEM CONFIG CONFIG1_C CONFIG2_C
CONFIG3_C snap F screen listing
data numeric character
```

Both the full and the abbreviated catalogue can be listed on screen simultaneously. Or you could, for example, list disk 1 in full and disk 2 in the abbreviated form (but you must list the full catalogue first, as CAT 1 clears the screen, while CAT 1! does not)

If you wish to print out the catalogue on your printer, you can use the special CAT #3 command. For example, to send the catalogue on drive 1 to the printer, you enter:

```
CAT #3;1
```

To print an abbreviated catalogue from drive 2, you enter:

```
CAT #3;2!
```

+DivIDE Disk Drive Commands

When you use DivIDE Plus and a disk drive, BASIC commands typed on the computer keyboard are followed by a Syntax Operator – a code to let the computer know that the instructions are directed toward the disk drive, and not, for example, to a tape recorder.

For ease of explanation in this Manual, the syntax operator will always be referred to as Dl, meaning that the instruction is for drive number 1. However, when you are working on the computer, the following options are also available:

Dl or dl = a command for drive 1
D2 or d2 = a command for drive 2
D* or d* = a command for the last drive in use

(Programmers should note that D* is particularly useful when your disk file autoruns and calls up another file on the same drive. Also note that a variable can be assigned for drive number 1 or 2 but not for *.)

By typing an upper case D rather than a lower case d when saving a file, the abbreviated disk catalogue will be displayed automatically when the save is complete. A new catalogue also appears automatically if you type an upper case D when erasing or renaming a file.

Saving a File and Verifying

First, type in this short program called "Squares". We'll use this program to show +DivIDE's various disk operations:

```
10 REM Squares
20 FOR n=1 TO 10
30 PRINT n,n*n
40 NEXT n
```

Then, to save the program on the disk in drive 1, ENTER:

```
SAVE Dl "Squares"
```

The screen border will flash (this can be configured), and after about 2 seconds the program will be saved.

As described in the last section, if you have saved using a capital D, the abbreviated disk catalogue is automatically displayed. You'll see the program Squares shown on the catalogue as the last entry. But if you wish, you can do a further check to ensure that the program has been correctly saved, by entering:

```
VERIFY Dl "Squares"
```

When the O.K. message confirms the save, clear Squares from the computer's memory by entering `NEW`.

Loading a Program

Now you can reload Squares from the disk, by entering

```
            LOAD D1 "Squares"
```

When the O.K. message appears, the program has been loaded, But so far we haven't given an instruction to make the program run. Let's do that now by entering:

```
            SAVE D1 "Squares" LINE 10
```

But of course, there's already a program called Squares saved on the disk. +DivIDE tells you this and asks you whether you wish to overwrite the existing file by entering Y (= Yes) or N (= No)

Enter Y. Then:

```
            LOAD D1 "Squares"
```

Breaking into a Program

To break into the program and stop Squares running, use the normal Spectrum BREAK key.

Note that BREAK will stop the computer running all normal routines, but it will have no effect during a disk read/write (i.e. Load or Save) operation until the disk activity has been completed.

Renaming a File

To rename a file, we use the two keywords ERASE and TO. For example, let's rename the Squares file as Example 1:

```
            ERASE D1 "Squares" TO "Example 1"
```

The abbreviated catalogue will confirm that the change has been made.

The file-names you choose can be any combination of letters and numbers - or even spaces - but the maximum number of characters is ten. You can use upper-case or lower-case characters, or both. In the catalogue, the file-names will appear exactly as you originally typed them, but when you re-load a file it doesn't matter if letters originally typed in lower case are changed to upper case, or vice versa.

Warning: Don't use the characters ? or * in your file-names, as this may cause problems with other commands we deal with later.

Merging a Program

If you wish to merge a file on disk with the program currently in memory you can do this using the command:

```
            MERGE Dl "Squares2"
```

Please note that any line numbers in the merged program which already exist, will be overwritten by those from the disk.

Erasing a File

Let's now erase the file called Example 1, by entering:

```
            ERASE Dl "Example 1"
```

After the erase, your abbreviated catalogue will confirm that Example 1 is no longer there. Enter CAT 1 for the full listing, and you'll see that more space has been made free on the disk (although this is such a small program that any change will be minimal!)

(Note that unlike the ZX Interface 1, +DivIDE returns an error message if you try to erase a non-existent file. When using programs originally written for Microdrive, you may sometimes need to delete ERASE commands in the listing to avoid the error message when the program is running.)

Creating an Autoload File

An autoload file is one which loads as soon as you connect the interface and enter the command RUN. You are allowed one autoload file per virtual disk.

For an illustration of an autoload file, let's rename the file Newcopy. Enter:

```
            ERASE Dl "Newcopy" TO "Autoload"
```

When the catalogue confirms that the change has been made, turn off the power to your Spectrum so that you lose everything in the computer's memory. (Turn off the power - don't use the Spectrum's reset button - we're coming to this in the next section.)

Turn on the power again and enter the command RUN. Your Autoload program (which, you'll remember, is the original Squares program) will be loaded at the same time, and instead of the normal +DivIDE title screen, you'll see the Squares program running.

Snapshot Files

For many +DivIDE users, one of the most important features will be the interface's ability to transfer virtually any piece of Spectrum software to disk easily and instantly by means of the Snapshot (NMI) Button. You can take Snapshot saves at any time during a program and as often as you like; if you are a games-player, you'll find it especially valuable to be able to:

- start a game at an advanced level instead of at the beginning
- come back to a game on another occasion and start at exactly the point where you had to leave it last time
- keep a permanent record of your high scores.

A Snapshot save is made when you press the Snapshot (NMI) Button on the DivIDE Plus once. Anything and everything held in the Spectrum's memory at that time is saved to the last drive in use, shown by the indicator light on the disk drive.

But there are three different types of Snapshot file.

To experiment, first make sure that your system has been booted up. then load in any program (a game is best for demonstration) in the normal way - from disk or from cassette. (If you're loading from cassette, you should of course use the Spectrum's normal LOAD "" command, and not the special +DivIDE disk syntax.)

When the program has loaded and started to run, press the Snapshot (NMI) Button. You'll notice that your program will be frozen at its current status, and you will see a number of multi-coloured lines filling the borders on the screen. You now have three Snapshot save options:

Press key 3 on the Spectrum to save only the current SCREEN. A screen save will occupy 7K of memory space on a disk. When the save is complete, you'll see that the program will continue at the same point from which you left it.

Press key 4 on the Spectrum for a 48K PROGRAM to be saved to disk. Every time you save a 48K program, 48K of memory space (or actually 97 sectors) will be used on the disk. When the save is complete, in just over 3 seconds, the program will continue from the point at which you left it. If you want to, you can take more Snapshots at later points in the program until your disk is full - that's up to 16 Snapshots per disk on a 780K capacity drive.

Press key 5 on the Spectrum for a 128K PROGRAM to be saved to disk. Every time you save a 128K program, 128K of space will be used on the disk (= 256 sectors). The procedure here is slightly different. The save will commence (and if you have kept flashing borders in your system file, you'll see the borders flashing), but then the program will appear to be frozen again.' If the picture on your screen (the picture - not the border) has changed, press key Y (= Yes) on the Spectrum: if it has not, press key N. The 128K save will then be complete in about 10 seconds and the program will resume from the point at which you left it.

If you have loaded the program on drive 1, but wish to save your snapshot file to drive 2 - or vice versa - press the CAPS SHIFT key on the Spectrum at the same time as pressing key 3, 4, or 5.

If you hit the Snapshot (NMI) Button accidentally, and wish to return immediately to your program, press key X on the Spectrum.

When you've saved a Snapshot file, press the computer reset button once (or in the case of Spectrums without reset buttons, turn off the power, then turn it back on again and reboot the DivIDE Plus system). Then call up the catalogue - enter CAT 1.

You'll see that your Snapshot file has been automatically labelled "Snap A/B/C" etc. with the letter of the alphabet dependent upon the file's position in the catalogue. Clearly, it would be difficult to remember the precise contents of your snapshot file if this were the only name possible. So you'll need to give your Snapshot file an identifiable file-name.

First choose a new name for the file you have snapshotted and then rename it using the `ERASE....TO` keywords.

Then, to reload Snapshot files, you enter:
`LOAD Dl "Your new file-name" S` (for a 48K file)
`LOAD Dl "Your new file-name" K` (for a 128K file)
`LOAD Dl "Your new file-name" SCREEN$` (for a screen file)

NOTE: Without the S, K and SCREEN$ identifiers, the Snapshot files will not load - unless you use the special abbreviated syntax.

Problems?

1. I've found lots of programs I can't snapshot.
We can't guarantee 100% success with snapshotting, but we're pretty close. If you're getting failure with several of your programs, there's probably a problem with your equipment. If you find that you can only snapshot a small number of programs per disk, this strongly suggests a problem with the heads on your disk drive.

2. I can't copy my Snapshot files using `MOVE....TO`.
Quite right - this is a security precaution against Illegal professional copying of commercial software.

Abbreviated syntax

Now that you've learnt the basic disk drive commands, we can introduce you to a much easier way of loading a file, You can just enter:

```
        LOAD 6
Or      LOAD p6
```

This has the effect of loading the file labelled number 6 in the catalogue. If you use this command you don't need to worry about the S, K or SCREEN$ identifiers to load snapshot files. It also makes it much easier to load code files. Normally, you would have to type, for example:

```
 LOAD Dl "CONFIG1_C" CODE 40300,1750
```

stating the start address of the file and the number of bytes used.

Instead, you can now simply enter

LOAD 3. Try it.

MERGE and VERIFY can also be used with the program number in this way - but ERASE cannot.

Wild-Card Files

Wild-card files allow you to CATalogue, copy (using SAVE...TO) or ERASE a group of similar files in a single operation. Let us imagine, for example, that you have a series of files called:
numbers1, numbers2, numbers3, numbers4

To erase all of them you can enter:

```
ERASE Dl "n*"
```

The effect of the star is to say that all the rest of the file-name doesn't matter: you wish to erase all files commencing with "n". Be careful: this really means all files beginning with "n".

You could also enter:

```
ERASE Dl "??mb*"
```

You would then erase all the files which have "-mb-" as their third and fourth letters. The effect of each question-mark is to say that "this letter is of no significance".

Formatting New Virtual Disks

Before a virtual disk is used for the first time, it must be formatted. You can also format a previously-used virtual disk: the effect will be to wipe the virtual disk clean of information. You must therefore be careful not to format a disk which contains any files you may wish to keep (and let's say it again - ALWAYS make back-up disks for important files!). It is also risky to format a disk while you are holding a program in the Spectrum's memory, in case the program is overwritten with format information.

To format disks. enter:

```
 FORMAT Dl
```

ERROR REPORTS

There are a number of standard Spectrum error messages which will appear on screen if the command you have typed in cannot be obeyed. For full details, consult the Spectrum manual. When you are using +DivIDE, these standard error messages will continue to appear, but there are some additional error reports which refer specifically to disk handling problems. These are some of the more important you may meet.

Nonsense in Basic

This is a standard Spectrum error report, and if it appears as soon as you start trying to use +DivIDE, it suggests that the connection between the computer and the DivIDE Plus is not being properly made, so the special DivIDE Plus commands cannot be recognised, Or it may mean that you have typed a command incorrectly - try again.

Format Data Lost

This means that the disk has been damaged or the file has been corrupted - perhaps you've been working too close to a magnetic field. You won't be able to access the current file, and other files may also have been lost.

Copy any remaining files to another disk immediately. Try reformatting the original disk, and if there's no permanent damage, you may be able to use it again, But don't risk storing important files on this disk.

Sector Error

This means that the information you've tried to save has been corrupted. You'll need to rewrite the file and save it again. But your disk has not been damaged and need not be reformatted.

The numbers after the error message tell you which track and which sector have been corrupted, You should always make a back-up copy of key files.

If the Sector Error or Format Data Lost messages appear regularly, you may have a problem with your disk drive. The bead may be dirty, or the drive may not be centring the disks correctly on insertion. Try cleaning the heads. If this fails, repairs may be necessary.

Disk Write Protected

You cannot save, erase, or otherwise modify files on this disk until you uncover the security lock on the disk. (Use the security lock on the disk to keep important files safe.)

File Not Found

You are trying to load, copy, verify, erase or rename a file which is not present on the disk. Check that you have typed in the file-name correctly. Make sure that you have used the file identifier - S, K or SCREEN$ - if necessary.

ADVANCED DISK OPERATIONS

Sector by Sector Copying - Two Drives

```
FORMAT Dl TO 2
```

- formats virtual disk 1 (and thus wipes it clean of data) and copies the contents of the virtual disk 2 to virtual disk 1, sector by sector. BE VERY CAREFUL NOT TO FORMAT THE WRONG DISK.

Because the copying is done a sector at a time, the copying process takes much longer than the MOVE...TO command; but the FORMAT.. TO command will give you a disk which is identical in every respect to the one which has been copied - useful if you are investigating corrupt sectors.

Reading and Writing to a Sector

```
LOAD @ D,T,S,Address

SAVE @ D,T,S,Address
```

These commands are used for reading from or writing to a specified track and sector on the disk to a memory location in the Spectrum memory area, where:
D = the drive number (type either 1 or 2 - not the D)
T = the track number from 0 to 39 (40 track) or from 0 to 79 (80 track)
To read the second side on a double-sided disk drive use 128 to 167 (40 track) or 128 to 207 (80 track)
S = the sector number - from 1 to 10
Address = the start address of any location in the Spectrum memory. Nb: 512 bytes will be used.

Working with a monitor program, this powerful command allows you to access a single sector on the disk from Basic. Particularly useful for breaking into, examining and modifying a Snapshot file.

Using Streams and Channels

i) `OPEN #S; Dl "file-name" IN` or `OUT`

This command is used to open a file on the disk and attach a stream number to it, so that you can print out to or input from that Stream, where:
S = Stream number - from 4 to 16 (type the number only)
D = Dl or D2 (drive number - type D as well as drive no.)
IN = When you specifically need an input or read file opened
OUT = When you specifically need an output or write file opened

If you choose not to specify IN or OUT, then the default is the same as Interface 1: an input file if the file-name is found in the catalogue, and an output file if the file-name is not found.

ii) `CLOSE #*S`

This command closes the file and stream opened in the previous command. S = Stream number. If you do not type the stream number, then all current open streams will be closed. Note the addition of the * to Microdrive syntax.

iii) `MOVE`
This is a command which moves a file, one sector at a time, to either another file or a stream.

```
            MOVE Dl "file-name" TO #S
```

This command reads the file and outputs it to the specified stream (where Dl = Dl or D2, and S = the stream number, from 4 to 16).

```
            MOVE Dl "file-name" TO Dl "file-name"
```

This command reads the file and writes it to the second file.

iv) `CLEAR #`
This command clears all open streams and channels, but it does not `CLOSE` a file.

v) `CLS #`
This command clears the screen and attribute area, and resets `BORDER`, `PAPER`, `INVERSE`, `BRIGHT`, `OVER` and `FLASH`.

The Execute File

An execute file is a machine code file occupying up to one sectorlength of memory (510 bytes) . When loaded from disk it is executed (`RANDOMISE USR`) in the DivIDE Plus RAM, and not in the Spectrum’s memory as other files are. An execute file gives the programmer the ability to execute utility routines - such as renumbering a program - without affecting or using the Spectrum memory.

To save a sector length of memory as an execute file, enter
 SAVE Dl "file-name",X,Address
where:
 Dl = Dl or D2 (drive number)
 X = the execute file identifier - just type X
 Address = the location in the Spectrum of the machine-code program, which has been assembled to run at 3BD6 hex.

To load the execute file and run it internally in the DivIDE Plus, enter:
 LOAD Dl "file-name" X
or LOAD p(n) - in other words, the normal abbreviated syntax for LOAD.

After execution, control is returned to the Spectrum.

QUICK REFERENCE GUIDE

BASIC

`GOTO *(d);(m),(n)`	Specify location of virtual disk
`RUN`	Runs an autoload file on Virtual Disk 1

READ DISK

`CAT (n)`	Displays catalogue
`CAT (n)!`	Displays shortened catalogue

LOAD FILES

`LOAD D(n) "file-name"`	Loads file (except Snapshot files)
`LOAD *(n);"file-name"`	Loads file (OPUS Discovery syntax)
`LOAD D(n) "file-name" S`	Loads 48K Snapshot file
`LOAD D(n) "file-name" K`	Loads 128K Snapshot file
`LOAD D(n) "file-name"SCREEN$`	Loads screen file
`LOAD (n)`	Loads the program (from its number)
`LOAD p(n)`	Loads the program (from its number)

`MERGE D(n) "file-name"`	Merges file with memory (except Snapshot file)
`MERGE *(n);"file-name"`	Merges file with memory (OPUS format) **CHECK**
`MERGE p(n)`	Merges file with memory (from its number)
`MERGE (n)`	Merges file with memory (from its number)

SAVE FILES

`SAVE D(n) "file-name"`	Saves file
`SAVE *(n);"file-name"`	Saves file (OPUS Discovery syntax)
`VERIFY D(n) "file-name"`	Confirms save has been made
`VERIFY *(n);"file-name"`	Confirms save (OPUS Discovery syntax)

DELETE FILES

`ERASE D(n) "file-name" TO "new file-name"`	Renames a file
`ERASE D(n) "file-name"`	Erases a file
`ERASE *(n);"file-name"`	Erases a file
`ERASE (n);"file-name"`	Opus Discovery **CHECK**
`ERASE (n)`	**CHECK**
`ERASE p(n)`	**CHECK**

FORMATTING DISKS

`FORMAT D(n)`	Formats virtual disk in drive n

`FORMAT D1 TO 2` Formats drive n and copies contents of drive 2 to drive 1 ***BEWARE***
`FORMAT 1` (OPUS Discovery) **CHECK**

STREAMS

`OPEN #(n);D(n) "file-name" IN or OUT`	Opens a file; attaches stream no.
`CLOSE #*(n)`	Closes a file & stream no.
`CLOSE #*`	Closes all open files
`MOVE D(n) "file-name" TO *(n)`	Reads file and outputs it to specified stream
`MOVE D(n) "file-name" TO D(n) "file-name"`	Reads file and writes it to a 2nd file

`CLEAR #`	Clears all open streams & channels
`CLS #`	Clears screen & attributes area

`SAVE D(n) "file-name"X,Address`	Saves sector length memory to disk
`LOAD D(n) "file-name"X`	Executes file & runs it in DivIDE Plus RAM

DIRECT SECTOR ACCESS

`LOAD @ D,T,S,Address`	Reads a specified track/ sector to memory
`SAVE @ D,T,S,Address`	Writes to specified track/ sector on disk

WILDCARDS

`ERASE, CAT`	can be used with * and ? for wild-card operations

Snapshot (NMI) Button halts program temporarily & allows selection of:
3 - Screen save;
4 - 48K Snapshot save;
5 - 128K Snapshot save;
Key X breaks out of Snapshot and returns to program

DEMFIR Firmware

for

ZX Spectrum DivIDE Plus Interface

USER MANUAL

Manual Update 1 (August 2009)

Introduction

DEMFIR is similar to FATware and handles the ISO 9660 file system found on CD-ROMs. You can use either CDs or unfragmented ISO images stored on a hard disk or compact flash card.

DEMFIR supports TAP, TZX, SNA, Z80, MFC, POK and SCR files, but as with FATware is currently read only (unless you try the v0.8 beta version). Note that the TZX support does not support non-standard loading techniques.

If using DEMFIR with a CD please note that you should put your CD into the CD-ROM drive and turn that on and allow it to start up BEFORE you switch on the Spectrum.

Starting Up DEMFIR

DEMFIR comes in two parts – the part installed on the Flash EPROM and a second part which needs to be downloaded and installed on the first available ISO image – this can be the generic boot image (demfirXXXX_R.bin) placed in the first sector of the image, or demfirboot.img used on a standard bootable CD made with software on your PC such as Nero. The image file can be downloaded from http://demfir.sourceforge.net/. You may also like to try the beta test version of DEMFIR v0.8 which includes the ability to save to a .tap file – this is available from: http://speccy.trilogic.cz/tritol/demfir-80324.tar.gz

v0.8 includes an empty.tap file, which you can overwrite by writing to from the ZX Spectrum if you are using a compact flash card or hard disk with an ISO image.

Once the drive is up and running, turn on the Spectrum and then press the NMI button to start up DEMFIR. You will see the DEMFIR version message (D0.7b) appear in green in the top right hand corner – if it appears in red, then something has gone wrong with the install. Press the ENTER key to browse the ISO image and find the files you need.

If the ISO image is not the first file on the hard disk or compact flash card, then you can use the search option on the startup menu to find the ISO image you wish to work with.

Using DEMFIR

The integrated file browser when you press the NMI button, makes finding and loading the various file formats easily. As with FATWare, if you select a .tap or .tzx file, you will need to enter `LOAD ""` in BASIC to load the file (otherwise `LOAD ""` will try to load a file from cassette as normally).

The NMI Menu

Pressing the NMI button on the interface whilst DEMFIR is chosen as the firmware, pulls up the NMI menu.

The first line of the NMI menu shows (from left to right)

- Interrupt Status (IM1/ IM2, EI / DI)
- Actual Mode (48K / 128K)
- Active 128K page
- Selected Virtual Memory (VRAM)

The second line of the NMI menu shows a list of supported formats, as these can be configured within config.a80 when DEMFIR is built:

- TAP - Standard emulator compatible file - emulates a cassette tape
- TZX - Standard emulator compatible file
- Z80 – Z80 snapshots
- SNA – SNA snapshots
- MFC – MFC snapshots
- SCR – screenshot view
- POK – PokeManager
- DFW – DivIDE Firmware – file for other MAPRAM firmware installation

The third line of the NMI menu shows other settings (again configured within config.a80 when DEMFIR is built):

- CHS – Support for HDD without LBA is enabled
- WDC – Automatic disk geometry pre-set during device detection (required for some Western Digital hard disk drives)
- +3R – Changes ROM of +2A/+3 via port 8189 (?)
- MTF – TZX support for nontap blocks is enabled
- SSv – Screensaver enabled

Finally, you can also change the logo within config.a80 (USELOGO 1,2 or 3) and enabled BOLD_FONT to change the font used within the NMI menus.

The NMI menu gives you the ability to undertake the following tasks by pressing the following keys (the first key operates the command):

`ENTER`

Switch to the file browser.

`D`evice

Switch between the master and slave devices connected to the DivIDE Plus

`M`edium

Re-read the medium and detect the ISO image.

`R`eset

Soft reset of the Spectrum. Press the `Symbol Shift` key at the same time to clear the RAM used by DEMFIR also.

`I`nit

Detect new devices. Press the `Symbol Shift` key at the same time to bypass the automatic detection of device size – DEMFIR will assume the maximum supported size (128GB for LBA devices and 8GB for CHS devices). In the case of CHS hard disks, this may assist with some troublesome disks.

`Q`uit

Exit the NMI menu and switch to video RAM mode – see `V`RAM below.

`F`ind

In the case of a hard disk or compact flash card, this will find all ISO images on the device (this can be slow). Press the `Shift` key at the same time to enter a start position for the search. Press the `SPACE` key to stop searching.

`S`ector

Enter the sector where the ISO image is expected, or where you wish DEMFIR to start looking from. You need to enter the sector number in hexadecimal.

`V`RAM

This shows the actual Video RAM and is automatically entered after you `Q`uit the file browser. Press `V` again to switch between the 1st and 2nd video RAM locations (on a 128K machine). This can be useful to check if a screenshot is what you expect. If you do not press anything for 2 seconds, you will return to the NMI menu.

`P`aging

This disables the 128K paging (is equivalent to `OUT 32765,48`). You will have to reset the Spectrum hardware to re-enable paging.

`C`heat

This enters the PokeManager, where available and reads in .pok files from the ISO image, containing cheats for Spectrum programs. It lists the cheats and current state. If no cheats exist, then it enables you to enter `POKE`s by hand – press `CAPS SHIFT 1` to escape.

The File Browser

Once in the file browser, use the cursor keys to navigate:
Up, Down – move through the file list
Left – Go to the top of the current page – if cursor is already at the top, it will go to the previous page.
Right – Go to the bottom of the current page – if cursor is already at the top, it will go to the next page.

`ENTER`

Load program, or if it is a screenshot, view the image (press any key to exit). If it is a directory which is highlighted, the file browser will then open that directory.

`SYMBOL SHIFT + ENTER`

Jump to `LOAD ""` when a .tap file is selected.

`SYMBOL SHIFT + W`

V0.8 only. Use this to highlight the empty.tap file and then press these keys to mark it as writeable. Note that empty.tap has a fixed size of 128K and you can only save to it once – use TZX2TAP and then TAP2TZX to truncate the saved file on your PC if using on a 48K ZX Spectrum system.

`SPACE`

Exit File Browser.

The File Browser – Search by Name

Whilst in the File Browser, pressing any other key allows you to start entering a filename to search by name. The search is case sensitive. Press the . (full stop) key to jump to the start of the directory, you can also jump to the end of the directory by searching for “pipe” (or pressing `SYMBOL SHIFT SPACE`)

If you wish to search for characters on the same keys as the control keys (eg 5,6,7 and 8), press `CAPS SHIFT 3` first – the border will change to blue and the next key you press will use the ASCII character, rather than thinking it is a command.

As you enter the search term, the cursor will move to the first filename which matches the search term. The search term is entered in overwrite mode, and the following keys can be used to edit the search term:
`CAPS SHIFT 1` – exit without storing the value
`CAPS SHIFT 2` – clear the whole search term
`CAPS SHIFT 4` – insert space to the left of the cursor
`CAPS SHIFT 5` – move cursor left
`CAPS SHIFT 6` – move cursor to the start of the search term
`CAPS SHIFT 7` – move cursor to the end of the search term
`CAPS SHIFT 8` – move cursor right
`CAPS SHIFT 9` – delete character on the right of the cursor
`CAPS SHIFT 9` – delete character on the left of the cursor
`ENTER` – exit and store the value, finding the first matching filename

Search is cancelled as soon as you use one of the other commands above.

The File Browser – Working with .tap or .tzx Files

When you select a .tap or .tzx file from within the file browser, DEMFIR checks the validity of the selected file. The cursor will remain green if the file is valid, or change to yellow if the file is valid, but contains too many files for DEMFIR to index, or red if the file is broken. Further, if the cursor turns magenta, it indicates that there is a block within a .tzx file which may make it unreliable.

When `LOAD`ing from a .tap or .tzx file, DEMFIR remembers the position within the file, and will automatically “rewind” the file to the start once the file end is reached.

The PokeManager

The PokeManager is useful for controlling game cheats. It provides a list of all cheats currently stored on the ISO image, listing the name of the cheat and indicating whether it is active (YES) or inactive (NO). In the case of cheats which require a specific value entering (eg. Number of lives), the value is also shown.

To use, load a game into the ZX Spectrum, and then press the NMI button. Press the `C` key to enter the PokeManager

The controls for the PokeManager are the same as for the file browser, with the following additional keys (again the first key operates the command):

`A`dd

Add a `POKE` through manual entry. This will ask you to enter a string in the format: "page:address,value". You will need to enter the 128K RAM page number (0-7, 8 to ignore), and then the memory location (23552-65535), followed by the value to be `POKE`d into that memory location. The original value (if known) is shown, rather than the current memory location.

`D`elete

Delete the last `POKE` or cheat in the list.

`C`lear

Deletes all cheats and starts the manual entry of a `POKE`.

The cursor colour is a helpful indication of the validity of cheats:

- Green – indicates the original value is known and the cheat can be turned ON or OFF.
- Yellow – the original value is unknown, therefore only use the cheat if you are certain that you have the correct .pok file – do not try to turn the cheat OFF.
- Red – the original value is known, but neither the original value nor the cheat value is actually in memory – therefore turning the cheat ON or OFF will probably cause memory corruption.

MDOS3 Firmware

for

ZX Spectrum DivIDE Plus Interface

USER MANUAL

General DivIDE Version

Manual Update 1 (February 2010)

What is MDOS3?

MDOS3 is a modified version of the file system used on the Czechoslovakian Didaktik D40 and D80 disk drives. It can access physical floppy disk drives, virtual hard disks saved as an .md3 hard disk image file on an IDE hard disk, CD-ROM or compact flash card and can even play audio CDs.

You need to ensure that your compact flash card or hard disk has at least a .md3 hard disk image file on it, otherwise *MDOS3* will not start up. A suitable .md3 file appears at: http://velesoft.speccy.cz/mdos-discs.zip

Windows based utilities are also available for writing to / reading from the MDOS3 .md3 hard disk image files - see http://velesoft.speccy.cz/zx/divide/divide-mdos3.htm.

If you want access to more than one .md3 hard disk image file, you will need to partition your hard disk (or compact flash card), as MDOS3 can only recognise one .md3 hard disk image file per partition.

Within the hard disk image file, *MDOS3* expects you to have a series of virtual floppy disks which are identified by ending in .d80. The RealSpectrum emulator allows you to create .d80 files and there is also an archive of D80 files held at: www.frgt10.wz.cz (although most of the programs are in Czech).

When creating .d80 virtual floppy disks, it is recommended that you separate each .d80 file to ensure that there is sufficient room for them to be filled up to their maximum. To do this you need to copy the infosec.bin file, followed by the .d80 file, and then free.bin file.

A dedicated page on *MDOS3* also appears at http://ci5.speccy.cz/mdos3/ (unfortunately it is mainly in Czech).

Using MDOS3

MDOS3 allows you to use 4 virtual disks at a time, lettered from drive A: through to drive D:

The ZX BASIC `LOAD` and `SAVE` commands default to drive A: although you can perform various drive commands, using a format similar to DOS on a PC see the command reference below.

Before you can mess with drives, you need to set them up, using the NMI Menu.

NMI Menu

Pressing the NMI button at the rear of the DivIDE Plus interface enables the *MDOS3* filemanager. You see a screen on which appears a list of four items at the top which correspond to the drives A: B: C: and D: as they would be seen from the ZX Spectrum.

You can set each of these drives to look at physical floppy disk drives, or to a virtual floppy disk drive contained on a .md3 hard disk image file. To do this, press the keys 1-4 (where 1 represents drive A: and 4 represents drive D:). This allows you to access the Drive Settings sub-menu.

Altering Drive Settings Sub-Menu
Having pressed the key 1,2, 3 or 4 you can are given the option to point the chosen drive to the physical floppy disk drives (fp0 or fp1) or to the .md3 hard disk image file (HDD).

You can also change the following settings for the chosen drive:
a) Change the drive allocation letter by pressing the keys 0 or 9 to change the drive letter (these keys cycle through the letters A: to D:).
b) Set the read-only status of the drive (press the Y key)
c) Alter the cassette tape emulation mode by pressing the T key.

The tape emulation mode forces the ZX Spectrum to use a drive as though it was a cassette tape, which allows you to use the normal `LOAD ""` and `SAVE ""` commands.

To do this, select a file and press the T button which circles through various options indicated by letters below the drive names:

--	No tape emulation
L-	Used for LOAD command
-S	Used for SAVE command
LS	Used for both LOAD and SAVE

As you can see, this allows you to use different drives for tape loading and saving.

Allocating Virtual Floppy Disk Drives
For each drive that is set as pointing to a HDD device, you will need to select the .d80 virtual floppy disk contained within the .md3 hard disk image file to be pointed to by the chosen drive letter.

If you have partitioned your compact flash card (or hard disk), then you can switch to a different partition (which must also contain a .md3 hard disk image file) by pressing the W key to change the partition.

Pressing the drive letters A to D enables you to find the .d80 virtual floppy disk which you wish to allocate as that drive - this will overwrite the original setting.

Once you press A to D, you can enter a wildcard mask to find the file you want - press ENTER if you want to browse through the .md3 hard disk image file yourself. The mask allows you to enter wildcards, such as test*.* which will find all files beginning with test (such as test.d80, test1.d80 and tester.d80).

The results of the search are presented in pages of 20 filenames - use the cursor keys to move up and down the list, or the right and left cursor keys to move between pages. Press ENTER to select the chosen file.

You can press the C key when a file is highlighted to get a catalogue of the .d80 virtual floppy disk - this will return the name of the disk, the number of files and free space. Each file held on the virtual floppy disk is then listed, together with their length, start address, BASIC memory size (excluding variables) and file attributes.

The CAPS SHIFT C key can also be used to just list the executable files on the .d80 virtual floppy disk.

If you make a mistake and do not wish to continue with the search, you can press EDIT (CAPS SHIFT + 1) whilst in a text entry field, or BREAK (CAPS SHIFT + SPACE) during the actual search mode.

Finally, if you know the name of a file contained within a .d80 virtual floppy disk but are unsure which virtual floppy disk it is, press the F key to enter a search mask. CAPS SHIFT+f can also be used to use a case sensitive search. Again, wildcards are allowed.

Other Options
You can change the name of the virtual floppy disk drive by pressing CAPS SHIFT+the drive letter key (A to D). This name can then be used in the *MDOS3* commands in place of the drive letter.

You can also enable/disable write protection for the virtual floppy disk drive by pressing SYMBOL SHIFT+the drive letter key (A to D)

Other buttons allow you to flick to the Spectrum screen (press the E key) or return to BASIC and save a snapshot of the current program (S key) or return to BASIC and save the screen (P key).

You can also enter a machine code monitor by pressing the V key, or play an audio CD by pressing the L key.

Exit the NMI button
Once you have set up the virtual floppy disk drives as required, press the Q key to exit the NMI menu and return to 48K BASIC.

NMI Menu Keys Summary:

1,2,3,4	Change Drive settings
0,9	Change actual drive allocation letter
t	Select TAPE emulation mode.
y	Change overwrite mode for tape emulation.
a,b,c,d	Select virtual floppy disks on the .md3 hard disk image file
CS+A,B,C,D	Rename the virtual floppy disk drive
SS+A,B,C,D	Toggle write-protection for the virtual floppy disk drive
W	Select a different .md3 hard disk image file
Q	Exit
E	Shows Spectrum screen from when the NMI button was pressed (key must be held down)
F	Search for a file
CS+F	Search for a file - (case sensitive)
S	Return to BASIC and save a SNAPSHOT
V	Start the monitor
P	Return to BASIC and save screen
I	Try to return to BASIC
L	Play audio CD attached to interface
H	Helpscreen

MDOS3 Command Reference

```
CLS attr
```

Sets the INK, PAPER and COLOUR by reference to the value of attr - for example, `CLS 7` sets black paper and border, with white ink.

```
LIST *
```

This command provides background information on MDOS3.

```
CAT "x:mask"
```

This command produces a directory of the disk (or drive) where x is the name of the disk (or drive), and the mask is the DOS like wildcard to select filenames.

The following examples will therefore work:
`CAT "a:"` - provide a directory of all files on the A: drive.
`CAT "a:test*.*"` - provide a directory of all files on the A: drive which begin with the letters test.
`CAT "a:*.P"` - provide a directory of all BASIC files on the A: drive.
`CAT "a:*.B"` - provide a directory of all the code files on the A: drive.
`CAT "test: "` - provide a directory of all the files on the virtual floppy disk drive named test.

```
LOAD *"file"
LOAD *"x:file"
LOAD *"file" CODE
LOAD *"x:file" CODE
```

This command loads the specified file in the same way as on the Spectrum. You can specify the drive letter if you do not wish to use the current drive. The asterisk (*) is optional.

```
MERGE *"file"
MERGE *"x:file"
```

This command merges the specified file with the one currently in memory. You can specify the drive letter if you do not wish to use the current drive.

```
SAVE *"file"
SAVE *"x:file"
SAVE *"file" CODE
SAVE *"x:file" CODE
SAVE *"file" RUN
SAVE *"x:file" RUN
```

This command saves the specified file. You can specify the drive letter if you do not wish to use the current drive. The asterisk (*) is optional.

```
RUN *"file"
RUN !"file"
RUN "file"
```

These all set RAMTOP according the start address of the file (stored in Bytes type - "file.b") - 1 and then starts the program from that address. If the file is not found, then RAMTOP remains the same.

These commands are equivalent to:

```
  CLEAR adr-1: LOAD *"file" CODE: RANDOMIZE USR adr
```

```
CLEAR *"file"
CLEAR !"file"
CLEAR "file"
```

These are similar to the `RUN` command - they set RAMTOP according the start address of the file (stored in Bytes type - "file.b") - 1 and then load the program to that address. If the file is not found, then RAMTOP remains the same. The program is not started.

These commands are equivalent to:

```
  CLEAR adr-1: LOAD *"file" CODE
```

```
GO TO *"file"
GO TO !"file"
GO TO "file"
```

These are similar to the `RUN` command, except they do not alter RAMTOP. They merely load the program and then start it.

These commands are equivalent to:

```
  LOAD *"file" CODE: RANDOMIZE USR adr
```

```
MOVE "x:mask", "y:"
MOVE "x:mask", "y:file"
```

This command is used to copy disks. x and y are the names of the disk (or drive), and the mask is the DOS like wildcard to select filenames. If x and y are the same, then the mask cannot contain wildcards. file is the destination filename - used when copying files on the same disk.

The following examples will therefore work:

`MOVE "a:*.*", "b:"` - copy all files on drive a: to drive b:

`MOVE "a:*.scr", "b:"` - copy all files ending in .scr from drive a: to drive b:

`MOVE "HRY1:a.*", "HRY60:"` - copy all files beginning with the letter a from the disk with the name HRY1 to the disk with the name HRY60 (both disks must be on the same drive)

```
ERASE *"file"
ERASE *"x:file"
```

This command deletes the specified file from disk. You can specify the drive letter if you do not wish to use the current drive.

```
FORMAT *"x:name"
```

This command formats the specified disk and gives it the specified name.

```
READ *"x: ",sector,address
```

This command reads a specified sector on the given drive (x) to the specified address in memory.

MDOS3 Programming

It is possible to access some of the features of the operating system from machine-code programs, so that utility programs which access files can be written.

Calls are available in *MDOS3* using *hook codes*, which are invoked by paging into DivIDE memory and then calling the hook routine $24, with the call ID in A and parameters in other registers

To page into DivIDE Memory

```
.divIDE_page
         ld       a,$4F
         ld       de,TAB-26
         call     $25AB
         ld       hl,0
         ld       (TAB),hl
         ld       hl,$3EF7
         ld       (TAB+2),hl
         rst      0
         ret

TAB      dw       0
         dw       $3EF7
```

Now call the hook routine

```
         ld       a,($3E6B)
         ld       e,a
         ld       a,4                  ; GET_PART_TABLE call
```

Then page back in the ZX ROM

```
ZXROM    ld       a,32
         ld       (16119),a
         jp       $1700
```

Reading data from a virtual floppy disk is done the same way as reading from physical drive. The routines DREAD and DWRITE are changed accordingly by MDOS3 - it is pointless trying to use lower level access. Errors are caught in the same way as on real media.

If you wish to use these service hook codes, you will need to ensure that the user presses the NMI button after booting MDOS3 so that all of the partition and device data has been stored within the system. GET_VER can be used to check status of MDOS3 initialisation.

The MDOS3 Service Hook Codes

MDOS3 provides various service hook codes to access facilities and hardware features and can be used without knowing anything about the interface on which MDOS is running.

MDOS3 provides the following service hook codes:

- GET_VER
- GET_MAPTAB
- GET_GEOM
- GET_AKTPAR
- GET_PART_TABLE
- DRIVEPART
- PREPOCET

- CMMAPTAB
- DRVCMPS
- DIVIDE0
- INC32
- DEC32
- ADD32HL
- DEC32HL
- ADD1693
- DEC1693
- NEXTDISK
- PREVDISK
- NASDRNMI
- BACKSCR
- RESTSCR
- SET_BOOTDISK
- RES_BOOTDISK
- CLSMAPTA
- DRVSELS
- READHDD
- WRITEHDD
- TST_BSY
- READIDATA
- READIDATAPI

GET_VER (0)

Fetches the version of MDOS3 - the version is the date of the build across the L, H and DE registers. If A=0 and the z flag is set, then MDOS3 was not initialised and the services below marked with asterisk (*) cannot be used - the tables generated by the NMI button are empty.

IN:	-
OUT:	L = day, H=month, DE = year, A =init MDOS3

Register status on return: z flag suggests MDOS3 not initialised.

GET_MAPTAB (1)

Returns the address of the MAPTAB (drive mapping) table. This is the main routine for getting details about drives allocated to the system. The entries in the drive mapping table for each drive can be set into BCDE for use with the other hook codes.

IN:	-
OUT:	HL = address of table.

Register status on return: none

GET_GEOM (2)

Returns the address of the GEOM (drive geometry) table.

IN: -

OUT: HL = address of table.

Register status on return: none

GET_AKTPAR (3)

Returns the address of the AKTPAR (active partition) table.

IN: -

OUT: HL = address of table.

Register status on return: none

GET_PART_TABLE (4)

Returns the address of the PART_TABLE (detected partitions) table.

IN: -

OUT: HL = address of table.

Register status on return: none

DRIVEPART (5*)

This returns details of the partition which contains the selected virtual floppy disk identified by the specified drive number.

IN: E = drive number (0-3)

OUT: IX = address of partition details related to floppy in selected drive.
A - partition number
Flag Z is set if

Register status on return: Flag Z is set if drive is empty.
Changes BC,HL and DE

PREPOCET (6)

This updates the drive system variables address to the MAPTAB address.

IN:	IX = $3E00 (drive system variables address)
OUT:	IX = address of MAPTAB table for appropriate drive

Register status on return:	None

CMMAPTAB (7)

This is alternative way of finding the MAPTAB table for a given drive.

IN:	E = drive number (0-3)
OUT:	IX = address of MAPTAB table for appropriate drive

Register status on return:	None

DRVCMPS (8)

This returns the parameters for a specified floppy disk drive. The parameters are set during the drive mapping process.

IN:	E = drive number (floppy disk 0, or floppy disk 1)
OUT:	IX = address of drive parameters

Register status on return:	BC Changed

DIVIDE0 (9)

This performs 32 bit division on a specified number (DEHL/C). DEHL is the higher part of an 4 byte register, followed by the lower part of the 4 byte register (useful for dealing with LBA offsets)

IN:	DEHL = number
	C = divider

OUT:	DE = result
	A = remainder

Register status on return:	B Changed

INC32 (10)

This increases the specified four byte register by one (useful for dealing with LBA offsets).

IN: BCDE = 32 bit number

OUT: BCDE= result

Register status on return: Flag Z is set if BCDE=0

DEC32 (11)

This decreases the specified four byte register by one (useful for dealing with LBA offsets).

IN: BCDE = 32 bit number

OUT: BCDE= result

Register status on return: Flag Z is set if BCDE=0
Flag NC is set if BCDE = -1

ADD32HL (12)

This adds HL to the specified four byte register (useful for dealing with LBA offsets).

IN: BCDE = 32 bit number
HL = 16 bit number

OUT: BCDE= result

Register status on return: Flag C is set for overflow

DEC32HL (13)

This deducts HL from the specified four byte register (useful for dealing with LBA offsets).

IN: BCDE = 32 bit number
HL = 16 bit number

OUT: BCDE= result

Register status on return: Flag C is set for overflow

ADD1693 (14)

This adds 1693 to the specified four byte register (useful for dealing with LBA offsets).

IN: BCDE = 32 bit number

OUT: BCDE= result

Register status on return: Flag C is set for overflow

DEC1693 (15)

This deducts 1693 from the specified four byte register (useful for dealing with LBA offsets).

IN: BCDE = 32 bit number

OUT: BCDE= result

Register status on return: Flag C is set for overflow

NEXTDISK (16*)

This moves the pointer in BCDE to the start of the next virtual disk on the current drive.

IN: BCDE = 32 bit number as pointer to virtual disk
IX = pointer to partition table linked to the drive on which the virtual disk is located = see GET_PART_TABLE

OUT: BCDE= result

Register status on return: Flag C is set if the pointer has reached the end of the partition
HL Changed

PREVDISK (17*)

This moves the pointer in BCDE to the start of the previous virtual disk on the current drive.

IN:	BCDE = 32 bit number as pointer to virtual disk IX = pointer to partition table linked to the drive on which the virtual disk is located = see GET_PART_TABLE
OUT:	BCDE= result

Register status on return: Flag NC is set if the pointer is at the start of the partition
HL Changed

NASDRNMI (18)

This resets the MDOS3 system - for example, if you change the mapping in the MAPTAB table, you will need to call this to update the drive parameters. This call re-sets the MDOS3 system in accordance with the latest MAPTAB, returns the physical heads of the disk drives back to the beginning.

IN:	none
OUT:	none

Register status on return: HL, BC, DE, IX Changed

BACKSCR (19)

This copies the VRAM1 memory (the Spectrum's screen) into DivIDE memory.

IN:	none
OUT:	none

Register status on return: None

RESTSCR (20)

This copies the DivIDE memory back into VRAM1 memory (the Spectrum's screen).

IN:	none
OUT:	none

Register status on return: None

SET_BOOTDISK (21)

This takes the disk from which the boot was performed and sets that as the specified drive number. The MAPTAB table is changed, but not the correct mapping on a hard disk drive. The original disk setting is memorised for use with the RES_BOOTDISK call.

IN:	E = drive number (0-3)
OUT:	none
Register status on return:	HL, BC, DE and IX changed

RES_BOOTDISK (22)

This reverses the action of SET_BOOTDISK, re-setting the specified drive number and the MAPTAB table.

IN:	E = drive number (0-3)
OUT:	none
Register status on return:	HL, BC, DE and IX changed

CLSMAPTA (23)

This restores the MAPTAB and AKTPAR tables to their original values when the DivIDE Plus was first switched on.

IN:	none
OUT:	none
Register status on return:	HL, BC, DE and IX changed

DRVSELS (24)

This starts the specified physical drive (floppy disk 0 or 1) spinning up to speed.

IN:	E = physical disk drive number (0-1)
OUT:	none
Register status on return:	none

READHDD (25*)

This reads a specified sector detailed by the 32 bit number in BCDE. The number is converted to CHS using the disk geometry. The error number corresponds to the output of the DREAD routine. To read data it is recommended that you set a disk into a drive letter and then use the floppy sector reading routines (DREAD and BREAD).

IN: BCDE = 32 bit number of the LBA sector - the 4th bit of the 4th byte is used to select the master/slave device
HL is the address of the space to store the sector

OUT: BC = error number
If successful, then HL = HL +512

Register status on return: HL,BC,DE and IX are updated

WRITEHDD (26*)

This writes a specified sector detailed by the 32 bit number in BCDE. The number is converted to CHS using the disk geometry. The error number corresponds to the output of the DWRITE routine. To write data it is recommended that you set a disk into a drive letter and then use the floppy sector reading routines (DWRITE and BWRITE).

IN: BCDE = 32 bit number of the LBA sector - the 4th bit of the 4th byte is used to select the master/slave device
HL is the address of the space to store the sector

OUT: BC = error number
If successful, then HL = HL +512

Register status on return: HL,BC,DE and IX are updated

TST_BSY (27)

This tests to see if the busy status register is clear - it will wait around 2 seconds if the drive is busy. All Hard disk read/write commands use this automatically.

IN: BCDE = 32 bit number of the LBA sector - the 4th bit of the 4th byte is used to select the master/slave device
OUT: A = 0 + flag Z (ready)
or
A=255 + flag NZ (busy)

Register status on return: Z and NZ flag are updated

READIDATA (28)

This reads the ID sector from an ATA device to the specified address. The error number corresponds to the output of the DREAD routine.

IN:	BCDE = 32 bit number of the LBA sector - the 4th bit of the 4th byte is used to select the master/slave device
	HL = address for storage
OUT:	BC = error number

Register status on return: None

READIDATAPI (29)

This reads the ID sector from an ATAPI device (such as a CD) to the specified address. The error number corresponds to the output of the DREAD routine.

IN:	BCDE = 32 bit number of the LBA sector - the 4th bit of the 4th byte is used to select the master/slave device
	HL = address for storage
OUT:	BC = error number

Register status on return: None

Table definitions:

MAPTAB

This is the drive mapping table - it defines whether a physical drive is mapped to floppy disk 0, floppy disk 1 or a (virtual) hard disk.

```
MAPTAB        db 2,0,0,0,0  ;drive A:
              db 3,0,0,0,0  ;drive B:
              db 4,0,0,0,0  ;drive C:
              db 5,0,0,0,0  ;drive D:
              db 3          ;TAPE emulation indicator
              db 0          ; Rewrite mode enabled?
```

The first four rows (drive A to drive B) define the drives. If the first byte contains 0 or 1, then this is the physical floppy disk fd0 or fd1.

If however, the first byte in a row contains 2 to 5, the drive is mapped to a hard disk and the next four bytes contain the number of the LBA sector on that drive, where the floppy disk image is stored (pointing to the infosector). The number is stored from the lower to the higher byte. If the 4th bit is set in the last byte, then this indicates a slave hard disk device.

The next row is then the TAPE emulation. 1 = LOAD, 2=SAVE, 3=LOAD/SAVE

The final row indicates the rewrite mode. 0 = Ask for confirmation before overwriting an existing file. 1 = Do not overwrite. 2 = Overwrite without confirmation.

GEOM

This is the geometry of the master and slave devices.

```
GEOM          db $D2,$03,$08,$20          ;master
              db $EA,$01,$02,$20          ;slave
```

The two rows define the geometry for each hard disk device. Bytes 0-1 contain the number of cylinders, byte 2 contains the number of heads and byte 3 the number of sectors.

AKTPAR

This is the details of the current active partition (selected by the W key in the NMI menu).

```
AKTPAR        db 1                        ;number of the partition
              db $20,0,0,0                ;start of the partition
              db $E0,$D1,$03,0            ;length of the partition
```

The first row defines the number of the partition. 1-4 is a primary partition on the Master device, 5-8 is a primary partition on the Slave device. 254 indicates that the whole of the Master device is available. 255 indicates that the whole of the Slave device is available.

The next row then contain 4 bytes each equal the number of the LBA sector on that drive where the partition starts.

The final row contains four bytes to provide the length of the partition in LBA sectors - hence the last LBA sector can be found through beginning+length+1.

PART_TABLE

This is the details of all partitions found on the devices by MDOS3 during initialisation.

```
PART_TABLE    db 1                       ;number of the partition
              db $20,0,0,0               ;start of the partition
              db $E0,$D1,$03,0           ;length of the partition
.
.
.
              db 6                       ;number of the partition
              db $80,$3E,0,0             ;start of the partition
              db $0,$3C,0,0              ;length of the partition

              db 0                       ; Indentify end of partition list.
```

The table is formed by repeating details of the number of the partition, start and length for each partition found.

The first row defines the number of the partition. 0 indicates the end of the table.
1-4 is a primary partition on the Master device, 5-8 is a primary partition on the Slave device.
255 indicates that the whole of the Slave device is available and the first floppy disk image has been found by the user selecting the disk in the NMI menu
254 indicates that the whole of the Master device is available and the first floppy disk image has been found by the user selecting the disk in the NMI menu
253 indicates that the whole of the Slave device is available but has not yet been searched for the first floppy disk image.
252 indicates that the whole of the Master device is available but has not yet been searched for the first floppy disk image.

NOTE: if the partition number is 255 or 254, the start of the partition is actually a pointer to the start of the floppy disk image, and the length is the length of the floppy disk image.

Programming for DivIDE Plus

These notes are intended to assist developers of firmware and emulators to support the latest features. If you require any more specific details, then please do not hesitate to contact us directly.

Implementation of the 128K BASIC Mode on DivIDE Plus

The ZX Spectrum 128K has two ROM banks: ROM0 and ROM1. ROM1 is the basic one, used in BASIC48. ROM0 has menu, calculator, screen editor and extra functions specific to the 128K Spectrum (AY commands etc). The Original DivIDE Hard Disk Interface will cause a ZX Spectrum to crash whilst in 128K BASIC Mode (ROM0) - this appears to be due to a clash between the DivIDE TR-DOS traps #3D00...#3DFF and the ZX Spectrum's ROM0 code.

DivIDE Plus therefore addresses this problem by implementing its own latch (known as the B4 Latch) on the 4th bit of Port &7FFD (decoded A15, A14, A2, /IORQ, /WR), keeping the /M1 line coming from the Z80CPU high when ROM0 is selected (this disables the autopaging of the DivIDE). This provides support for the ZX Spectrum's 128K BASIC Mode, by ensuring that DivIDE Plus is not active when ROM0 is selected.

The B4 Latch is unaffected by the ZX Spectrum +3's `OUT` commands such as `#0FFD`, `#1FFD`, `#2FFD`, `#3FFD` and is also not affected by an `IN` such as on the internal Port `&7FFD` on the ZX Spectrum 128K.

However, the use of the B4 Latch to lock the /M1 line can lead to incompatibility with some 128K snapshots and the DivIDE Plus therefore defaults to traps being enabled in both ROM0 and ROM1 (as per the original DivIDE).

ROM1 is used by most software, as it contains the maths and tape routines, character set, code to read the keyboard and many more standard system calls. On the other hand, the contents of ROM0 differs a lot between the various 128K models (for example there are differences between the Sinclair 128K and Amstrad +2 starting at address #0562). Generally a 128K snapshot created with different ROM0 code will work on other Spectrums, provided it does not call any routines contained within ROM0.

In order to enable this 128K Mode, the traps can be disabled in ROM0, by sending

```
OUT 23, BIN 001xxxxx
```

With 128K Mode enabled, there is a software solution to loading 128K snapshots which access ROM0 on DivIDE Plus. In order to implement this, a fix is required to the firmware:

1. On the stack place BC then a pointer to the following commands OUT (C),B:POP BC:RET also placed on the stack below (8 bytes total).

```
 ------------ ------------ ------------ ------------ -----------

|  OUT (C),B  | POP BC: RET |     code    |      BC     |     jump    |

 ------------ ------------ ------------ ------------ -----------

^code                      ^SP                        ^original SP
```

2. Now set BC to #4xFD (x is the proper RAM bank), SP to the address of the stacked code address and jump to the DivIDE exit area. The RAM bank must be set before this call, as we only change the ROM bank here.

   This will first jump to OUT, change the ROM bank, restore BC and return to the main program.

Implementation of greater ROM/RAM space on DiviDE Plus

DivIDE Plus provides a total of 512K ROM and 512K RAM on the DivIDE Plus in order to enable much improved firmware to be written (this compares to the 8K ROM and 32K RAM available on the original DivIDE Hard Disk Interface).

We decided on 512K as this would allow ramdisk storage and the following additional firmware to be used (as well as the original divIDE firmware):
ResiDOS
CPM22QED MBD (CP/M compatible)
ZXVGS MBD

The DivIDE Plus interface is equipped with 512K SRAM (628512 SMD soldered) and 512K ROM (29F040 PLCC in a socket) with address and data lines corrected, so no dedicated .RAW files are needed to program firmware on another system. The RAM is battery backed.

The scheme chosen mixes MB-02 paging using the Port &17 with DivIDE paging using the &E3 Port. It is compatible with the original 8K firmware without amendment. You will however, need to obtain the ROM reprogramming software from ourselves, rather than use that provided for the original DivIDE, due to different procedures for programming the 29F040 chip than on the 28C64 used by the original DivIDE.

The &E3 Port is untouched. The &17 Port is 8bit, outputs are called Q0..Q7, as follows:
- Q6 and Q7 select the mode (DivIDE, RAM, ROM, reset).
- Q5 must be set to write-enable in RAM and ROM modes.
- Q0..Q4 select the memory banks.

We have also added a jumper to possibly enable extra facilities based on Q4. The jumper is set by default (open) allowing operation as above - however when closed (if Q4 is set, bit 4 of Port &7FFD replaces Q4 which gives the possibility of using custom ROM banks in the 128K Mode (pages 0..15 are used as ROM1, whilst pages 16..31 are used as ROM0 for ZX Spectrum 128K bank-switching).

How the DivIDE Modes Operate

At power up, Ports &E3, &17 and &7FFD (internal) are all zeroed. Bit 4 of Port &7FFD (B4 Latch) is set whilst the ZX Spectrum has ROM0 selected - this is to force the DivIDE ROM and traps to act on power-on, even when the ZX Spectrum wakes up with ROM0.

If you press the Spectrum's reset button, it is slightly different. Only Ports &17 and &7FFD (internal) are zeroed. Bit 4 of &7FFD (B4 Latch) is unchanged.

When the DivIDE is reset using software (ie. Port &17 is sent Q7=1,Q6=1), Port &E3, &17 and Internal &7FFD are all zeroed. Again bit 4 of Port &7FFD is cleared.

This means that at this stage, DivIDE Mode is selected and the code stored in offsets #06000..#07FFF in the 29F040 ROM is used to boot the computer.

DivIDE Mode (Send %000aaaax to Port &17)

This is the standard DivIDE Compatibility mode with all divIDE traps enabled, provided that the Spectrum is using ROM1 or bit 4 of Port &7FFD (B4 Latch) is set by power-on.
Bit Q5 (%x) and bit Q0 (%x) are both unused in this mode (should be set to 0).

Bits Q1..Q4 (%aaaa) select the ROM/RAM area for DivIDE operation - providing 16 separate areas of 32K ROM + 32K RAM.

Each area consists of a set of 4 x 8K ROM banks (ROMa0, ROMa1, ROMa2, and ROMa3) and 4 x 8K RAM banks (RAMa0, RAMa1, RAMa2, and RAMa3). Although you can access all 4 RAM banks, you can only address ROMa3. This enables you to install standard DivIDE firmware in ROMa3.

In MAPRAM mode (selected by Port &E3 as on the original DivIDE), RAMa3 is used as a replacement for ROMa3. You can still access the other 3 RAM banks (RAMa0, RAMa1 and RAMa2 as memory), however, RAMa3 is write protected. You cannot access the ROMa3 bank until you either turn the ZX Spectrum off and back on, or issue a Reset Command to Port &17 (pressing the Spectrum's reset button will have no effect as it does not zero Port &E3).

DivIDE Plus 128K Mode (Send %001xxxxx to Port &17)

This enables the 128K BASIC Mode - see above. Only Bit Q5 is required to be set to 1 - all other bits (Q0...Q4) are ignored.

RAM Mode (Send %01waaaaa to Port &17)

This enables you to replace the ZX Spectrum ROM with the selected RAM page. All of the DivIDE traps are inactive, although you can continue to use IDE commands and Port &E3 (changes to Port &E3 will only take effect when DivIDE mode is selected, by Port &17 or reset button).

Bit Q5 (%w) can be set to provide a 64K RAM mode, allowing you to run software that requires 64K RAM in the Z80CPU address space (e.g. CP/M). If it is zero, then the RAM is write-protected.

Bits Q0..Q4 (%aaaaa) selects the RAM page to be used in place of the ZX Spectrum ROM - providing 32 separate pages of 16K.

ROM Mode (Send %10waaaaa to Port &17)

This enables you to replace the ZX Spectrum ROM with the selected ROM page. All of the DivIDE traps are inactive, although you can continue to use IDE commands and Port &E3.

Bit Q5 (%w) can be set to allow you to re-program the 29F040 Flash EEPROM chip. Please note that there is a special algorithm required to re-program this chip, therefore special software will need to be developed.

Bits Q0..Q4 (%aaaaa) selects the ROM page to be used in place of the ZX Spectrum ROM - providing 32 separate pages of 16K.

Reset Mode (Send %11xxxxxx to Port &17)

This resets divIDE Plus (during the next ROM access or the refresh cycle if I register is range 0..63). All other bits are ignored.

Port &17, Port &E3 and internal Port &7FFD are all zeroed. However bit 4 of Port &7FFD (B4 Latch) remains unchanged.

This will then force the ZX Spectrum to restart - if ROM1 was selected at the time, then the DivIDE Plus will restart in DivIDE Mode (as if you had just turned on the Spectrum). However, if ROM0 was selected at the time, you will just be presented with the Spectrum 128 selection menu (Tape Loader, 128 BASIC, Calculator, 48 BASIC) as DivIDE Plus will be disabled.

The Spectrum's 128K banking latch lock (bit 5 of &7FFD Port) can be cleared in software by the reset instruction on Port &17, allowing you to test for ZX Spectrum +3 features (namely the floppy disk) to see if they exist. However be warned that on a standard ZX Spectrum 128K these OUT commands can crash the computer, as they go to Port &7FFD